TinyML

Книга рецептов

Джан Марко Йодиче

ДМК ПРЕСС
издательство

Более 50 примеров разработки интеллектуальных приложений с использованием возможностей машинного обучения

TinyML – технология, призванная расширить использование искусственного интеллекта за счет устройств с малым энергопотреблением, таких как микроконтроллеры. Прочитав эту книгу, вы сможете свободно использовать передовые практики и фреймворки ML в своей работе.

Среди рассматриваемых тем:

- *принципы программирования микроконтроллеров;*
- *основы развертывания приложений на Arduino Nano 33 BLE Sense и Raspberry Pi Pico;*
- *внедрение приложения, реагирующего на человеческий голос, с помощью Edge Impulse;*
- *использование трансфертного обучения для классификации внутренних помещений с помощью Arduino Nano 33 BLE Sense;*
- *создание приложения для распознавания жестов с помощью Raspberry Pi Pico;*
- *разработка модели CIFAR-10 для микроконтроллеров с ограниченным объемом памяти;*
- *запуск классификатора изображений на виртуальном Arm Ethos-U55 microNPU с microTVM.*

Издание предназначено для инженеров-разработчиков, заинтересованных в создании приложений машинного обучения на микроконтроллерах. Требуется базовое знакомство с языками C/C++, Python и интерфейсом командной строки (CLI); предварительные знания о микроконтроллерах не обязательны.

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
books@alians-kniga.ru

Packt>

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО
www.дмк.рф

ISBN 978-5-93700-169-6

Джан Марко Йодиче

TinyML. Книга рецептов

Gian Marco Iodice

TinyML Cookbook

Combine artificial intelligence and ultra-low-power embedded devices to make the world smarter

BIRMINGHAM—MUMBAI

Джан Марко Йодиче

TinyML. Книга рецептов

Искусственный интеллект и интегрированные устройства со сверхнизким энергопотреблением делают мир умнее

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО
Москва, 2023

УДК 004.04
ББК 32.372
И75

Йодиче Дж. М.
И75 TinyML. Книга рецептов / пер. с англ. Ю. В. Ревича. – М.: ДМК Пресс, 2023. – 298 с.: ил.

ISBN 978-5-93700-169-6

TinyML – технология, призванная расширить использование искусственного интеллекта за счет устройств с малым энергопотреблением, таких как микроконтроллеры. Прочитав эту книгу, вы сможете свободно использовать передовые практики и фреймворки ML в своей работе. Для начала вы ознакомитесь с основами развертывания интеллектуальных приложений на Arduino Nano 33 BLE Sense и Raspberry Pi Pico, а затем на примере работы с реальными датчиками получите необходимые навыки для внедрения комплексных интеллектуальных приложений в различных сценариях.

Издание предназначено для инженеров-разработчиков, заинтересованных в создании приложений машинного обучения на микроконтроллерах. Требуется базовое знакомство с C/C++, языком программирования Python и интерфейсом командной строки (CLI); предварительные знания о микроконтроллерах не обязательны.

УДК 004.04
ББК 32.372

ISBN 978-1-80181-497-3 (англ.)
ISBN 978-5-93700-169-6 (рус.)

Мне удалось завершить это уникальное путешествие в течение долгих ночей только благодаря поддержке моей жены Элеоноры. Я посвящаю эту книгу ей, с самого начала верившей в этот проект.

Содержание

От издательства 13

Введение 14

Составители 16

Предисловие 18

Глава 1. Начало работы с TinyML 25

Технические требования 25
Представление TinyML 26
Что такое TinyML? 26
Почему ML на микроконтроллерах? 26
Зачем запускать ML локально? 27
Возможности и проблемы TinyML 28
Среды развертывания для TinyML 28
tinyML Foundation 30
Краткое описание глубокого обучения (DL) 30
Глубокие нейронные сети 31
Сверточные нейронные сети 32
Квантизация 35
Разница между мощностью и энергией 36
Различие между напряжением и током 36
Мощность и энергия 38
Программирование микроконтроллеров 39
Архитектура памяти 42
Периферийные устройства 43
Вход/выход общего назначения (GPIO или IO) 44
Аналого-цифровые преобразователи 45

Последовательная связь 45
Таймеры 45
Представление Arduino Nano 33 BLE Sense и Raspberry Pi Pico 45
Настройка Arduino Web Editor, TensorFlow и Edge Impulse 47
Подготовка веб-редактора Arduino Web Editor 47
Подготовка TensorFlow 47
Подготовка Edge Impulse 49
Как это делается 49
Запуск скетча на Arduino Nano 33 и Raspberry Pi Pico 51
Подготовка 51
Как это делается 52

Глава 2. Прототипирование на микроконтроллерах 53

Технические требования 53
Отладка кода 54
Подготовка 54
Как это делается 55
Дополнительно 58
Подключение светодиодного индикатора на макетной плате 58
Подготовка 58
Размещение прототипа на макетной плате 60
Как это делается 62
Управление внешним светодиодом с помощью GPIO 65
Подготовка 65
Представляем периферийное устройство GPIO 69
Как это делается 70
Включение и выключение светодиода с помощью кнопки 74
Подготовка 74
Как это делается 76
Использование прерываний для считывания состояния кнопки 79
Подготовка 79
Как это делается 80
Питание микроконтроллеров от батарей 82
Подготовка 82
Увеличение выходного напряжения при последовательном подключении батарей 83
Увеличение энергетической емкости за счет параллельного подключения батарей 83
Подключение батарей к плате микроконтроллера 84
Как это делается 85
Дополнительно 86

Глава 3. Создание метеостанции с помощью библиотеки TensorFlow Lite for microcontrollers 88

Импорт данных о погоде из WorldWeatherOnline 89

Подготовка ... 89
Как это делается ... 90
Подготовка набора данных ... 92
Подготовка ... 92
Подготовка сбалансированного набора данных ... 92
Масштабирование параметров с помощью Z-score ... 93
Как это делается ... 94
Обучение модели с помощью TF ... 98
Подготовка ... 98
Как это делается ... 99
Оценка эффективности модели ... 104
Подготовка ... 104
Наглядное представление эффективности с помощью матрицы ошибок ... 104
Оценка полноты (recall), точности (precision) и критерия F-score ... 106
Как это делается ... 106
Квантизация модели с помощью конвертера TFLite ... 108
Подготовка ... 109
Квантизация входной модели ... 109
Как это делается ... 112
Использование встроенного датчика температуры и влажности на Arduino Nano ... 114
Подготовка ... 114
Как это делается ... 115
Использование датчика DHT22 с Raspberry Pi Pico ... 116
Подготовка ... 116
Как это делается ... 117
Подготовка входных характеристик для просчета модели ... 119
Подготовка ... 119
Как это делается ... 120
Запуск на устройстве с помощью TFLu ... 122
Подготовка ... 122
Как это делается ... 123

Глава 4. Голосовое управление светодиодами с помощью Edge Impulse ... 127

Технические требования ... 128
Сбор аудиоданных с помощью смартфона ... 128
Подготовка ... 129
Сбор звуковых семплов для KWS ... 129
Как это делается ... 129
Извлечение параметров MFCC из аудиосемплов ... 134
Подготовка ... 134
Анализ звука в частотной области ... 134
Генерация Mel-спектрограммы ... 136

Извлечение MFCC 137
Как это делается 138
Дополнительно 140
Пример проектирования и обучения нейронной сети (NN) 142
Подготовка 142
Как это делается 142
Настройка эффективности модели с помощью EON Tuner 144
Подготовка 144
Как это делается 145
Классификация в реальном времени с помощью смартфона 147
Подготовка 147
Как это делается 147
Классификация в реальном времени с помощью Arduino Nano 149
Подготовка 149
Как это делается 149
Непрерывное распознавание на Arduino Nano 151
Подготовка 151
Изучение примера приложения KWS в реальном времени 151
Как это делается 153
Схема для голосового управления светодиодами на Raspberry Pi Pico 157
Подготовка 157
Представляем модуль электретного микрофона с усилителем MAX9814 158
Подключение микрофона к АЦП Raspberry Pi Pico 159
Как это делается 159
Выборка звука на Raspberry PI Pico с помощью АЦП и прерываний по таймеру 164
Подготовка 164
Выборка звука на Raspberry PI Pico с помощью АЦП и прерываний по таймеру 164
Как это делается 165
Дополнительно 169

Глава 5. Распознавание интерьеров помещений с помощью TensorFlow Lite for Microcontrollers и Arduino Nano 171

Технические требования 172
Съемка с помощью модуля камеры OV7670 172
Подготовка 173
Как это делается 173
Захват кадров камеры через последовательный порт с помощью Python 176
Подготовка 177
Передача изображений RGB888 через последовательный порт 177
Изучаем, как преобразовать RGB565 в RGB888 179
Как это делается 179

Преобразование изображений QQVGA из YCbCr422 в RGB888 183
Подготовка 183
Преобразование YCbCr422 в RGB888 184
Как это делается 184
Создание набора данных для распознавания интерьеров помещений 186
Подготовка 186
Как это делается 187
Трансфертное обучение с помощью Keras API 189
Подготовка 189
Изучение вариантов дизайна сети MobileNet 190
Как это делается 191
Подготовка и тестирование квантизованной модели TFLite 194
Подготовка 195
Как это делается 195
Сокращение объема RAM за счет объединения функций обрезки, изменения размера, масштабирования и квантизации 197
Подготовка 198
Изменение размера с помощью билинейной интерполяции 199
Как это делается 200

Глава 6. Создание интерфейса на основе жестов для управления воспроизведением на YouTube 206

Технические требования 207
Подключение к MPU-6050 IMU через интерфейс I2C 207
Подготовка 208
Представляем MPU-6050 IMU 208
Связь с помощью I2C 209
Как это делается 211
Получение данных акселерометра 214
Подготовка 214
Как это делается 217
Построение набора данных с помощью инструмента пересылки данных Edge Impulse data forwarder 220
Подготовка 221
Как это делается 222
Разработка и обучение модели ML 225
Подготовка 225
Использование спектрального анализа для распознавания жестов 226
Как это делается 228
Классификации в реальном времени с помощью инструмента пересылки данных Edge Impulse data forwarder 231
Подготовка 231
Как это делается 231
Распознавание жестов на Raspberry Pi Pico в ОС Arm Mbed 232
Подготовка 232

Создание рабочих потоков с помощью RTOS API в Arm Mbed OS 233
Фильтрация избыточных и ложных прогнозов 234
Как это делается..... 235
Создание бесконтактного интерфейса с помощью PyAutoGUI 241
Подготовка 241
Как это делается..... 242

Глава 7. Запуск модели TinyML CIFAR-10 на виртуальной платформе ОС Zephyr 244

Технические требования..... 245
Начало работы с ОС Zephyr 245
Подготовка 245
Как это делается..... 246
Разработка и обучение малой модели CIFAR-10 248
Подготовка 249
Замена свертки 2D на DWSC 249
Контроль поддержки требований модели к памяти 251
Как это делается..... 252
Оценка достоверности модели TFLite 255
Подготовка 256
Как это делается..... 256
Преобразование цифрового изображения в C-байтовый массив 258
Подготовка 258
Как это делается..... 259
Подготовка основы проекта TFLu..... 261
Подготовка 261
Как это делается..... 262
Создание и запуск приложения TFLu на QEMU..... 263
Подготовка 264
Как это делается..... 264

Глава 8. К следующему поколению TinyML с microNPU 269

Технические требования..... 270
Настройка Arm Corstone-300 FVP 270
Подготовка 270
Как это делается..... 272
Установка TVM с поддержкой Arm Ethos-U..... 274
Подготовка 275
Мотивация, лежащая в основе TVM..... 275
Как TVM оптимизирует работу модели 275
Как это делается..... 277
Установка набора инструментов Arm и стека драйверов Ethos-U..... 279
Подготовка 280
Как это делается..... 280

Генерация C-кода с помощью TVM 282
Подготовка 283
Запуск TVM на микроконтроллерах с помощью microTVM 284
Как это делается 284
Генерация C-байтовых массивов для входа, выхода и меток 286
Подготовка 286
Как это делается 288
Создание и запуск модели на Arm Ethos-U55 291
Подготовка 291
Как это делается 291

Предметный указатель 295

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Введение

Без сомнения, индустрия высоких технологий продолжает оказывать все большее влияние на нашу повседневную жизнь. Изменения столь же стремительны, сколь и постоянны, и происходят повсюду вокруг нас – в наших телефонах, автомобилях, интеллектуальных динамиках и микрогаджетах, которые мы используем для повышения эффективности, комфорта и возможностей коммуникации. Машинное обучение (*machine learning*, ML) – одна из самых преобразующих технологий нашего времени. Предприятия, ученые и инженерные сообщества продолжают углублять понимание, развивать и исследовать возможности этой невероятной технологии и служат все большему раскрытию ее потенциала для создания новых вариантов использования во многих отраслях.

Я менеджер по продуктам машинного обучения в компании ARM. В этой роли я нахожусь в центре революции ML, которая происходит в смартфонах, автомобильной промышленности, играх, AR, VR[1] и других областях. Мне ясно, что в ближайшем будущем функциональность ML будет в каждом отдельном электронном устройстве, – от крупнейших в мире суперкомпьютеров до самых маленьких и маломощных микроконтроллеров. Работа в области ML познакомила меня с некоторыми из самых блестящих и ярких умов в области технологий – теми, кто бросает вызов традиции, задает сложные вопросы и открывает новые ценности благодаря использованию ML.

Когда я впервые встретил Джана Марко, я едва мог произнести «ML», но в то время он уже был ветераном в этом космосе. Я был поражен широтой и глубиной его знаний и его способностью решать сложные проблемы. Вместе с командой ARM он работал над созданием Arm Compute Library (ACL) – самой эффективной библиотеки, доступной для ML на платформе ARM. Успех ACL не имеет себе равных. Он развернут на миллиардах устройств по всему миру – от серверов и флагманских смартфонов до интеллектуальных духовых шкафов.

Когда Джан Марко сказал мне, что пишет книгу о ML, моей немедленной реакцией было: «Какую часть?» Экосистема ML настолько разнообразна, что необходимо учитывать множество различных технологий, платформ и фреймворков[2]. Я знал, что благодаря обширным знаниям всех аспектов ML он был подходящим человеком для этой работы. Кроме того, Джан Марко обладает удивительной способностью объяснять вещи прямо и логично.

Книга Джана Марко раскрывает мир TinyML, проводя нас через ряд практических примеров из реального мира. Каждый пример изложен в форме рецепта, в четком и последовательном стиле, обеспечивающем простое по-

---

[1] AR (*augmented reality*) – дополненная реальность; VR (*virtual reality*) – виртуальная реальность. – *Здесь и далее прим. перев.*

[2] Фреймворк (*framework*) – программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

шаговое руководство. Начиная с базовых принципов, он растолковывает основы электронных или программных технологий, которые будут использоваться в примере. Затем в книге рассказывается об используемых платформах и технологиях, за которыми следуют основы ML, – разработка моделей нейронных сетей, проведение обучения и разворачивания моделей на целевом устройстве. Это действительно стиль кулинарного руководства по приготовлению супа с орехами. Каждый пример немного сложнее предыдущего, и в них удачно сочетаются традиционные и новые технологии. Вы не просто узнаете «как», вы также получите понимание «почему». Когда дело доходит до периферийных устройств, эта книга действительно дает панорамный обзор области ML.

Машинное обучение продолжает менять все аспекты технологий, и разработчикам программного обеспечения необходимо начинать его изучение. Эта книга позволяет быстро адаптироваться благодаря использованию легкодоступных и недорогих аппаратных средств. Независимо от того, являетесь ли вы новичком в ML или имеете некоторый опыт, каждый пример будет содержать новые знания и оставлять достаточно возможностей для саморазвития и дальнейших экспериментов. Независимо от того, используете ли вы эту книгу в качестве учебника или справочника, вы создадите прочную основу в области ML для будущего развития. Это даст вашей команде возможность достичь повышения эффективности и производительности, получить новые идеи и новые возможности для ваших продуктов.

Ронан Нотон,
старший менеджер по продуктам машинного обучения в ARM

Составители

Об авторе

Джан Марко Йодиче (Gian Marco Iodice) – командный и технологический лидер группы Machine Learning Group в компании ARM. В 2017 году он стал соавтором библиотеки ARM Compute Library (ACL). ACL в настоящее время является самой эффективной библиотекой для ML на процессорах ARM, она развернута на миллиардах устройств по всему миру – от серверов до смартфонов.

Джан Марко получил степень магистра с отличием в области электронной инженерии в Пизанском университете (Италия) и имеет многолетний опыт разработки алгоритмов ML и компьютерного зрения на периферийных устройствах. Теперь он руководит оптимизацией производительности ML на графических процессорах (GPU) ARM Mali.

В 2020 году Джан Марко стал соучредителем английского сообщества TinyML, возникшего с целью поощрения обмена знаниями, обучения и мотивации следующего поколения разработчиков ML на малогабаритных и энергоэффективных устройствах.

О рецензентах

Алессандро Гранде (Alessandro Grande) – физик, инженер, коммуникатор и технологический лидер, страстно желающий объединить людей и расширить их возможности для создания более эффективных и устойчивых технологий. Алессандро является директором по продуктам Edge Impulse и соучредителем сообщества TinyML в Великобритании и Италии. До прихода в Edge Impulse Алессандро работал в ARM евангелистом-разработчиком и менеджером экосистемы, уделяя особое внимание созданию основ более умного и эффективного интернета вещей (IoT). Он имеет степень магистра в области ядерной и электронной физики от Римского университета «Сапиенца».

Дакш Трехан (Daksh Trehan) начал свою карьеру в качестве аналитика, руководствуясь особой любовью к статистике и обработке данных. Различные

статистические методы познакомили его с миром ML и *data science*[3]. Хотя Дакш Трехан сосредоточен на анализе данных, он любит заниматься прогнозами с использованием ML. Он понимает значение данных в современном мире и постоянно пытается изменить мир, используя различные методы ML и свои навыки визуализации данных.

Дакш Трехан любит писать статьи о ML и искусственном интеллекте, и они принесли ему более 100 000 просмотров на сегодняшний день. Он также внес свой вклад в качестве консультанта по ML в книгу о TikTok, написанную доктором Маркусом Рейчем и доступную в магазине электронных книг Amazon.

[3] Data science (букв. наука о данных) – общее название широкого спектра дисциплин, занимающихся анализом и обработкой данных в различных областях бизнеса, науки или инженерных разработок. Само по себе название «Data scientist» в смысле обозначения профессии вы можете встретить крайне редко – обычно такие специализации называются по своим конкретным областям (напр., специалист по математической статистике – обработке результатов эксперимента и проверке статистических гипотез – или специалист по анализу данных). К «наукам о данных» относится в том числе и создание и внедрение алгоритмов машинного обучения, о которых идет речь в этой книге.

Предисловие

Эта книга о TinyML, быстрорастущей области на пересечении **машинного обучения (ML)** и встраиваемых систем, позволяющей искусственному интеллекту (ИИ) работать на устройствах на основе микроконтроллеров с чрезвычайно низким энергопотреблением.

TinyML – захватывающая область, полная возможностей. При небольшом бюджете мы можем создавать устройства, разумно взаимодействующие с окружающим миром и меняющие наш образ жизни к лучшему. Однако к этой области может быть трудно подступиться, если мы исходим из области ML и мало знакомы с микроконтроллерами. Цель этой книги – разрушить подобные барьеры и на практических примерах сделать TinyML доступным для разработчиков, не имеющих опыта программирования встраиваемых систем. Каждая глава будет представлять собой самостоятельный проект, позволяющий узнать, как использовать технологии, лежащие в основе TinyML, для взаимодействия с электронными компонентами (например, датчиками), и развертывать модели ML на устройствах с ограниченным объемом памяти.

«*Поваренная книга TinyML*» начинается с практического введения в эту междисциплинарную область, чтобы ознакомить вас с некоторыми основами развертывания интеллектуальных приложений на Arduino Nano 33 BLE Sense и Raspberry Pi Pico. По мере продвижения вы будете решать различные задачи, с которыми можете столкнуться при создании прототипов устройств на микроконтроллерах, вроде управления состоянием светодиода с помощью GPIO-вывода и кнопки или подачей питания на устройство с помощью батарей. После этого мы поговорим о примерах, касающихся измерителя температуры-влажности или датчиков **три-V** (**голос** (*voice*), **зрение** (*view*) и **вибрация** (*vibration*)), чтобы получить необходимые навыки для внедрения комплексных интеллектуальных приложений в различных сценариях. Затем вы изучите рекомендации по созданию уменьшенных моделей для микроконтроллеров с ограниченным объемом памяти. Наконец, вы познакомитесь с двумя самыми последними технологиями, microTVM и microNPU, которые помогут усовершенствовать ваши игры с TinyML.

К концу этой книги вы будете хорошо разбираться в лучших практиках и фреймворках ML, позволяющих разрабатывать ML-приложения на микроконтроллерах, и будете иметь четкое представление о ключевых аспектах, которые следует учитывать на этапе разработки.

Для кого предназначена эта книга

Эта книга предназначена для инженеров-разработчиков ML, заинтересованных в быстрой разработке приложений ML на микроконтроллерах с помощью практических примеров. Книга поможет вам расширить знания о революции TinyML, получить навыки создания комплексных интеллектуальных

проектов с использованием реальных датчиков данных на Arduino Nano 33 BLE Sense и Raspberry Pi Pico. Требуется базовое знакомство с C/C++, программированием на Python и **интерфейсом командной строки** (**CLI**). Однако никаких предварительных знаний о микроконтроллерах не требуется.

О ЧЕМ РАССКАЗЫВАЕТ ЭТА КНИГА

В главе 1 «*Начало работы с TinyML*» представлен обзор TinyML, а также возможности и проблемы, связанные с внедрением ML на микроконтроллерах с чрезвычайно низким энергопотреблением. В этой главе основное внимание уделяется фундаментальным элементам ML, энергопотреблению и микроконтроллерам, отличиям TinyML от обычного ML в облаке, на настольных компьютерах или даже смартфонах.

В главе 2 «*Прототипирование на микроконтроллерах*» представлены краткие и простые проекты, позволяющие разобраться с соответствующими основами программирования микроконтроллеров. Мы разберемся с отладкой кода и тем, как передавать данные на монитор последовательного порта Arduino. После этого мы узнаем, как программировать GPIO-периферию, управляющую выводами микроконтроллера, с помощью ARM Mbed API и использовать макетную плату для подключения внешних компонентов, таких как светодиоды и кнопки. В конце мы узнаем, как питать Arduino Nano 33 BLE Sense и Raspberry PI Pico от батареек.

Глава 3 «*Создание метеостанции с помощью библиотеки TensorFlow Lite for microcontrollers*» проведет вас через все этапы разработки приложения на основе библиотеки TensorFlow Lite for microcontrollers[4] и научит получать данные датчиков температуры и влажности. Приложение, которое будет разработано в этой главе, представляет собой метеостанцию на базе ML для прогнозирования снегопада.

Сначала мы сосредоточимся на подготовке набора данных путем получения исторических данных о погоде из WorldWeatherOnline. После этого мы представим соответствующие основы обучения и тестирования модели на основе библиотеки TensorFlow. В конце концов мы развернем модель на Arduino Nano 33 BLE Sense и Raspberry Pi Pico с библиотекой TensorFlow Lite for microcontrollers.

В главе 4 «*Голосовое управление светодиодами с помощью Edge Impulse*» показано, как разработать сквозное приложение для определения ключевых слов (**keyword spotting, KWS**) с помощью Edge Impulse и ознакомиться со сбором аудиоданных и аналого-цифровыми преобразователями (**АЦП**). Приложение, рассмотренное в этой главе, управляет цветом светодиода (красный, зеленый и синий) и количеством миганий (один, два и три).

---

[4] TensorFlow – библиотека машинного обучения, разработанная Google и ориентированная на обычные среды выполнения (серверы, облачные хранилища, ПК и смартфоны). В 2017 году была представлена специальная облегченная версия TensorFlow Lite для мобильных и малопотребляющих устройств с небольшим объемом памяти.

Сначала мы сосредоточимся на подготовке набора данных, показав, как получать аудиоданные с помощью мобильного телефона. После этого мы разработаем модель с использованием функций **MFCC**[5] и оптимизируем производительность с помощью EON Tuner. В конце концов мы доработаем приложение KWS на Arduino Nano 33 BLE Sense и Raspberry Pi Pico.

Глава 5 «*Распознавание интерьеров помещений с помощью TensorFlow Lite for microcontrollers и Arduino Nano*» призвана показать вам, как применять трансфертное обучение[6] с помощью TensorFlow и ознакомиться с лучшими практиками использования модуля камеры с микроконтроллером. Для целей этой главы мы разработаем приложение для распознавания интерьеров с помощью Arduino Nano 33 BLE Sense и модуля камеры OV7670.

В первой части мы увидим, как получать изображения с модуля камеры OV7670. После этого мы сосредоточимся на дизайне модели, применяя трансфертное обучение с помощью Keras[7] для распознавания кухонных и ванных комнат. В конце концов мы развернем квантизованную модель на Arduino Nano 33 BLE Sense с помощью TensorFlow Lite for microcontrollers.

Глава 6 «*Создание интерфейса на основе жестов для управления воспроизведением на YouTube*» направлена на разработку сквозного приложения для распознавания жестов с помощью Edge Impulse и Raspberry Pi Pico. Оно познакомит вас с инерциальными датчиками, научит использовать периферийные устройства I2C и создавать многопоточные приложения в Arm Mbed OS.

Сначала мы соберем данные акселерометра с помощью сборщика данных Edge Impulse data forwarder. После этого разработаем модель с использованием функций в частотной области для распознавания трех жестов. В конце концов мы развернем приложение на Raspberry Pi Pico и внедрим программу на Python с библиотекой PyAutoGUI для создания бесконтактного интерфейса для управления воспроизведением видео на YouTube.

В главе 7 «*Запуск модели TinyML CIFAR-10 на виртуальной платформе ОС Zephyr*» приведены рекомендации по созданию уменьшенных моделей для микроконтроллеров с ограниченным объемом памяти. В этой главе мы будем разрабатывать модель на основе набора данных для классификации изображений CIFAR-10 на виртуальном микроконтроллере на базе ARM Cortex-M3.

Сначала мы установим Zephyr, основной фреймворк, используемый в этой главе для выполнения нашей задачи. После этого разработаем малую квантизованную модель CIFAR-10 с библиотекой TensorFlow. Эта модель подойдет для микроконтроллера с объемом программной памяти всего 256 Кбайт и оперативной памятью 64 Кбайт. В конце концов мы создадим приложение

---

[5] MFCC (Mel-frequency cepstral coefficients) – сложный алгоритм анализа звуковых фрагментов с целью выделения характерных частот для распознавания речи. Подробнее об MFCC на русском языке см. https://habr.com/ru/post/140828/.

[6] Подробно о трансфертном обучении рассказывается в главе 5.

[7] Keras – высокоуровневый API, надстройка над библиотеками машинного обучения (в том числе Tensorflow), позволяющий реализовать их функциональность наиболее простым образом.

для классификации изображений с помощью TensorFlow Lite for microcontrollers и ОС Zephyr и запустим его на виртуальной платформе с помощью Quick Emulator (QEMU).

Глава 8 «*К следующему поколению TinyML с microNPU*» поможет вам ознакомиться с microNPU, новым классом процессоров для работы ML на периферийных устройствах. В этой главе мы будем запускать квантизованную модель CIFAR-10 на виртуальном контроллере ARM Ethos-U55 microNPU с помощью оптимизирующего компилятора TVM.

Сначала мы узнаем, как работает микропроцессор ARM Ethos-U55, и установим программные средства для сборки и запуска модели на фиксированной виртуальной платформе ARM Corstone-300. После этого мы будем использовать компилятор TVM для преобразования предварительно обученной модели TensorFlow Lite в код на языке Си. В конце мы покажем, как скомпилировать и развернуть код, сгенерированный TVM, в ARM Corstone-300 для выполнения вычислений с помощью ARM Ethos-U55 microNPU.

Как извлечь максимум пользы из этой книги

Вам понадобится компьютер (ноутбук или настольный компьютер) с архитектурой x86-64 и по крайней мере одним USB-портом для программирования плат Arduino Nano 33 BLE Sense и Raspberry Pi Pico. Для первых шести глав вы можете использовать Ubuntu 18.04 (или более позднюю версию) или Windows (например, Windows 10) в качестве операционной системы. Однако вам понадобится Ubuntu 18.04 (или более поздняя версия) для главы 7, посвященной запуску малой модели CIFAR-10 на виртуальной платформе с ОС Zephyr, и главы 8, посвященной следующему поколению TinyML с microNPU.

Необходимо иметь на вашем компьютере следующие программы:

- Python (рекомендуется Python 3.7),
- текстовый редактор (например, gedit в Ubuntu),
- медиаплеер (например, VLC),
- средство просмотра изображений (например, приложение по умолчанию в Ubuntu или Windows 10),
- веб-браузер (например, Google Chrome).

Для путешествия по TinyML нам понадобятся различные программные средства для разработки ML и программирования встроенных систем. Благодаря Arduino, Edge Impulse и Google эти инструменты будут находиться в облаке с доступом на основе браузера и с бесплатным планом использования.

Программы Arduino Nano 33 BLE Sense и Raspberry Pi Pico будут разрабатываться непосредственно в веб-браузере с помощью веб-редактора Arduino (https://create.arduino.cc). Однако бесплатный веб-редактор Arduino имеет ограничение в 200 с времени компиляции в день. Поэтому вы можете рассмотреть возможность перехода на любой платный тарифный план или на использование бесплатного локального редактора Arduino IDE (https://www.arduino.cc/en/software), чтобы получить неограниченное время компиляции. Если вас интересует бесплатная локальная среда разработки Arduino IDE, мы

предоставили на GitHub (https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Docs/setup_local_arduino_ide.md) инструкции по ее настройке.

В следующей таблице суммированы сведения об аппаратных устройствах и программных средствах, рассмотренных в каждой главе.

Глава	Аппаратные устройства	Программные средства
1	Arduino Nano 33 BLE Sense Raspberry Pi Pico	Arduino Web Editor
2	Arduino Nano 33 BLE Sense Raspberry Pi Pico	Arduino Web Editor, Google Colaboratory
3	Arduino Nano 33 BLE Sense Raspberry Pi Pico	Arduino Web Editor, Google Colaboratory
4	Arduino Nano 33 BLE Sense Raspberry Pi Pico	Arduino Web Editor, Edge Impulse Python 3.6 (local)
5	Arduino Nano 33 BLE Sense	Arduino Web Editor, Google Colaboratory Python 3.6 (local)
6	Raspberry Pi Pico	Arduino Web Editor, Edge Impulse Python 3.6 (local)
7	Виртуальная платформа	Google Colaboratory, Python 3.6 (local) Zephyr SDK
8	Виртуальная платформа	ARM Corstone-300, Python 3.6 (local) TVM/microTVM

Для проектов могут потребоваться датчики и дополнительные электронные компоненты для создания реалистичных прототипов TinyML и полного процесса разработки. Все компоненты перечислены в начале каждой главы и в файле *README.md* на GitHub (https://github.com/PacktPublishing/TinyML-Cookbook). Поскольку вы будете создавать настоящие электронные схемы, нам потребуется комплект электронных компонентов, который включает в себя по крайней мере беспаечную макетную плату, разноцветные светодиоды, резисторы, кнопки и соединительные провода-перемычки. Не волнуйтесь, если вы новичок в электронике, – вы познакомитесь с этими компонентами в первых двух главах этой книги. Кроме того, мы подготовили список покупок для начинающих на GitHub, чтобы вы точно знали, что купить: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Docs/shopping_list.md[8].

Если вы используете цифровую версию этой книги, мы советуем вам вводить код самостоятельно или получить доступ к коду через репозиторий GitHub (ссылка доступна ниже). Это поможет вам избежать любых потенциальных ошибок, связанных с копированием и вставкой кода.

[8] Указанный по ссылке список ориентирован на реалии США и Европы. Рекомендации по приобретению в российских условиях оборудования или компонентов будут размещаться в разделах «*Технические требования*» соответствующих глав. Отсутствие такой рекомендации означает, что компонент широко доступен и может быть приобретен без длительного поиска в основных отечественных интернет-магазинах электроники (http://www.chipdip.ru, http://iarduino.ru, https://www.electronshik.ru, https://dip8.ru и др.), а также на https://aliexpress.ru.

Загрузка файлов с примерами кода

Вы можете загрузить файлы с примерами кода для этой книги с GitHub по адресу https://github.com/PacktPublishing/TinyML-Cookbook. В случае обновления кода он также будет обновлен в существующем репозитории GitHub.

У нас также есть другие пакеты кода из нашего богатого каталога книг и видео, доступных по адресу https://github.com/PacktPublishing. Ознакомьтесь с ними!

Загрузка цветных изображений

Мы предоставляем PDF-файл, содержащий цветные изображения скриншотов и графиков, используемых в этой книге. Вы можете скачать его здесь: https://static.packt-cdn.com/downloads/9781801814973_ColorImages.pdf.

Текстовые соглашения

В этой книге используется ряд текстовых соглашений.

Гиперссылки, домены и интернет-адреса: выделяются жирным курсивом, если они не представляют собой законченный интернет-адрес с указанием протокола (URL). В последнем случае ссылка выделена синим шрифтом: https://github.com.

Имя файла: курсивом указаны имена папок, имена файлов, расширения файлов, пути.

Пример: «Войдите в папку *~/project_npu* и создайте три папки с именами *binaries*, *src* и *sw_libs*».

Также *курсивом* указываются названия глав и разделов и указываемые в скобках эквиваленты переводных терминов (напр. «узел (*node*)»).

`Код в тексте`: моноширинный шрифт указывает кодовые обозначения в тексте, команды, имена таблиц базы данных, фиктивные URL-адреса, вводимые пользователем, и дескрипторы Twitter.

`Блок кода` задается следующим образом:

```
export PATH=~/project_npu/binaries/FVP_Corstone_SSE-300/models/Linux64_GCC-6.4:$PATH
```

Когда мы хотим привлечь внимание к определенной части блока кода, соответствующие строки или элементы выделяются **`жирным`**:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Любой ввод или вывод из командной строки записывается следующим образом:

```
$ cd ~/project_npu
$ mkdir binaries
$ mkdir src
```

Жирный шрифт: также обозначает новый термин, важное слово или названия, которые вы видите на экране.

Например, названия в меню или диалоговых окнах отображаются в тексте следующим образом: «Нажмите на **Corstone-300 Ecosystem FVPs**, а затем нажмите на кнопку **Download Linux**». Русский перевод пунктов меню и названий кнопок, если это необходимо для лучшего понимания текста, указывается курсивом в скобках. Например, указанная фраза может заканчиваться следующим образом: «нажмите на кнопку **Download Linux** (*Загрузить Linux*)».

Важные примечания выглядят так.

Примечания выглядят вот так.

Подсказки выглядят вот так.

Часто используемые разделы

В этой книге вы найдете несколько часто появляющихся заголовков. Подробности инструкций по выполнению примеров помещены в следующие разделы.

Подготовка

В этом разделе рассказывается, чего ожидать от примера, и описывается, как настроить программное обеспечение или выполнить предварительные настройки, необходимые для его выполнения.

Как это делается...

В этом разделе приведены шаги, необходимые для выполнения примера.

Дополнительно

Этот раздел содержит дополнительную информацию о примере, чтобы вы лучше ориентировались.

Глава 1

Начало работы с TinyML

Мы делаем первый шаг в мир TinyML.

Глава начинается с обзора этой развивающейся области, в которой рассматриваются возможности и проблемы, связанные с внедрением машинного обучения (ML) в микроконтроллеры с чрезвычайно низким энергопотреблением.

Основная часть этой главы посвящена фундаментальным элементам ML, энергопотреблению и микроконтроллерам, уникальным особенностям TinyML и его отличиям от обычного ML в облаке, настольных компьютерах или даже смартфонах. В частности, раздел «*Программирование микроконтроллеров*» будет иметь существенное значение для тех, у кого мало опыта в программировании встроенных систем.

После представления основных строительных блоков TinyML мы настроим среду разработки для создания простого приложения управления светодиодами, что официально ознаменует начало нашего практического путешествия по TinyML.

В отличие от того, что мы найдем в следующих главах, эта глава имеет более теоретическую структуру, чтобы познакомить вас с концепциями и терминологией быстрорастущей технологии TinyML.

В этой главе мы рассмотрим следующие темы.

- Представление TinyML.
- Краткое описание глубокого обучения (DL).
- Разница между мощностью и энергией.
- Программирование микроконтроллеров.
- Представление Arduino Nano 33 BLE Sense и Raspberry Pi Pico.
- Настройка Arduino Web Editor, TensorFlow и Edge Impulse.
- Запуск скетча на Arduino Nano 33 и Raspberry Pi Pico.

Технические требования

Чтобы выполнить практический пример в этой главе, нам понадобится следующее:

- плата Arduino Nano 33 BLE Sense,
- плата Raspberry Pi Pico,
- кабель micro-USB,
- ноутбук или ПК с Ubuntu 18.04 или Windows 10 на x86-64.

Представление TinyML

Во всех проектах, представленных в этой книге, мы дадим практические решения для **малого машинного обучения**, или, как мы будем его называть, **TinyML**. В этом разделе мы узнаем, что такое TinyML и какие огромные возможности оно открывает.

Что такое TinyML?

TinyML – это набор технологий машинного обучения (ML) для использования интеллектуальных приложений на встраиваемых устройствах с чрезвычайно низким энергопотреблением. Как правило, эти устройства имеют ограниченную память и вычислительные возможности, но они могут воспринимать физические параметры среды с помощью датчиков и действовать на основе решений, принимаемых алгоритмами ML.

Для TinyML собственно ML и платформа, на которой оно разворачивается, – это не просто две независимые сущности, а скорее сущности, которые должны в лучшем случае соответствовать друг другу. На практике разработка архитектуры ML без учета характеристик целевого устройства усложняет развертывание эффективно работающих приложений TinyML.

С другой стороны, было бы невозможно спроектировать энергоэффективные процессоры для расширения возможностей ML таких устройств, не зная задействованных программных алгоритмов.

В этой книге в качестве целевого устройства для TinyML будут рассмотрены микроконтроллеры, а следующий подраздел поможет мотивировать наш выбор.

Почему ML на микроконтроллерах?

Первой и главной причиной выбора микроконтроллеров является их популярность в различных областях, таких как автомобилестроение, бытовая электроника, кухонная техника, здравоохранение и телекоммуникации. В настоящее время микроконтроллеры незаметно присутствуют во всех наших повседневных электронных устройствах.

С появлением интернета вещей (IoT) рынок микроконтроллеров стремительно рос. В 2018 году компания по исследованию рынка IDC (https://www.idc.com) сообщила о 28.1 млрд микроконтроллеров, проданных по всему миру, и прогнозируется рост до 38.2 млрд к 2023 году (www.arm.com/blogs/blueprint/tinyML). Это впечатляющие цифры, учитывая, что на рынках смартфонов и ПК в том же году было продано 1.5 млрд и 67.2 млн устройств соответственно.

Таким образом, TinyML представляет собой значительный шаг вперед для устройств IoT, способствуя распространению малых устройств, способных выполнять задачи ML локально.

Вторая причина выбора микроконтроллеров заключается в том, что они недороги, просты в программировании и достаточно мощны для запуска сложных алгоритмов глубокого обучения (DL).

Однако почему мы не можем перенести вычисления в облако, если мощные серверы гораздо более производительны? Другими словами, зачем нам нужно запускать ML локально?

Зачем запускать ML локально?

Есть три основных ответа на этот вопрос: «задержки», «энергопотребление» и «конфиденциальность».

- *Сокращение задержек*: отправка данных в облако и обратно не происходит мгновенно и может повлиять на приложения, которые должны надежно реагировать в кратчайшие сроки.
- *Снижение энергопотребления*: отправка и получение данных в облако и из него не являются энергоэффективными даже при использовании маломощных протоколов связи, таких как Bluetooth Low Energy (BLE). На следующей блок-схеме мы приводим распределение энергопотребления для встроенных компонентов платы Arduino Nano 33 BLE Sense, одной из двух плат микроконтроллеров, используемых в этой книге:

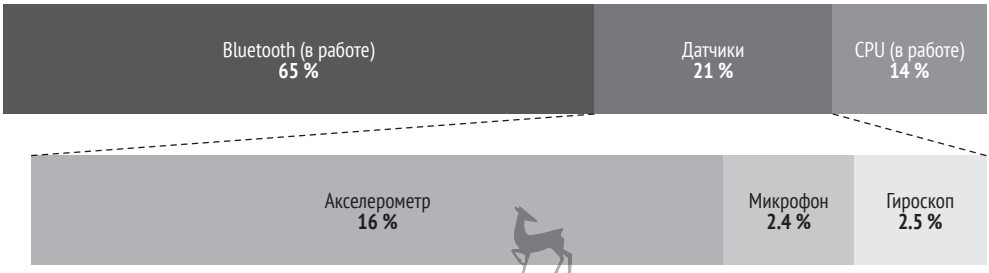


Рис. 1.1 ❖ Распределение энергопотребления платы Arduino Nano 33 BLE Sense

Как мы можем видеть из распределения энергопотребления, вычисления на процессоре более энергоэффективны, чем связь по Bluetooth (14 % против 65 %), поэтому предпочтительнее вычислять больше и передавать меньше, чтобы снизить риск быстрого разряда батареи. Как правило, радио – это компонент, который потребляет больше всего энергии в типичных встраиваемых устройствах.

- *Конфиденциальность*: локальное управление означает сохранение конфиденциальности пользователей и отказ от обмена конфиденциальной информацией.

Теперь, когда мы знаем о преимуществах использования ML на этих малых устройствах, каковы практические возможности и проблемы, связанные с доведением ML до предела его возможностей?

Возможности и проблемы TinyML

TinyML находит свое естественное применение там, где невозможно или сложно получить питание от сети, и приложение должно работать от батареи как можно дольше.

Если вдуматься, то мы уже окружены устройствами с батарейным питанием, которые содержат ML под капотом. Например, носимые устройства, такие как умные часы и браслеты для отслеживания фитнеса, могут распознавать действия человека, чтобы отслеживать заданные цели в области здравоохранения или обнаруживать опасные ситуации вроде падения на землю.

Эти повседневные вещи являются полноценными приложениями TinyML, потому что они питаются от батарей и используют встроенную память устройств для придания смысла данным, получаемым датчиками.

Однако решения с батарейным питанием не ограничиваются только носимыми устройствами. Существуют сценарии, в которых нам могут понадобиться устройства для мониторинга окружающей среды. Например, мы можем рассмотреть возможность развертывания устройств с батарейным питанием под управлением TinyML в лесу для обнаружения пожаров и предотвращения распространения пожаров на большую площадь.

Существует неограниченное количество потенциальных вариантов использования TinyML, и те, которые мы только что кратко представили, – это лишь некоторые из вероятных областей применения.

Однако наряду с открывающимися возможностями предстоит столкнуться и с некоторыми серьезными проблемами. Проблемы возникают с вычислительной точки зрения, поскольку наши устройства ограничены в памяти и вычислительной мощности. Мы работаем на системах с несколькими килобайтами оперативной памяти и в некоторых случаях на процессорах без аппаратного ускорения операций с плавающей запятой.

С другой стороны, среда развертывания может быть недружелюбной. Факторы окружающей среды, такие как пыль и экстремальные погодные условия, могут помешать правильному выполнению наших приложений.

В следующем подразделе мы представим типичные среды развертывания для TinyML.

Среды развертывания для TinyML

Приложение TinyML может работать как в **централизованных**, так и в **распределенных** системах.

В **централизованной** системе приложению не обязательно требуется связь с другими устройствами.

Типичным примером является **распознавание ключевых слов**. В настоящее время мы легко взаимодействуем со смартфонами, камерами, беспилотными летательными аппаратами и кухонной техникой с помощью голосовых команд. Волшебные слова «*OK*», «*Google*», «*Alexa*» и т. д., которые мы используем для пробуждения умных помощников, являются классическим примером ML-модели, постоянно работающей локально в фоновом режиме.

Приложению требуется работать в системе с низким энергопотреблением без отправки данных в облако, чтобы быть эффективным, мгновенно реагировать и минимизировать энергопотребление.

Обычно централизованные приложения TinyML нацелены на запуск более энергоемких функций и извлекают выгоду из того, что они являются автономными по своей природе, поскольку им не нужно отправлять какие-либо данные в облако.

В **распределенной** системе устройство, т. е. **узел** (*node*) или **узел датчика** (*sensor node*), по-прежнему выполняет ML локально, но также взаимодействует с близлежащими устройствами или хостом для достижения общей цели, как показано на рис. 1.2.

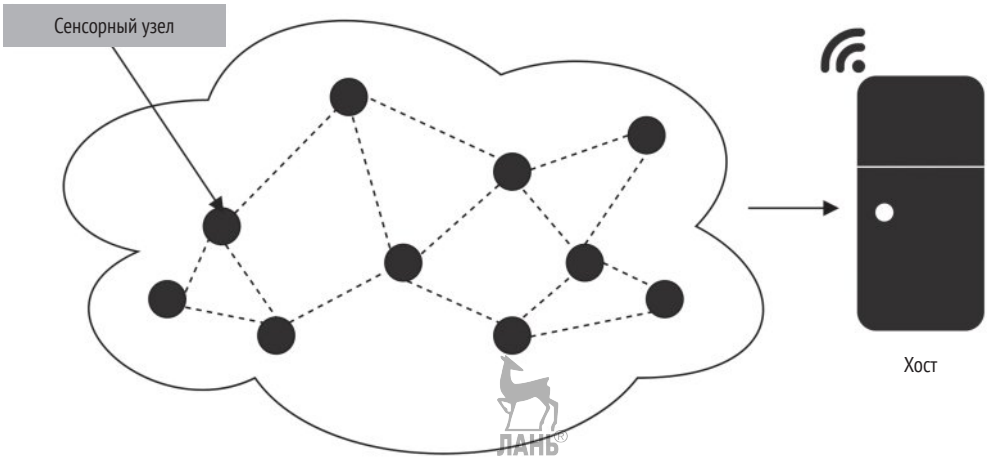


Рис. 1.2 ❖ Беспроводная сенсорная сеть

Поскольку узлы (*node*) являются частью сети и обычно обмениваются данными с помощью беспроводных технологий, мы обычно называем сеть **беспроводной сенсорной сетью** (**wireless sensor network**, **WSN**).

Хотя этот сценарий может показаться невыгодным из-за энергопотребления при передаче данных, устройствам, возможно, потребуется взаимодействовать, чтобы получить значимые и точные данные о рабочей среде. Знание температуры, влажности, влажности почвы или других физических величин для конкретного узла может оказаться неактуальным для некоторых приложений, которым вместо этого потребуются сведения о глобальном распространении этих величин.

WSN может помочь определить, какие участки поля требуют меньше или больше воды, чем другие, и сделать орошение более эффективным. Как мы можем представить, эффективные протоколы связи будут важны для срока автономной службы сети, а TinyML играет определенную роль в достижении этой цели. Поскольку отправка необработанных данных потребляет слишком много энергии, ML может выполнять частичную обработку, чтобы уменьшить объем передаваемых данных и частоту обмена.

TinyML предлагает бесконечные возможности, и **tinyML Foundation** – лучший источник сведений о возможностях, предоставляемых этой быстрорастущей областью ML и встраиваемых систем.

tinyML Foundation

tinyML Foundation (http://www.tinyml.org) – некоммерческая профессиональная организация, поддерживающая и объединяющая мир TinyML.

Для этого tinyML Foundation при поддержке нескольких компаний, включая Arm, Edge Impulse, Google и Qualcomm, выращивает разнообразное сообщество по всему миру: в США, Великобритании, Германии, Италии, Нигерии, Индии, Японии, Австралии, Чили и Сингапуре. В сообщество входят специалисты по аппаратному и программному обеспечению, системные инженеры, ученые, дизайнеры, менеджеры по продуктам и бизнесмены.

Фонд продвигает различные бесплатные инициативы онлайн и офлайн для привлечения экспертов и новичков, поощрения обмена знаниями, налаживания связей и создания более здорового и устойчивого мира с помощью TinyML.

С помощью групп Meetup (https://www.meetup.com) в разных странах вы можете бесплатно присоединиться к сообществу TinyML рядом с вами (https://www.meetup.com/en-AU/pro/TinyML), чтобы всегда быть в курсе новинок TinyML и предстоящих событий.

После представления TinyML давайте более подробно изучим его составляющие. В следующем разделе будет рассказано о том, что делает наши устройства способными принимать интеллектуальные решения: технологии глубокого обучения (*deep learning*, DL).

Краткое описание глубокого обучения (DL)

Машинное обучение ML – это тот компонент, который делает наши малые устройства способными принимать разумные решения. Программные алгоритмы ML в значительной степени полагаются на правильные данные для изучения шаблонов или действий, основанных на опыте. Как мы обычно говорим, данные – это все для ML, это то, что создает или рушит приложение.

В этой книге DL[9] будет рассматриваться как особый класс ML, который может выполнять сложные задачи классификации[10] непосредственно на необ-

[9] Концепция глубокого обучения (Deep Learning, DL) – одно из направлений машинного обучения. Как отдельная дисциплина DL оформилась в 2006 году, прежде всего в связи с задачами распознавания образов. Подробнее о Deep Learning на русском языке см. https://habr.com/ru/company/otus/blog/459785/.

[10] Классификацией (*classification*) в контексте машинного обучения называется самая распространенная разновидность задачи распознавания (*recognition*). Чаще всего эти термины означают одно и тоже, но первый термин более строгий. Поэтому в большинстве случаев в переводе сохранена «классификация» иногда с заменой на «распознавание» как более уместное по смыслу текста. Объекты классификации называются классами (*classes*).

работанных изображениях, тексте или звуке. Алгоритмы DL обладают самой высокой точностью, а также в некоторых задачах классификации могут быть лучше, чем люди. Именно эта технология сделала возможным беспилотное вождение, существование виртуальных помощников с голосовым управлением и подняла на невиданную высоту системы распознавания лиц.

Полное обсуждение архитектур и алгоритмов DL выходит за рамки этой книги. Тем не менее в этом разделе будут обобщены некоторые из его основных моментов, которые имеют отношение к пониманию материала следующих глав.

Глубокие нейронные сети

Глубокая нейронная сеть (**deep neural network**, **DNN**) состоит из нескольких вложенных слоев, направленных на изучение паттернов[11]. Каждый слой содержит несколько нейронов, фундаментальных вычислительных элементов для искусственных нейронных сетей (**artificial neural networks, ANN**), сделанных по образцу клеток человеческого мозга.

Нейрон выдает один выходной сигнал посредством линейного преобразования, определяемого как взвешенная сумма входных данных плюс постоянное значение, называемое **смещением** (**bias**), как показано на рис. 1.3.

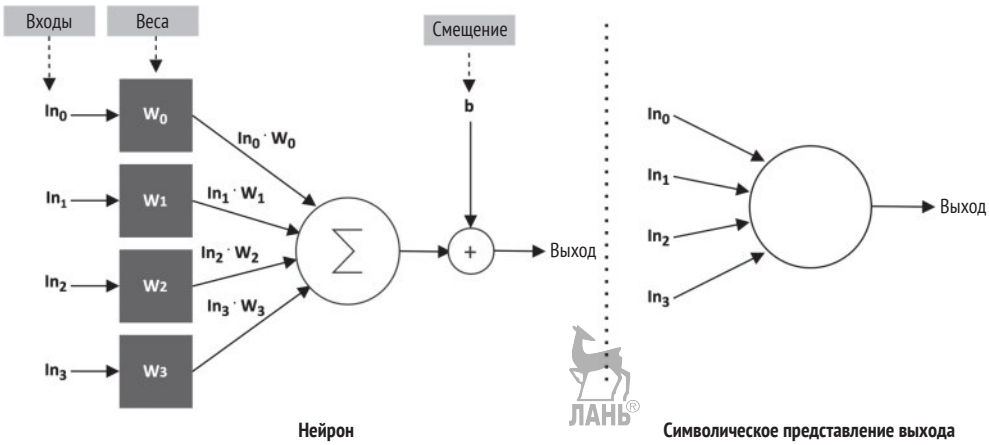


Рис. 1.3 ❖ Представление нейрона

Коэффициенты (W) взвешенной суммы называются **весами** (**weights**).

Веса и смещение получаются после итеративного процесса обучения, чтобы сделать нейрон способным к изучению сложных паттернов.

---

[11] Паттерн (*pattern*) – «шаблон», в машинном обучении термин обозначает группу многомерных объектов в пространстве признаков, указанных «учителем» как имеющие общие черты. Паттерн следует отличать от кластера (*cluster*) – группы объектов, общие черты которых выявляются в пространстве признаков автоматически, без участия «учителя».

Однако нейроны могут решать только простые линейные задачи с помощью линейных преобразований. Поэтому, чтобы помочь сети изучать сложные паттерны, к выходу нейрона обычно подключаются нелинейные функции, называемые **активациями** (**activations**). Активация – это нелинейная функция, выполняющаяся на выходе нейрона:

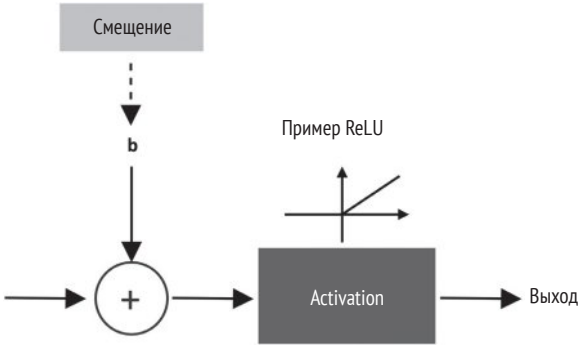


Рис. 1.4 ❖ Функция активации

Широко распространенной функцией активации является спрямленный линейный модуль (**rectified linear unit, ReLU**), описываемый следующим фрагментом кода:

```
float relu(float input) {
  return max(input, 0);
}
```

Его вычислительная простота делает его предпочтительным по сравнению с другими нелинейными функциями, такими как гиперболическая касательная или логистическая сигмоида[12], которые требуют больше вычислительных ресурсов.

В следующем подразделе мы увидим, как нейроны связываются для решения сложных задач визуального распознавания.

Сверточные нейронные сети

Сверточные нейронные сети (**CNNS**) – это специализированные глубокие нейронные сети, преимущественно применяемые для задач визуального распознавания.

Мы можем рассматривать CNNs как эволюцию упорядоченной версии классических полносвязных нейронных сетей с уплотненными (т. е. полностью связанными) слоями.

[12] Сигмоида – название класса кривых, общем признаком которых является S-образная симметричная (по обеим осям) форма с асимптотическим приближением к нулю или единице на краях. Подробнее см., например, в статье «Википедии» «Сигмоида».

Характеристикой **полносвязных сетей** (*fully connected*) является подключение каждого нейрона ко всем выходным нейронам предыдущего уровня, как мы можем видеть на следующем рисунке:

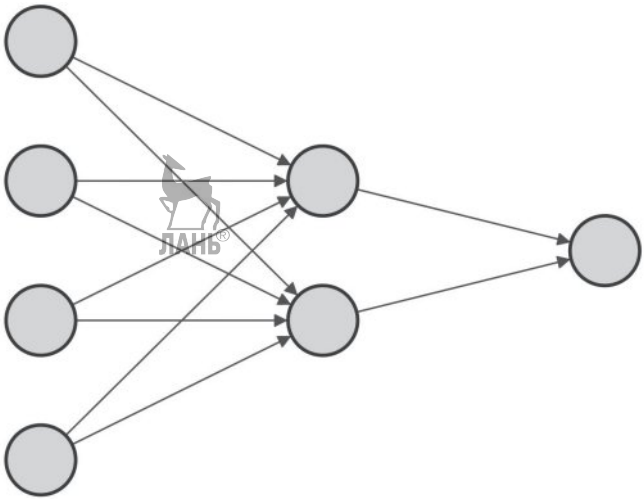


Рис. 1.5 ❖ Полносвязная сеть

К сожалению, этот подход плохо работает для обучения модели классификации изображений.

Например, если бы мы рассматривали изображение в формате RGB размером 320×240 (ширина × высота), нам понадобилось бы 230 400 (320×240×3) весов только для одного нейрона. Поскольку нашим слоям, несомненно, потребуется несколько нейронов для распознавания сложных проблем, модель, скорее всего, будет перегружена, учитывая неуправляемое количество обучаемых параметров.

В прошлом специалисты по обработке данных использовали особенности инженерных технологий для извлечения из изображений уменьшенного набора полезных признаков. Однако этот подход страдал от того, что было сложно выполнять отбор признаков, что отнимало много времени и зависело от конкретной предметной области.

С появлением CNN[13] задачи визуального распознавания улучшились благодаря слоям свертки (*convolution layers*), что делает извлечение признаков частью задачи обучения.

CNN возник в результате изучения биологических процессов в зрительной коре животных. В предположении, что мы имеем дело с изображениями, слой свертки заимствует широко распространенный оператор свертки из обработки изображений для создания набора признаков, поддающегося обучению.

---

[13] CNN (*convolutional neural network*, иначе ConvNet) – сверточная нейронная сеть, специальная архитектура искусственных нейронных сетей, направленная на эффективное распознавание образов. Входит в состав технологий глубокого обучения (DL). Подробнее на русском см. https://habr.com/ru/company/skillfactory/blog/565232/.

Оператор свертки (*convolution*) выполняется аналогично другим процедурам обработки изображений: перемещение окна оператора (фильтра, ядра) по всему входному изображению и применение скалярного произведения между его весами и соседними пикселями, как показано на следующем рисунке:

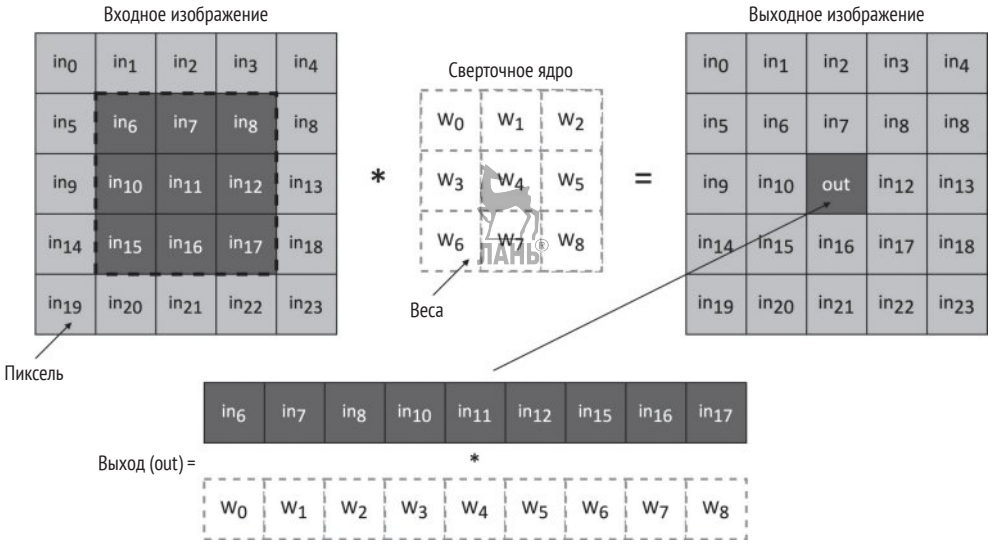


Рис. 1.6 ❖ Оператор свертки

Такой подход дает два существенных преимущества:

- он автоматически извлекает соответствующие функции без вмешательства человека;
- значительно уменьшает количество входных сигналов на один нейрон.

Например, для применения фильтра 3×3 к показанному RGB-изображению потребуется всего 27 весов (3×3×3).

Как и для полносвязных слоев, слоям свертки требуется несколько сверточных ядер, чтобы изучить как можно больше параметров. Следовательно, выходные данные слоя свертки создают набор изображений (**карты параметров**, *feature maps*), обычно хранящийся в памяти в виде многомерного объекта, называемого **тензором**.

При проектировании CNN для задач визуального распознавания мы обычно размещаем полносвязные слои на конце сети для выполнения этапа классификации. Поскольку выходные данные слоев свертки представляют собой набор изображений, как правило, мы используем стратегии подвыборки, чтобы уменьшить объем информации, распространяемой по сети, а затем снизить риск переобучения при подаче полносвязных слоев.

Как правило, существует два способа выполнения подвыборки.

- Пропуск оператора свертки для некоторых входных пикселей. В результате выходные данные слоя свертки будут иметь меньшее количество пространственных измерений, чем входные.

- Внедрение функций подвыборки, таких как объединение слоев (*pooling layers*).

На следующем рисунке показана общая архитектура CNN, в которой на уровне объединенного слоя уменьшается пространственная размерность, а на уровне полносвязного выполняется этап классификации:

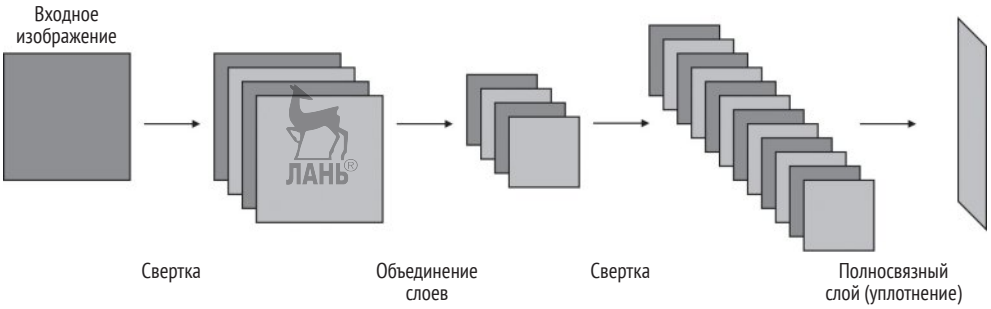


Рис. 1.7 ❖ Общая структура CNN с объединенным (*pooling*) слоем для уменьшения пространственной размерности

Одним из наиболее важных аспектов, которые следует учитывать при развертывании сетей DL для TinyML, является размер модели, обычно определяемый как объем памяти, необходимый для хранения значений весов.

Поскольку наши малые платформы имеют ограниченную физическую память, мы требуем, чтобы модель была компактной, чтобы соответствовать целевому устройству.

Однако ограниченный объем памяти не единственная проблема, с которой мы можем столкнуться при развертывании модели на микроконтроллерах. Например, хотя обученная модель обычно использует арифметические операции с точностью до плавающей запятой, процессоры на микроконтроллерах могут не иметь для этого аппаратной поддержки.

Незаменимым методом для преодоления указанных ограничений является **квантизация** (*quantizing*)[14].

Квантизация

Квантизация – это процесс выполнения вычислений для нейронной сети с более низкой битовой точностью. Широко распространенный метод для микроконтроллеров применяет после обучения квантизацию и преобразует 32-разрядные веса с плавающей запятой в 8-разрядные целочисленные значения. Этот метод позволяет уменьшить размер модели в 4 раза и значительно снизить время ожидания при очень незначительном снижении точности или вообще без него.

[14] Специфический для ML процесс квантизации (*quantizing*, см. далее) не следует путать с квантованием (*quantization*) – процессом разбиения диапазона значений на конечное число уровней.

DL имеет важное значение для создания приложений, которые принимают интеллектуальные решения. Однако ключевым требованием для приложений с батарейным питанием является использование устройства с низким энергопотреблением. До сих пор мы упоминали мощность и энергию в общих чертах, но в следующем разделе посмотрим, что они означают практически.

Разница между мощностью и энергией

Целевое потребление мощности в TinyML, к которой мы стремимся, находится в диапазоне милливатт (мВт) или ниже, что означает в тысячи раз большую эффективность, чем у традиционной настольной машины. Хотя есть случаи, когда мы могли бы рассмотреть возможность использования решений для дополнительного сбора энергии, таких как солнечные панели, это не всегда возможно из-за стоимости и физических размеров.

Однако что мы подразумеваем под мощностью и энергией? Давайте познакомимся с этими терминами, дав общий обзор фундаментальных физических величин, управляющих электронными схемами.

Различие между напряжением и током

Ток – это то, что заставляет работать электронную схему. Ток представляет собой *количество электрических зарядов Q через сечение S проводника за заданное время t*, как представлено на следующем рисунке:

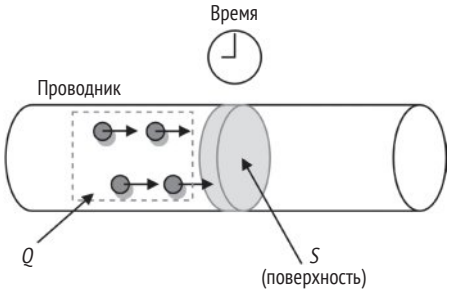


Рис. 1.8 ❖ Ток представляет собой количество электрических зарядов *Q* через сечение проводника *S* за промежуток времени *t*

Ток определяется следующим образом:

$$I = Q/t.$$

Здесь мы имеем:

- *I*: ток, измеряемый в амперах (А),
- *Q*: количество электрических зарядов в кулонах (К) через сечение *S* за заданное время,

- t: время, измеряемое в секундах (с).

Ток протекает в цепи при следующих условиях.

- У нас есть *проводящий материал* (например, медная проволока), позволяющий пропускать электрический заряд.
- У нас *замкнутая цепь*, т. е. цепь без разрывов, обеспечивающая непрерывный путь для протекания тока.
- У нас есть *источник разности потенциалов*, называемой **напряжением**, определяемым следующим образом:

$$V = V^+ - V^-.$$

Напряжение измеряется в **вольтах** (В или *V*) и создает электрическое поле, позволяющее электрическому заряду протекать по цепи. Любой источник питания, как порт USB, так и аккумулятор, являются источником разности потенциалов.

Символическое представление источника питания приведено на следующем рисунке:

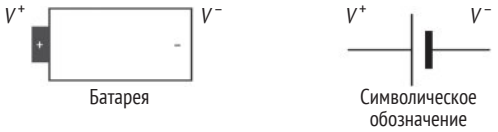


Рис. 1.9 ❖ Символическое обозначение батареи

Чтобы избежать постоянного обращения к V^+ и V^-, мы условно определяем отрицательный вывод батареи как опорный, присваивая ему значение 0 В (GND[15]). Напряжение и ток связаны через **закон Ома**: он гласит, что *ток через проводник пропорционален напряжению и обратно пропорционален сопротивлению проводника*:

$$I = V/R.$$

Все проводники обладают сопротивлением, однако есть специальные компоненты под названием резисторы. **Резистор** – это электрический компонент, используемый для установки значения тока. Он имеет сопротивление, измеряемое в омах (Ом), и обозначается буквой *R*.

Символическое обозначение резистора показано на рис. 1.10.

Резисторы являются важными компонентами любой электронной схемы, и для разновидностей, используемых в этой книге, их значение указано с помощью цветных полос на корпусе. Стандартные резисторы имеют четыре,

[15] Обозначение GND (от *ground* – земля) произошло от названия общего провода схемы, который на заре электронной эры часто «заземлялся» – соединялся с настоящей (электротехнической) землей. В настоящее время для GND более правильным будет употребление названия «общий провод» (а не «заземление»), так как с электротехнической землей его соединять, наоборот, не рекомендуется, кроме особо оговоренных случаев.

пять или шесть полос. Цвет на полосах обозначает значение сопротивления, как показано на рис. 1.11.

Рис. 1.10 ❖ Символическое обозначение резистора

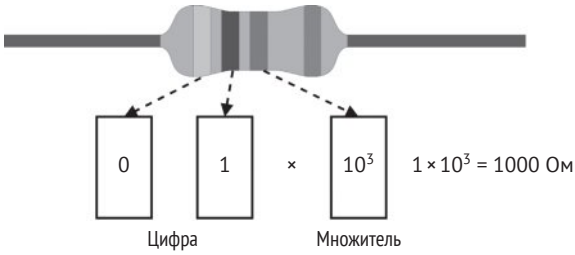


Рис. 1.11 ❖ Пример резистора с четырьмя полосами

Чтобы легко декодировать цветовые полосы, мы рекомендуем использовать онлайн-инструмент магазина «Чип и дип» (https://www.chipdip.ru/calc/resistor?ysclid=l9nxpl83xb883320558).

Теперь, когда мы знаем основные физические величины, управляющие электронными схемами, рассмотрим разницу между мощностью и энергией.

Мощность и энергия

Иногда мы меняем местами слова «мощность» и «энергия», потому что думаем, что это одно и то же, но на самом деле они относятся к разным физическим величинам. Энергия – это способность выполнять работу (например, использовать силу для перемещения объекта), в то время как мощность – это скорость выделения или потребления энергии, т. е. энергия в единицу времени.

С практической точки зрения мощность говорит, например, о том, как быстро разрядится аккумулятор, – более высокая мощность подразумевает более быструю разрядку аккумулятора.

Мощность P и энергия E связаны с напряжением и током по следующим формулам:

$$P = V \cdot I,$$
$$E = P \cdot t.$$

В следующей таблице представлены входящие в эти формулы физические величины мощности и энергии:

Величина	Единица	Значение
P	Ватт (Вт)	Мощность
E	Джоуль (Дж)	Энергия
V	Вольт (В)	Напряжение питания
I	Ампер (А)	Потребление тока
t	Секунда (с)	Время операции

Таблица 1.1. Таблица с указанием физических величин в формулах для мощности и энергии

Для микроконтроллеров напряжение питания составляет порядка нескольких вольт (например, 3.3 В), в то время как потребляемый ток находится в диапазоне микроампер (мкА) или миллиампер (мА). По этой причине мы обычно используем микроватт (мкВт) или милливатт (мВт) для мощности и микроджоуль (мкДж) или миллиджоуль (мДж) для энергии.

Теперь рассмотрим следующую задачу, чтобы ознакомиться с концепциями мощности и энергии поближе. Предположим, у вас есть задача обработки, и у вас есть возможность выполнить ее на двух разных процессорах. Эти процессоры имеют следующее энергопотребление и производительность:

Таблица 1.2. Два процессора с разной производительностью и энергопотреблением

Процессор	Энергопотребление, мВт	Производительность, отн. ед.
PU1	12	8x
PU2	3	1x

Какой процессор вы бы использовали для выполнения этой задачи?

Хотя PU1 имеет в 4 раза более высокое энергопотребление, чем PU2, это не означает, что PU1 менее энергоэффективен. Напротив, PU1 более производительный в вычислительном отношении, чем PU2 (в целых 8 раз), что делает его лучшим выбором с энергетической точки зрения, как показано в следующих формулах:

$$E_{PU1} = 12 \cdot T_1;$$
$$E_{PU2} = 3 \cdot T_2 = 3 \cdot 8 \cdot T_1 = 24 \cdot T_1.$$

Мы можем сказать, что PU1 – наш лучший выбор, поскольку он потребит вдвое меньше энергии от аккумулятора при выполнении тех же самых операций.

Обычно мы используем величину количества арифметических операций, выполняемых на один ватт мощности (**OPS per Watt**), чтобы привязать потребляемую мощность к вычислительным ресурсам процессоров.

Программирование микроконтроллеров

Микроконтроллер (**MCU, МК**) является полноценным компьютером, потому что у него есть процессор (который в настоящее время также может быть многоядерным), система памяти (например, RAM или ROM) и неко-

торые периферийные устройства. В отличие от стандартного компьютера микроконтроллер полностью помещается на одном интегрированном чипе, обладает невероятно низкой мощностью и низкой стоимостью.

Часто путают микроконтроллеры с **микропроцессорами (МП)**, но эти названия относятся к разным устройствам. В отличие от микроконтроллера микропроцессор содержит на одном чипе только процессор (**central processing unit**, **CPU**), требующий внешних подключений к системе памяти и другим компонентам для формирования полностью работающего компьютера.

На следующем рисунке приведены основные различия между микропроцессором и микроконтроллером:

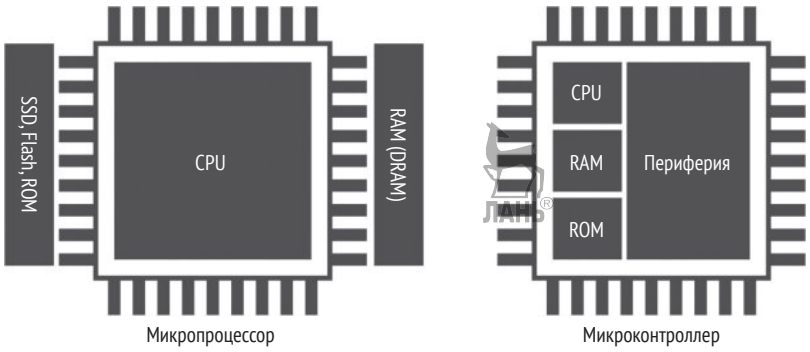


Рис. 1.12 ❖ Микропроцессор в сравнении с микроконтроллером

Как и для всех процессорных блоков, целевое применение влияет на выбор архитектуры.

Микропроцессор выполняет сценарии, в которых задачи заключаются, например, в следующем:

- динамические задачи (меняются в зависимости от действий пользователя или во времени),
- общего назначения,
- интенсивные вычисления.

Микроконтроллер выполняет совершенно другие сценарии, и в следующем списке мы выделим некоторые из важнейших из них.

- *Задачи МК являются одноцелевыми и повторяющимися*.
  В отличие от микропроцессорных приложений задачи для МК, как правило, являются одноцелевыми и повторяющимися, поэтому микроконтроллер не требует перезагрузки программ. Как правило, приложения МК требуют меньших вычислительных затрат, чем микропроцессорные, и не требуют частого взаимодействия с пользователем. Однако они могут взаимодействовать с окружающей средой или другими устройствами.
  В качестве примера вы могли бы рассмотреть термостат. Устройство требует только регулярного контроля температуры и связи с системой отопления.
- *Могут быть ограничения по срокам выполнения*.
  Определенные задачи должны быть выполнены в течение определенного периода времени. Это требование характерно для приложений

реального времени (**real-time applications**, **RTA**), где нарушение временных ограничений может повлиять на качество обслуживания (мягкий режим реального времени) или быть опасным (жесткий режим реального времени).

Автомобильная антиблокировочная система (**ABS**) является примером жесткого RTA, поскольку электронная система должна реагировать в течение определенного периода времени, чтобы предотвратить блокировку колес при нажатии на педаль тормоза.

Для построения эффективного RTA нам требуется устройство с предсказуемой задержкой, поэтому все аппаратные компоненты (процессор, память, обработчик прерываний и т. д.) должны реагировать за точное количество тактов. Изготовители оборудования обычно сообщают в техническом описании о задержке, выраженной в тактовых циклах.

Ограничение по времени требует некоторых архитектурных изменений и ограничений в сравнении с микропроцессором общего назначения. Примером может служить модуль управления памятью (**memory management unit**, **MMU**), который мы в основном используем для преобразования адресов виртуальной памяти, и обычно в составе процессорного модуля микроконтроллеров он отсутствует.

- *Ограничения вследствие низкого энергопотребления*.

  Приложения могут работать в устройстве с батарейным питанием, поэтому микроконтроллер должен быть маломощным, чтобы продлить срок службы источника.

  В соответствии с ограничениями по времени энергопотребление также создает некоторые архитектурные отличия от микропроцессора. Не вдаваясь в подробности аппаратного обеспечения, замечу, что, как правило, дополнительные компоненты на отдельных чипах снижают энергоэффективность. Это основная причина, по которой микроконтроллеры интегрируют как оперативную память, так и своего рода жесткий диск (постоянное запоминающее устройство, ROM) в единую микросхему.

  Как правило, микроконтроллеры также имеют более низкую тактовую частоту, чем микропроцессоры, чтобы потреблять меньше энергии[16].

- *Ограничения по физическому размеру*.

  Устройство может представлять собой продукт небольшого размера. Поскольку микроконтроллер представляет собой разновидность одно-

[16] Это не всегда связано именно с энергопотреблением: рассуждение, подобное приведенному автором выше о количестве операций на ватт мощности, поможет вам понять, что простое снижение тактовой частоты не принесет выгоды в потреблении энергии: те же операции будут выполняться дольше, и общее количество затраченной энергии останется примерно тем же самым. Тактовая частота МК обычно ниже МП по той причине, что МК, как правило, нацелен на выполнение более простых задач, не требующих высоких скоростей вычислений, а более медленные микросхемы не требуют суперсовременного производства с предельными технологическими нормами, и потому существенно дешевле. Кроме того, схемотехнические и конструктивные решения для них при встраивании в аппаратуру оказываются намного проще.

кристальных компьютеров, он идеально подходит для таких устройств. Площадь, занимаемая микроконтроллером, может варьироваться, но обычно находится в пределах нескольких квадратных миллиметров. В 2018 году команда инженеров из Мичиганского университета создала «самый маленький в мире компьютер» размером 0,3 мм с микроконтроллером, работающим на процессоре Arm Cortex-M0+ и системой датчиков без батареи для измерения температуры в живых клетках.

- *Ограничения по стоимости*.
  Все приложения чувствительны к стоимости их производства, а благодаря разработке микросхемы меньшего размера, интегрирующей процессор, память и периферийные устройства, мы делаем микроконтроллеры экономически более выгодными, чем микропроцессоры.

В следующей таблице кратко изложено рассмотренное выше для удобства использования в будущем:

Таблица 1.3. Таблица сравнения микропроцессора с микроконтроллером

Свойство	**Микропроцессор**	**Микроконтроллер**
Приложения	Общего назначения	Конкретного назначения
Арифметический сопроцессор	Может выполнять вычисления с плавающей запятой и с двойной точностью	Преимущественно целочисленные вычисления
RAM	Несколько гигабайт	Несколько сотен килобайт
ROM (или жесткий диск)	Гигабайты и терабайты	Килобайты или мегабайты
Тактовая частота	Гигагерцы	Мегагерцы
Потребление	Ватты	Милливатты и ниже
Операционная система	Требуется	Не обязательна
Стоимость	Десятки-сотни долларов	От нескольких центов до нескольких долларов

Со следующего раздела мы начнем углубляться в архитектурные аспекты микроконтроллеров, анализируя архитектуру памяти и внутренних периферийных устройств.

Архитектура памяти

Микроконтроллеры – это встроенные системы на основе центрального процессора, что означает, что центральный процессор (CPU) отвечает за взаимодействие со всеми его подкомпонентами. Всем процессорам требуется, по крайней мере, память для чтения инструкций и сохранения/считывания значений переменных во время выполнения программы.

В микроконтроллерах мы физически выделяем две отдельные системы памяти для инструкций и данных.

- **Память программ** (**ROM**). Это энергонезависимая память, доступная только для чтения, зарезервированная для хранения выполняемой программы. Хотя ее основная цель – хранить программу, она также

может хранить некоторые постоянные данные. Таким образом, память программ похожа на жесткие диски наших обычных компьютеров.

- **Память данных** (**RAM**). Это энергозависимая память, зарезервированная для хранения и чтения временных данных (переменных). Поскольку это оперативная память, мы теряем ее содержимое при выключении системы.

Поскольку память программ и память данных различаются функционально, для них обычно используются разные полупроводниковые технологии. Это **флеш-технологии** для памяти программ и **статическая оперативная память** (**SRAM**) для памяти данных.

Флеш-память является энергонезависимой и обеспечивает низкое энергопотребление, но, как правило, работает медленнее, чем SRAM. Однако, учитывая преимущество в стоимости[17] по сравнению с SRAM, мы обычно имеем память программ большего объема, чем память данных.

Теперь, когда мы знаем разницу между памятью программ и памятью данных, *где мы можем хранить веса нашей модели глубокой нейронной сети*?

Ответ на этот вопрос зависит от того, имеет ли модель веса с постоянными значениями. Если веса постоянны и не изменяются во время просчета модели, более эффективно хранить их в памяти программ по следующим причинам:

- память программ имеет больший объем, чем SRAM;
- это уменьшает нагрузку на память SRAM, поскольку другие функции требуют хранения переменных или фрагментов памяти во время выполнения.

Мы хотим напомнить вам, что ресурсы памяти микроконтроллеров ограничены, поэтому подобное решение может повлиять на эффективность работы памяти.

Периферийные устройства

Микроконтроллеры предлагают дополнительные встроенные функции, расширяющие их возможности и делающие эти малые компьютеры непохожими друг на друга. Эти функции реализуются через периферийные устройства и имеют важное значение, поскольку именно они позволяют взаимодействовать с датчиками и другими внешними компонентами.

Каждое периферийное устройство имеет специальную функциональность, и оно закреплено за металлической ножкой (выводом) интегральной схемы.

Мы можем обратиться к описанию назначения периферийных выводов в спецификации микроконтроллера, чтобы узнать функциональные возможности каждого вывода. Поставщики оборудования обычно нумеруют выводы против часовой стрелки, начиная с крайнего левого угла корпуса, отмеченного точкой для удобства ориентирования, как показано на следующем рисунке:

[17] А также в занимаемой площади на кристалле.

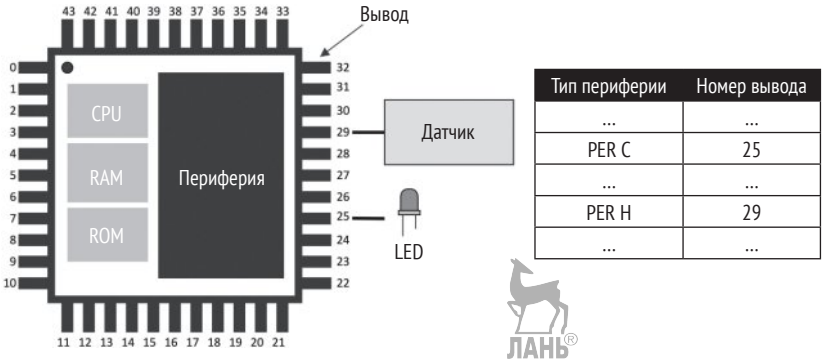


Рис. 1.13 ❖ Назначение выводов.
Выводы нумеруются против часовой стрелки,
начиная с верхнего левого угла, отмеченного точкой[18]

Поскольку периферийные устройства могут быть различных типов, для простоты мы можем сгруппировать их по четырем основным категориям.

Вход/выход общего назначения (GPIO или IO)

GPIO (*general-purpose input/output*) не имеют предопределенного и фиксированного назначения. Их основная функция заключается в установке или считывании двоичных сигналов, которые по своей природе могут находиться только в двух четко определенных состояниях: **высоком** (**HIGH** или 1) или **низком** (**LOW** или 0). На следующем рисунке показан пример двоичного сигнала:

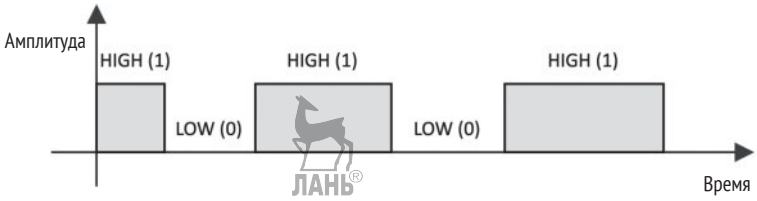


Рис. 1.14 ❖ Двоичный сигнал может находиться только в двух состояниях:
высоком (HIGH или 1) или низком (LOW или 0)

Типичные способы использования GPIO, например, такие:
- включение и выключение светодиода;
- определение того, нажата ли кнопка;
- реализация сложных цифровых интерфейсов/протоколов, таких как VGA.

Периферийные устройства GPIO универсальны и обычно доступны во всех микроконтроллерах.

[18] Нумерация выводов микросхем начинается с единицы, а не с нуля, как ошибочно показано на рис. 1.13.

Аналого-цифровые преобразователи

Приложения TinyML, скорее всего, будут иметь дело с изменяющимися во времени физическими величинами, такими как освещенность, звук или температура.

Какими бы ни были эти величины, датчик преобразует их в непрерывный электрический сигнал, интерпретируемый микроконтроллерами. Этот электрический сигнал, который может быть как напряжением, так и током, обычно называют аналоговым сигналом.

Микроконтроллер, в свою очередь, должен преобразовать аналоговый сигнал в цифровой формат, чтобы центральный процессор мог обрабатывать данные.

Аналого-цифровые преобразователи действуют как переводчики между аналоговым и цифровым мирами.

Аналого-цифровой преобразователь (**analog-to-digital converter, ADC, АЦП**) производит выборку аналогового сигнала с фиксированным интервалом времени и преобразует электрический сигнал в цифровой формат.

Цифроаналоговый преобразователь (**digital-to-analog converter DAC, ЦАП**) выполняет противоположную функцию: преобразует внутренний цифровой формат в аналоговый сигнал.

Последовательная связь

Коммуникационные периферийные устройства интегрируют стандартные протоколы связи для управления внешними компонентами. Типичными периферийными устройствами последовательной связи, доступными в микроконтроллерах, являются **I2C**, **SPI**, **UART** и **USB**.

Таймеры

В отличие от всех периферийных устройств, которые мы только что описали, таймеры не взаимодействуют с внешними компонентами, поскольку они используются для запуска или синхронизации событий.

В этом разделе мы завершили обзор ингредиентов TinyML. Теперь, когда мы знакомы с терминологией и общими понятиями, мы можем начать представлять платформы разработки, используемые в этой книге.

Представление Arduino Nano 33 BLE Sense и Raspberry Pi Pico

Плата микроконтроллера – это печатная плата (PCB), объединяющая микроконтроллер с необходимыми электронными компонентами, чтобы сделать его готовым к использованию. В некоторых случаях плата микроконтроллера может содержать дополнительные устройства, предназначенные для конкретных приложений.

В этой книге используются платы микроконтроллеров Arduino Nano 33 BLE Sense (сокращенно Arduino Nano[19]) и Raspberry PI Pico.

Arduino Nano, разработка компании Arduino (https://www.arduino.cc), представляет собой плату, сочетающую микроконтроллер (nRF52840 на базе процессора ARM Cortex-M4) с несколькими датчиками и беспроводную Bluetooth-связь для упрощения разработки TinyML. При разработке на Arduino Nano нам потребуется всего несколько дополнительных внешних компонентов, поскольку большинство из них уже доступно «на борту».

Raspberry Pi Pico, разработка Raspberry Pi Foundation (https://www.raspberrypi.org), не содержит датчиков и встроенного модуля Bluetooth. Тем не менее он оснащен микроконтроллером (RP2040), работающим на двухъядерном процессоре Arm Cortex-M0+, пригодным для уникальных и мощных приложений TinyML. Таким образом, эта плата идеально подойдет для изучения взаимодействия с внешними датчиками и построения электронных схем.

На следующем рисунке показано параллельное сравнение этих двух платформ, чтобы нагляднее показать их различия друг от друга:

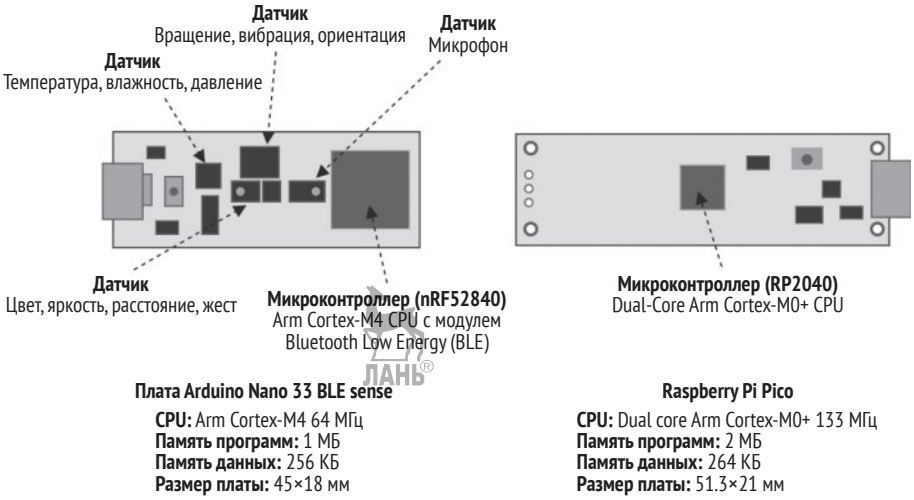


Рис. 1.15 ❖ Arduino Nano 33 BLE Sense и Raspberry Pi Pico

Как мы можем видеть, обе платы имеют небольшие размеры, USB-порт для питания и программирования, а также микроконтроллер на базе ARM. В то же время они обладают некоторыми уникальными функциями, которые делают платы идеальными для различных сценариев разработки TinyML.

[19] Под Arduino Nano без дополнительных пояснений в дальнейшем в этой книге подразумевается именно плата Arduino Nano 33 BLE Sense, которую не следует путать с собственно Arduino Nano – популярной в среде любителей платой Arduino-контроллера на основе ATmega328.

Настройка Arduino Web Editor, TensorFlow и Edge Impulse

Для TinyML нам требуются различные программные средства, охватывающие как разработку ML, так и программирование встроенных устройств. Благодаря компаниям Arduino, Edge Impulse и Google большинство инструментов, рассмотренных в этой книге, основаны на браузере и требуют всего нескольких шагов настройки.

В этом разделе мы представим эти инструменты и подготовим среду разработки Arduino, необходимую для написания и загрузки программ в Arduino Nano и Raspberry Pi Pico.

Подготовка веб-редактора Arduino Web Editor

Интегрированная среда разработки Arduino (**Arduino IDE**) – это программное приложение, разработанное компанией Arduino (https://www.arduino.cc/en/software) для написания и загрузки программ на платы, совместимые с Arduino. Программы написаны на C++ и обычно называются программистами скетчами Arduino.

Arduino IDE делает разработку программного обеспечения доступной и простой для разработчиков, не имеющих опыта в области встраиваемого программирования. Фактически этот инструмент скрывает все сложности, с которыми мы можем столкнуться при работе со встроенными платформами, такими как кросс-компиляция и программирование устройств.

Arduino также предлагает среду IDE на основе браузера (https://create.arduino.cc/editor). Она называется Arduino Web Editor и делает программирование еще более простым, поскольку программы могут быть написаны, скомпилированы и загружены на микроконтроллеры непосредственно из веб-браузера. Все проекты Arduino, представленные в этой книге, будут основаны на этой облачной среде. Однако, поскольку бесплатный тарифный план Arduino Web Editor ограничен 200 с времени компиляции в день, вы можете перейти на платный тарифный план или использовать бесплатную локальную среду разработки Arduino IDE, чтобы получить неограниченное время компиляции.

В следующих главах этой книги мы будем полагать Arduino IDE и Arduino Web Editor взаимозаменяемыми[20].

Подготовка TensorFlow

TensorFlow (https://www.tensorflow.org) – бесплатная программная платформа с открытым исходным кодом, разработанная Google для ML. Мы будем ис-

[20] Отметим, что наименования и размещение пунктов меню в локальной среде Arduino IDE и в онлайновом Arduino Web Editor могут существенно различаться.

пользовать это программное обеспечение для разработки и обучения наших моделей ML с использованием Python в Google Colaboratory.

Colaboratory (https://colab.research.google.com/notebooks), сокращенно **Colab**, – это бесплатная среда разработки на Python, которая запускается в браузере с использованием Google Cloud. Она похожа на записную книжку Jupyter, но имеет некоторые существенные отличия, например:

- среда не нуждается в настройке;
- среда основана на облаке и хостится в Google;
- имеет множество предустановленных библиотек Python (включая TensorFlow);
- интегрирована с Google Drive;
- предлагает бесплатный доступ к общим ресурсам GPU и TPU[21];
- результатами легко поделиться (в том числе на GitHub).

В этом случае библиотека TensorFlow не требует настройки, поскольку Colab поставляется вместе с ней.

В Colab мы рекомендуем включить ускорение с помощью графического процессора на вкладке **Runtime**, чтобы ускорить вычисления в TensorFlow. Для этого перейдите в раздел **Runtime | Change runtime type** и выберите **GPU** в раскрывающемся списке **Hardware accelerator**, как показано на следующем скриншоте:

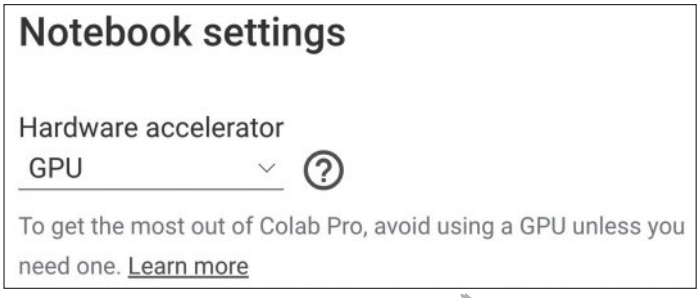


Рис. 1.16 ❖ Включение графического ускорителя

Поскольку ускорение графического процессора является общим ресурсом для других пользователей, доступ к нему из бесплатной версии Colab ограничен.

Вы можете подписаться на Colab Pro (https://colab.research.google.com) для получения приоритетного доступа к самым быстрым графическим процессорам.

[21] GPU (*graphics processing unit*, графический процессор, не путать с обычным процессором CPU) – отдельный процессор, изначально предназначенный для обработки изображений, но хорошо пригодный для выполнения других задач, включающих однотипные параллельные вычисления. TPU (Tensor Processing Unit, тензорный процессор) – разработка Google, специальная ориентированная на задачи машинного обучения. Об особенностях GPU и TPU в задачах ML см. по-русски https://habr.com/ru/post/422317.

TensorFlow не единственный инструмент от Google, который мы будем использовать. Как только мы создадим модель ML, нам нужно будет запустить ее на микроконтроллере. Для этого Google разработала специальную библиотеку TensorFlow Lite for microcontrollers.

TensorFlow Lite for microcontrollers (https://www.tensorflow.org/lite/microcontrollers), сокращенно **TFLu**, – ключевая библиотека программного обеспечения для размещения приложений ML на маломощных микроконтроллерах. Проект является частью TensorFlow и позволяет запускать модели DL на устройствах с несколькими килобайтами памяти. Написанная на C/C++, библиотека не требует операционной системы и динамического выделения памяти.

TFLu не нуждается в настройке, поскольку включена в Arduino Web Editor.

Подготовка Edge Impulse

Edge Impulse (https://www.edgeimpulse.com) – это программная платформа для сквозной разработки ML. Она бесплатна для разработчиков, и через несколько минут вы уже можете запустить модель ML на вашем микроконтроллере. Фактически платформа объединяет инструменты для следующих действий:

- сбора данных с датчиков,
- применения процедур цифровой обработки сигналов к входным данным,
- построения и обучения моделей ML,
- тестирования моделей ML,
- развертывания моделей ML на микроконтроллерах,
- поиска наилучшего блока обработки сигналов и модели ML для вашего варианта использования.

К сведению

Все перечисленные инструменты также доступны через открытые API.

Разработчикам просто нужно зарегистрироваться на веб-сайте, чтобы получить доступ ко всем этим функциям непосредственно через пользовательский интерфейс.

Как это делается...

Вы должны выполнить следующие шаги по настройке Arduino Web Editor.

1. Зарегистрируйтесь в Arduino по адресу https://auth.arduino.cc/register.
2. Войдите в Arduino Web Editor (https://create.arduino.cc/editor).
3. Установите агент Arduino, следуя пошаговой установке по адресу https://create.arduino.cc/getting-started/plugin/welcome.
4. Установите Raspberry Pi Pico SDK.

- Windows.
  i. Загрузите папку *pico-setup-windows* с https://github.com/ndabas/pico-setup-windows/releases.
  ii. Запустите *pico-setup.cmd*.
- Linux.
  i. Откройте Terminal.
  ii. Создайте временную папку:

```
$ mkdir tmp_pico
```

  iii. Измените каталог на свою временную папку:

```
$ cd tmp_pico
```

  iv. Загрузите скрипт настройки Pico с помощью `wget`:

```
$ wget wget https://raw.githubusercontent.com/raspberrypi/pico-setup/
master/pico_setup.sh
```

  v. Сделайте файл исполняемым:

```
$ chmod +x pico_setup.sh
```

  vi. Выполните скрипт:

```
$ ./pico_setup.sh
```

  vii. Добавьте `$USER` в группу удаленного доступа `dialout`:

```
$ sudo usermod -a -G dialout $USER
```

5. Проверьте взаимодействие Arduino Web Editor с Arduino Nano.
   i. Откройте Arduino Web Editor в браузере.
   ii. Подключите плату Arduino Nano к ноутбуку или ПК с помощью кабеля micro-USB.

   Редактор должен распознать плату в раскрывающемся списке устройств и сообщить для **Arduino Nano 33 BLE** имя порта (например, **/dev/ttyACM0**):

Рис. 1.17 ❖ Ожидаемый результат взаимодействия Arduino Web Editor с Arduino Nano 33

6. Проверьте взаимодействие Arduino Web Editor с Raspberry Pi Pico.
   i. Откройте Arduino Web Editor в браузере.
   ii. Подключите плату Raspberry Pi Pico к ноутбуку или ПК с помощью кабеля micro-USB.

   Редактор должен распознать плату в раскрывающемся списке устройств и сообщить для **Raspberry Pi Pico** имя порта (например, **/dev/ttyACM0**):

Рис. 1.18 ❖ Ожидаемый результат взаимодействия Arduino Web Editor с Raspberry Pi Pico

Мы успешно создали инструменты, которые помогут нам разрабатывать будущие примеры. Прежде чем закончить эту главу, мы хотим протестировать базовый пример на Arduino Nano и Raspberry Pi Pico, чтобы официально отметить начало нашего путешествия в мир TinyML.

Запуск скетча на Arduino Nano 33 и Raspberry Pi Pico

В этом примере мы будем мигать светодиодами Arduino Nano и Raspberry Pi Pico, используя готовый пример мигания из Arduino Web Editor.

Эта программа – аналог «Hello World», управляет встроенным светодиодом, мигающим с помощью периферийного устройства GPIO; через него мы сможем отправиться куда угодно.

Цель этого упражнения – познакомить вас с редактором Arduino Web Editor и помочь понять, как разрабатывать программу с помощью Arduino.

Подготовка

Скетч Arduino состоит из двух функций `setup()` и `loop()`, как показано в следующем фрагменте кода:

```
void setup() {
}
void loop() {
}
```


`setup()` – это первая функция, выполняемая программой, когда мы нажимаем кнопку сброса или включаем питание платы. Эта функция выполняется только один раз и обычно отвечает за инициализацию переменных и периферийных устройств.

После `setup()` программа выполняет бесконечный цикл `loop()`, как показано на следующем рисунке:

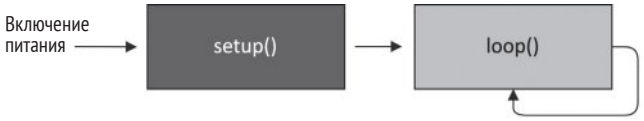


Рис. 1.19 ❖ Схема выполнения скетча

Эти две функции требуются во всех программах Arduino.

Как это делается...

Шаги, описанные в этом разделе, действительны как для Arduino Nano, Raspberry Pi Pico, так и для других плат, совместимых с Arduino Web Editor.

1. Подключите плату к ноутбуку или ПК с помощью кабеля micro-USB. Затем убедитесь, что Arduino IDE сообщает имя и порт устройства.
2. Откройте готовый пример **Blink**: щелкните по **Examples** в меню слева, выберите вкладку **BUILT IN** (т. е. встроенные примеры), а затем **Blink**, как показано на следующем скриншоте:

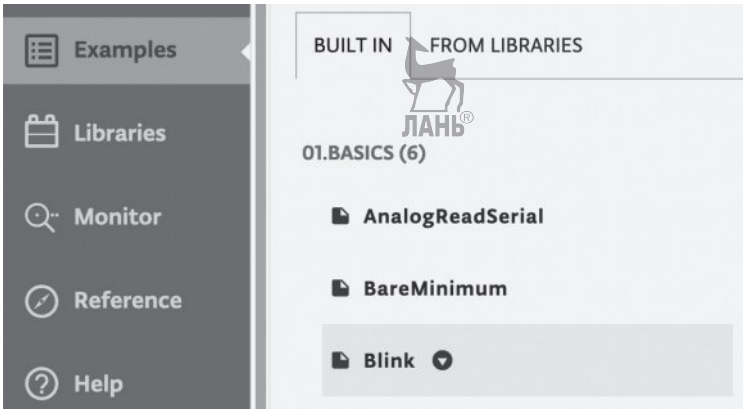


Рис. 1.20 ❖ Пример мигания встроенным светодиодом

После нажатия на название скетча код примера появится в области редактора.

3. Чтобы скомпилировать и загрузить программу на целевое устройство, нажмите на стрелку рядом с раскрывающимся списком плат, как показано на следующем скриншоте:

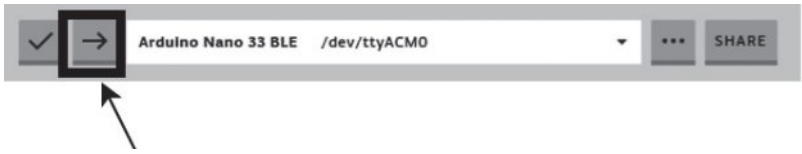


Рис. 1.21 ❖ Стрелка рядом с выпадающим списком плат скомпилирует и прошьет программу на целевом устройстве

По окончании процесса в консоли в нижней части страницы должно появиться сообщение **Done**, а встроенный светодиод должен начать мигать.

Глава 2

Прототипирование на микроконтроллерах

Развертывание приложений **машинного обучения** (**ML**) на микроконтроллерах удобно потому, что разрабатываемая модель живет не только в памяти нашего компьютера. Вместо этого она может оживлять многие вещи вокруг нас. Поэтому, прежде чем погрузиться в мир ML, давайте взглянем на то, как создавать базовые приложения на микроконтроллерах с точки зрения программного и аппаратного обеспечения.

В этой главе мы рассмотрим отладку кода и рассмотрим, как передавать данные на монитор последовательного порта Arduino. Далее мы узнаем, как программировать периферийные устройства GPIO с помощью API **ARM Mbed** и использовать беспаечную **макетную плату** для подключения внешних компонентов, таких как **светодиоды** и **кнопки**. В конце главы мы познакомимся с питанием Arduino Nano и Raspberry Pi Pico от батарей.

Цель этой главы – охватить соответствующие основы программирования микроконтроллеров для следующих разделов этой книги.

В этой главе мы рассмотрим следующие примеры.

- Отладка кода.
- Подключение светодиодного индикатора на макетной плате.
- Управление внешним светодиодом с помощью GPIO.
- Включение и выключение светодиода с помощью кнопки.
- Использование прерываний для считывания состояния кнопки.
- Питание микроконтроллеров от батарей.

Технические требования

Чтобы выполнить все практические примеры из этой главы, понадобится следующее:

- плата Arduino Nano 33 BLE Sense,
- плата Raspberry Pi Pico,
- кабель micro-USB,

- беспаечная макетная плата половинного размера (30 строк и 10 столбцов[22]),
- красный светодиод,
- резистор 220 Ом[23],
- батарейный отсек на 3 элемента типа AA (только для Raspberry Pi Pico),
- батарейный отсек на 4 элемента типа AA (только для Arduino Nano),
- 4 батарейки типа AA,
- тактовая кнопка,
- 5 соединительных проводов-перемычек (штырь–штырь),
- ноутбук/ПК с Ubuntu 18.04+ или Windows 10 на x86-64.

Исходный код и дополнительные материалы доступны в папке *Chapter02* в репозитории GitHub по адресу https://github.com/PacktPublishing/TinyML-Cookbook/tree/main/Chapter02.

Отладка кода

Отладка кода – фундаментальная часть разработки программного обеспечения, направленная на выявление ошибок.

В этом примере будет показано, как выполнить вывод отладочных сообщений для Arduino Nano и Raspberry Pi Pico путем передачи следующих строк на монитор последовательного порта:

- `Initialization completed` (*инициализация завершена*): по окончании инициализации последовательного порта;
- `Executed` (*выполняется*): через каждые две секунды.

Скетч *01_printf.ino*, содержащий код, описанный в этом примере, можно загрузить по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter02/ArduinoSketches/01_printf.ino.

Подготовка

Все программы подвержены ошибкам, и вывод отладочной информации – это базовый процесс, который выводит состояние программы на выходной терминал, как показано в следующем примере:

```
int func (int func_type, int a) {
  int ret_val = 0;
```

[22] В российских интернет-магазинах принято макетные платы сортировать либо по размеру, либо по количеству контактных точек. То, что имеет в виду автор, – плата на 400 точек размерами примерно 82×55 мм.

[23] Для зеленых, желтых и красных светодиодов обычной яркости может заменен на любой резистор в диапазоне сопротивлений 150–470 Ом. Для синих и белых рекомендуется сопротивление поменьше (см. таблицу на стр. 66) – около 80–150 Ом. Для светодиодов повышенной яркости (подобных установленным на различных платах Arduino) сопротивление токоограничивающих резисторов в разы больше – 1 кОм и более.

```
  switch(func_type){
    case 0:
      printf("FUNC0\n");
      ret_val = func0(a)
      break;
    default:
      printf("FUNC1\n");
      ret_val = func1(a);
  }
  return ret_val;
};
```

Чтобы приступить к реализации этого первого примера, нам нужно только знать, как микроконтроллер может отправлять сообщения на монитор последовательного порта.

Язык программирования Arduino предлагает функцию, аналогичную функции `printf()`, в виде функции `Serial.print()`.

Эта функция может отправлять символы, цифры или даже двоичные данные с платы микроконтроллера на компьютер через последовательный порт, обычно называемый UART или USART. За полным списком ее входных аргументов вы можете обратиться по адресу https://www.arduino.cc/reference/en/language/functions/communication/serial/print/.

Как это делается...

Код, приведенный в этом примере, действителен как для Arduino Nano, так и для Raspberry Pi Pico. Arduino IDE скомпилирует код в соответствии с платформой, выбранной в раскрывающемся меню устройств.

Откройте Arduino IDE и создайте новый пустой проект, нажав на **Sketchbook** в крайнем левом меню (**EDITOR**), а затем нажмите на **NEW Sketch**, как показано на рис. 2.1.

Как мы видели в главе 1 «*Начало работы с TinyML*», для всех скетчей требуется файл, содержащий функции `setup()` и `loop()`.

Следующие шаги демонстрируют, что должны содержать эти функции, чтобы реализовать вывод отладочной информации.

1. Инициализируйте скорость передачи данных UART в бодах в функции `setup()` и подождите, пока это периферийное устройство не будет открыто:

   ```
   void setup() {
     Serial.begin(9600);
     while (!Serial);
   ```

   В отличие от стандартной библиотечной функции `printf` в языке C, функция `Serial.print()` требует инициализации перед передачей данных. Поэтому мы инициализируем периферийное устройство с помощью функции Arduino `Serial.begin()`, которая требует только скорости передачи данных в бодах в качестве входного аргумента. Скорость

передачи данных в бодах в данном случае равна скорости передачи в битах в секунду, здесь она устанавливается на величину 9600 бит/с. Мы не можем использовать это периферийное устройство непосредственно после инициализации, потому что должны подождать, пока оно не будет готово к передаче. Поэтому мы используем `while(!Serial)`, чтобы дождаться открытия последовательной связи.

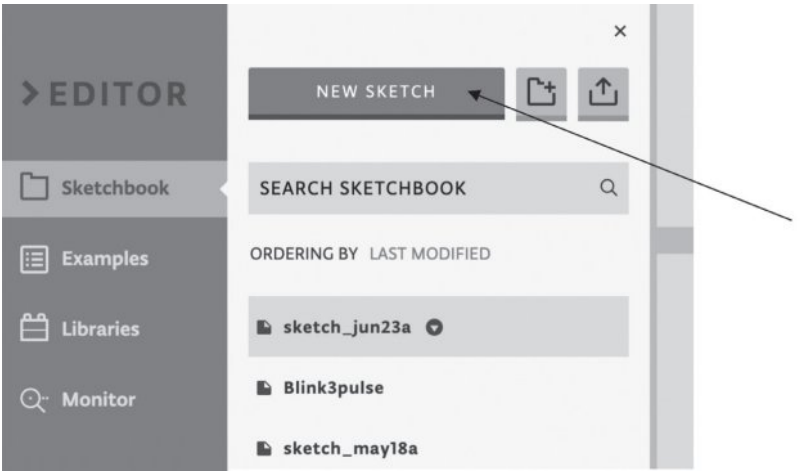


Рис. 2.1 ❖ Нажмите на кнопку **NEW SKETCH**, чтобы создать новый проект

2. Выводим `Initialization completed` после `Serial.begin()` в функции `setup()`:

```
  Serial.print("Initialization completed\n");
}
```

Мы передаем `Initialization completed` с помощью `Serial.print("Initialization completed\n")`, чтобы сообщить о завершении инициализации.

3. Выводим `Executed` каждые две секунды в функции `loop()`:

```
void loop() {
  delay(2000);
  Serial.print("Executed\n");
}
```

Поскольку функция `loop()` вызывается итеративно, мы используем функцию Arduino `delay()` для приостановки выполнения программы на две секунды. Функция `delay()` использует в качестве входного аргумента длительность времени в миллисекундах (1 с = 1000 мс).

Теперь убедимся, что устройство подключено к компьютеру с помощью кабеля micro-USB.

Если устройство распознано средой, мы можем открыть монитор последовательного порта, нажав на **Monitor** в меню **Editor**. В нем мы увидим любые

данные, передаваемые микроконтроллером через периферийное устройство UART. Однако перед началом любого обмена данными убедитесь, что монитор последовательного порта использует ту же скорость передачи данных, что и периферийное устройство микроконтроллера (9600), как показано на следующем скриншоте:

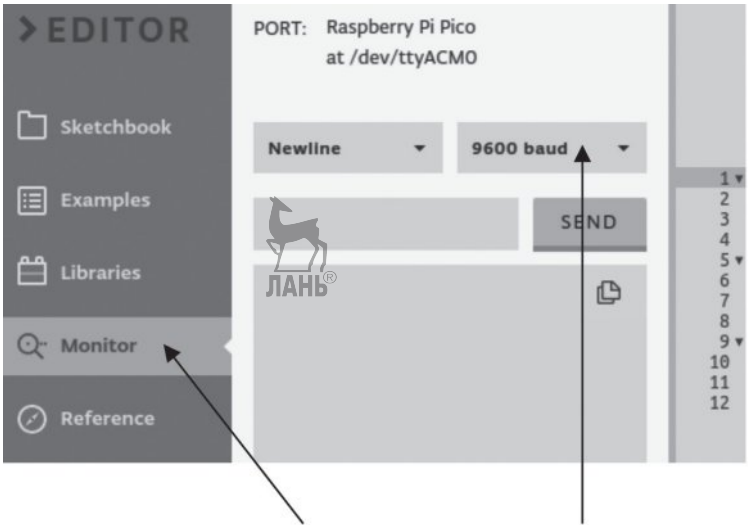


Рис. 2.2 ❖ Монитор последовательного порта должен использовать ту же скорость передачи данных, что и UART

При открытом мониторе последовательного порта мы можем нажать на стрелку рядом с раскрывающимся меню устройства, чтобы скомпилировать и загрузить программу на целевую платформу. Как только скетч будет загружен, монитор последовательного порта получит сообщения о завершении инициализации и выполнении, как показано на следующем скриншоте:

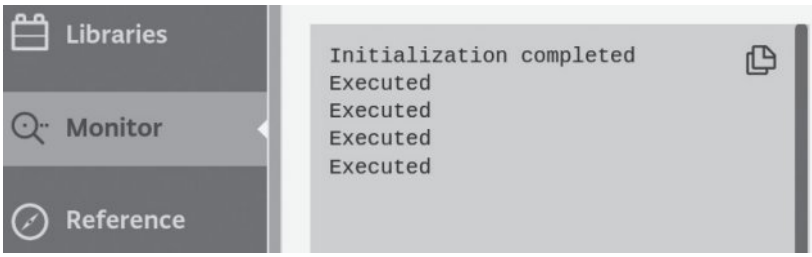


Рис. 2.3 ❖ Ожидаемый результат на мониторе последовательного порта

Как мы можем видеть из вывода на мониторе, `Initialization completed` выводится один раз, потому что функция `setup()` вызывается только при запуске программы.

Дополнительно

Вывод отладочной информации – это простой подход к отладке, но он имеет существенные недостатки, связанные с увеличением сложности программного обеспечения, например:

- необходимость повторной компиляции и прошивки платы каждый раз, когда мы добавляем или перемещаем `Serial.print()`;
- стоимость `Serial.print()` с точки зрения объема памяти программы;
- мы могли бы допустить ошибки в сообщениях – например, используя `print()` для вывода беззнаковой целой переменной (типа `unsigned int`), которая на самом деле имеет знак[24].

В этой книге мы не будем рассматривать более продвинутую отладку, но рекомендуем ознакомиться с отладчиками **serial wire debug** (**SWD**) (https://developer.arm.com/architectures/cpu-architecture/debug-visibility-and-trace/coresight-architecture/serial-wire-debug), чтобы сделать этот процесс менее напряженным. SWD – это протокол отладки с помощью всего двух проводов, пригодный почти для всех процессоров ARM Cortex, который вы можете использовать для прошивки микроконтроллера, пошагового выполнения кода, добавления точек останова и т. д.

Подключение светодиодного индикатора на макетной плате

У нас есть возможность взаимодействовать с окружающим миром с помощью микроконтроллеров. Например, мы можем получать данные от датчиков или выполнять физические действия, такие как включение и выключение светодиода или перемещение исполнительного механизма.

В этом примере мы узнаем, как подключить внешние компоненты к микроконтроллеру, построив на макетной плате электронную схему (рис. 2.4).

В показанной схеме используется красный светодиод, указывающий, подключен ли микроконтроллер к источнику питания.

Подготовка

Под подключением внешних компонентов к микроконтроллеру мы подразумеваем физическое соединение двух или более разъемов с металлическими контактами. Хотя мы могли бы припаять эти контакты, при создании прототипов так не делается, потому что это долго и сложно.

---

[24] Следует добавить к этому перечню довольно большое время вывода информации через последовательный порт: при скорости 9600 время передачи одного байта составляет около 1 мс. То есть для строки «*Initialization completed*», как в приведенном примере, время передачи составит около 25 мс, что может сказаться на работе программ, критичных ко времени выполнения.

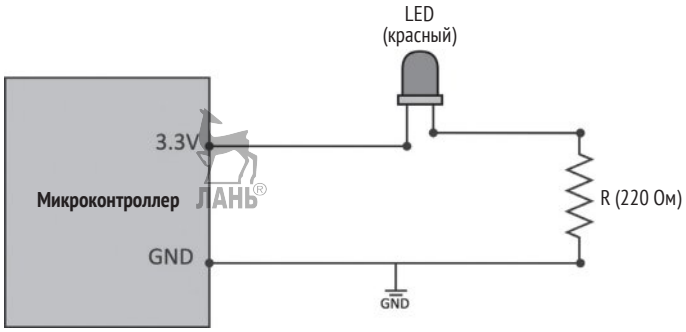


Рис. 2.4 ❖ Схема светодиодного индикатора состояния питания

Таким образом, цель этого раздела «*Подготовка*» – представить альтернативу пайке для легкого соединения наших компонентов.

Подключение контактов непосредственно к выводам микроконтроллера может быть чрезвычайно затруднено из-за крошечного расстояния между выводами. Например, у микроконтроллера RP2040 размер чипа составляет 7×7 мм, а расстояние между выводами составляет примерно 0,5 мм. Большинство клемм на соединительных проводах имеет диаметр ~ 1 мм, и было бы практически невозможно безопасно подключить какой-либо из наших компонентов.

По этой причине рассматриваемые микроконтроллерные платформы обеспечивают альтернативные точки подключения с бóльшим расстоянием на плате между ними. Эти точки на Arduino Nano и Raspberry Pi Pico представляют собой два ряда предварительно просверленных металлизированных отверстий, расположенных на краю платы.

Самый простой способ узнать соответствие между этими контактами и выводами микроконтроллера – посмотреть технические характеристики плат. Поставщики оборудования обычно предоставляют схему разводки внешних выводов, где отмечают расположение выводов и их функциональность.

Вот ссылки на схемы разводки выводов для плат Arduino Nano и Raspberry Pi Pico:

- **Arduino Nano 33**: https://content.arduino.cc/assets/Pinout-NANOsense_latest.pdf,
- **Rasberry Pi Pico**: https://datasheets.raspberrypi.org/pico/Pico-R3-A4-Pinout.pdf.

В эти предварительно просверленные отверстия, которые часто имеют расстояние между собой 2.54 мм[25], мы можем припаять разъем, чтобы легко соединять электронные компоненты. Припаиваемая часть разъема может быть либо штыревой (*male* или *pin*), либо гнездовой (*female* или *socket*), как показано на следующем рисунке:

[25] В некоторых аппаратных модулях и датчиках шаг отверстий для разъема составляет 2 мм и с обычной макетной платой (см. далее) не совмещается. В этих случаях приходится использовать переходные платы на шаг 2.54 мм.

Штыревой (male) разъем Гнездовой (female) разъем

Рис. 2.5 ❖ Штыревая и гнездовая части игольчатых разъемов

Если вы не знакомы с пайкой или просто хотите получить готовое решение, мы рекомендуем покупать устройства с предварительно припаянными разъемами.

Как мы видим, платы обеспечивают способ подключения внешних компонентов к микроконтроллеру. Однако как мы можем присоединить другие электрические элементы для построения полной электронной схемы?

Размещение прототипа на макетной плате

Беспаечная макетная плата представляет собой платформу для создания схем прототипов без пайки – вставкой выводов устройства в сетку отверстий с металлическими контактами внутри:

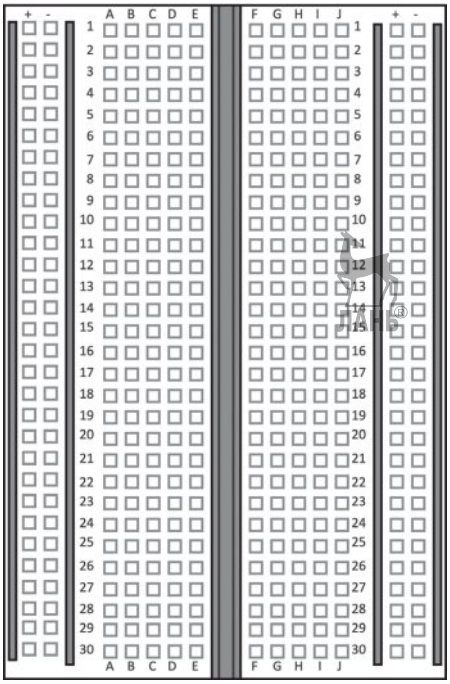


Рис. 2.6 ❖ Беспаечная макетная плата

Показанная на рисунке макетная плата обеспечивает две области соединения для компонентов.

- **Шины питания** обычно расположены с обеих сторон макетной платы и состоят из двух рядов отверстий, обозначенных символами + и –, как показано на следующем рисунке:

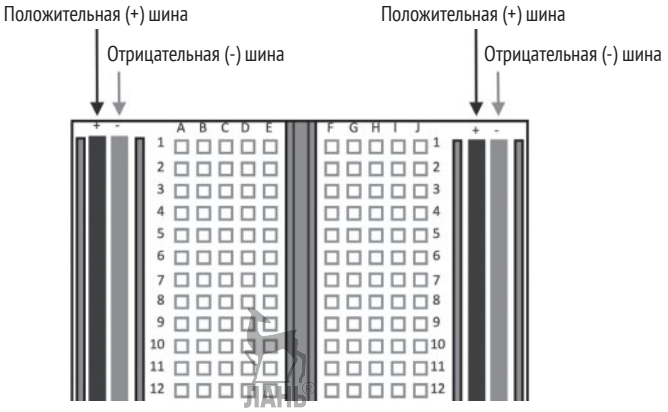


Рис. 2.7 ❖ Шины питания, обозначенные + и – с обеих сторон макетной платы

Все отверстия одной и той же шины соединены между собой внутри. Следовательно, мы будем иметь одинаковое напряжение во всех отверстиях при подаче напряжения на любое из них. Мы никогда не должны подавать разные напряжения на отверстия одной и той же шины.

- **Контактные ряды** расположены в центральной части макетной платы, в них отверстия соединяются между собой только в одном ряду, так что происходит следующее:
  – отверстия в одном ряду имеют одинаковое напряжение,
  – отверстия в одной и той же колонке могут иметь разное напряжение.
  Однако, поскольку у нас обычно есть выемка, проходящая в середине по всей макетной плате, в каждом ряду слева и справа образуются две разные колодки, как показано на следующем рисунке:

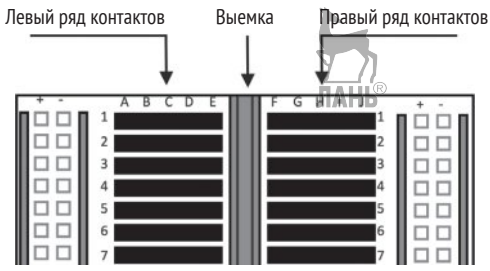


Рис. 2.8 ❖ Ряды контактов расположены в центральной части макетной платы

Мы можем разместить несколько устройств на макетной плате и соединить их с помощью перемычек.

Размер макетной платы определяется количеством строк и столбцов в области терминала. В нашем случае мы всегда будем ссылаться на макет половинного размера с 30 строками и 10 столбцами[26].

Как это делается...

Перед созданием каких-либо схем отсоедините кабель micro-USB от платы микроконтроллера, чтобы исключить возможность непреднамеренного повреждения каких-либо компонентов.

После того как мы отсоединили плату от источника питания, для построения схемы включения светодиода при включении питания выполните следующие действия.

1. Установите плату микроконтроллера на макетную плату.

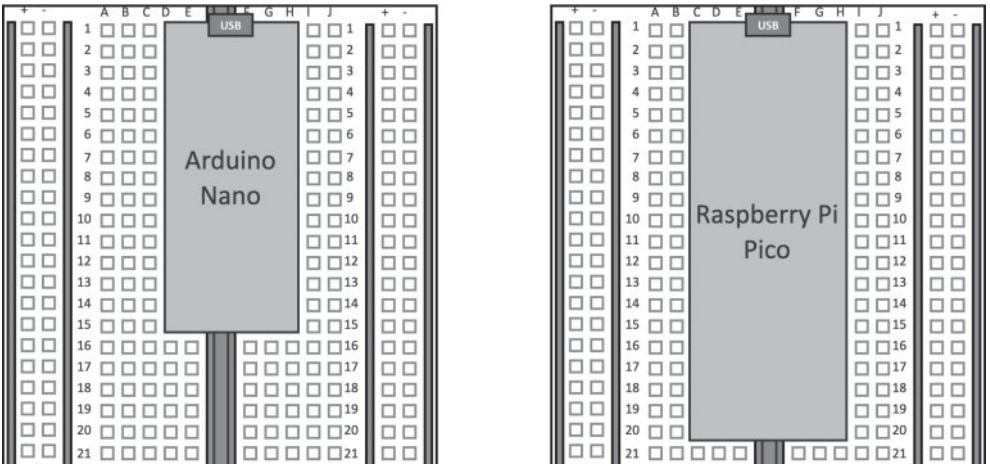


Рис. 2.9 ❖ Вертикально установите плату микроконтроллера между левыми и правыми шинами питания

Поскольку у нас есть выемка в середине, безопасно размещать платы микроконтроллеров именно таким образом – тогда штыри разъемов слева и справа контактируют с разными рядами контактов.

2. Используйте два провода-перемычки для подключения выводов 3.3 В и GND платы микроконтроллера к шинам + и – (рис. 2.10).
   Важно отметить, что все отверстия шин питания будут иметь напряжения соответственно 3.3 В и GND только тогда, когда микроконтроллер подключен к источнику питания.

[26] По поводу соответствия общепринятым в России обозначениям макетных плат см. первую сноску на стр. 54.

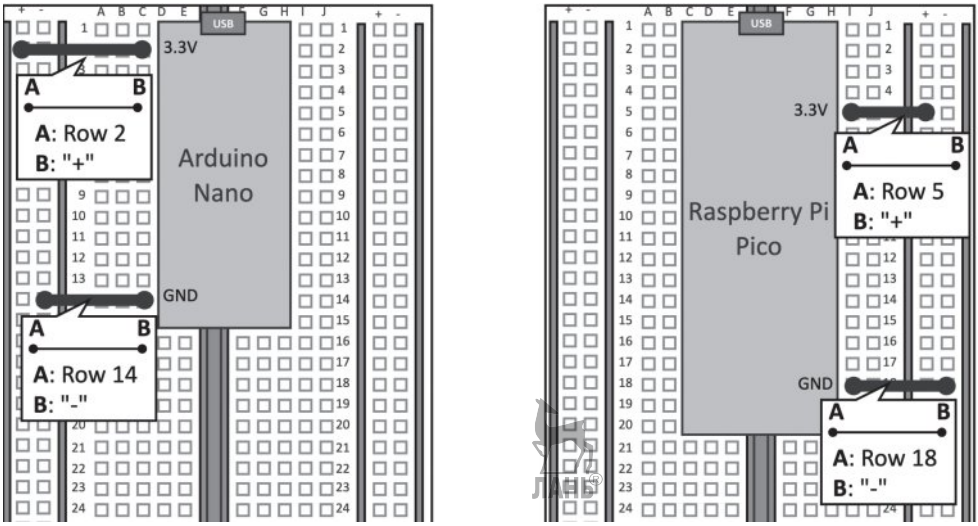


Рис. 2.10 ❖ Используйте перемычки для подключения 3.3 В и GND к шинам + и –

3. Вставьте контакты светодиода в два разных ряда контактов. На рис. 2.11 мы вставляем более длинный вывод светодиода в отверстие (G, 24), а более короткий – в отверстие (G, 25). Не путайте длинные и короткие выводы, так как в этом случае светодиод не загорится[27].

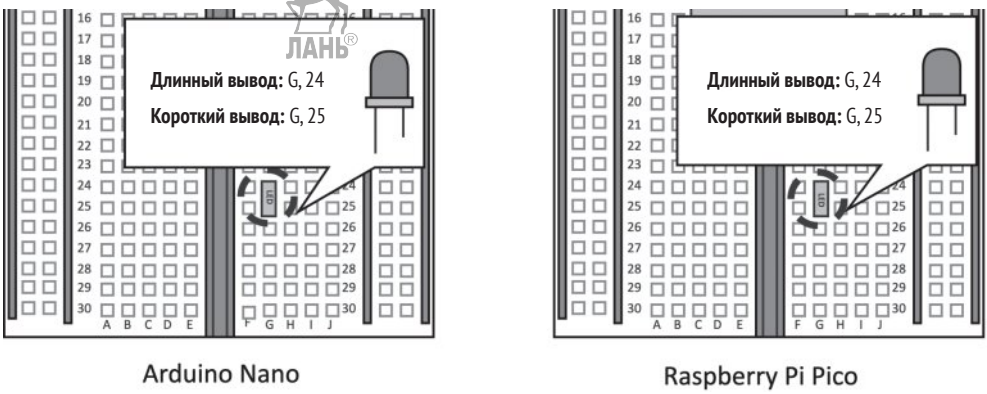


Рис. 2.11 ❖ Вставьте светодиод в макетную плату

[27] Если выводы светодиода были укорочены и невозможно определить, какой из них был длиннее, то у большинства светодиодов в круглых корпусах (3 и 5 мм) полярность можно определить по наличию скоса (сточенного бортика) на корпусе, которым обозначается отрицательный вывод (катод).

4. Установите последовательно со светодиодом резистор 220 Ом.

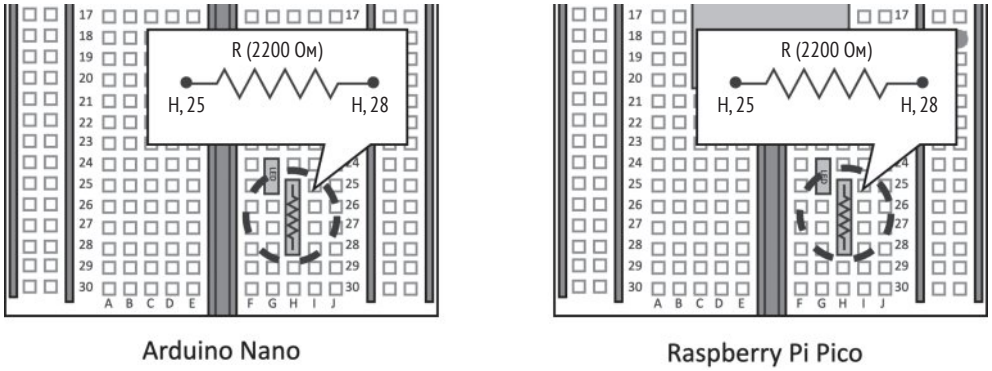


Рис. 2.12 ❖ Установите резистор последовательно со светодиодом

Цветной код резистора можно декодировать с помощью онлайн-калькулятора (см. ссылку на стр. 38). Резистор 220 Ом с пятью или шестью полосками будет кодироваться следующими цветами (по порядку от первой до четвертой полоски): красный–красный–черный–черный.

Как указано на схеме, представленной в начале этого примера, один вывод резистора должен соединяться с коротким выводом светодиода. В нашем случае мы вставляем один вывод в отверстие (H, 25), расположенное на одной контактной колодке с (G, 25), куда подключен короткий вывод светодиода. Оставшийся вывод резистора подключается к любой неиспользуемой контактной колодке. В нашем случае мы вставляем этот вывод в (H, 28).

5. Замкните цепь, подсоединив шину + (3.3 В) к длинному выводу светодиода, а шину – (GND) – к свободному выводу резистора:

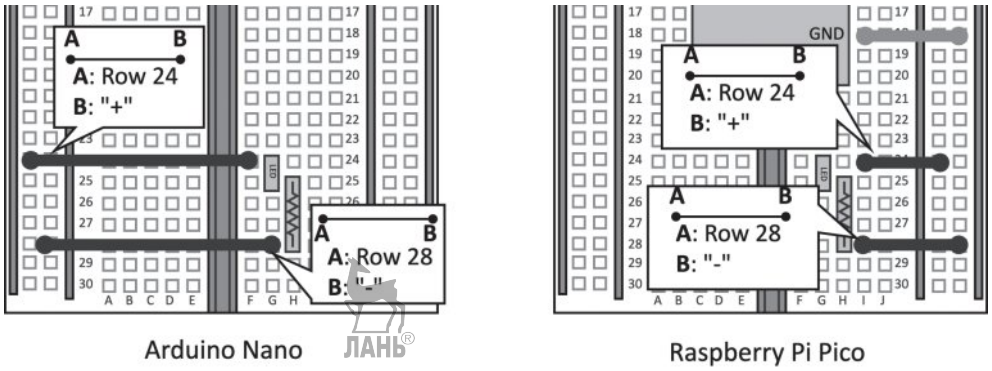


Рис. 2.13 ❖ Замкните цепь, подключив 3.3 В и GND

На рисунке показано, как подключить два оставшихся провода-перемычки, использовав их для замыкания цепи. Одна перемычка соединя-

ет шину + с длинным выводом светодиода (H, 24), в то время как другая соединяет шину – с резистором (H, 28).

Теперь светодиод должен светиться всякий раз, когда вы подключаете микроконтроллер к источнику питания с помощью кабеля micro-USB.

Управление внешним светодиодом с помощью GPIO

В настоящее время светодиоды используются повсюду, особенно в наших домах, потому что они при той же интенсивности света потребляют меньше энергии, чем старые лампы. Однако светодиоды, рассматриваемые в наших экспериментах, – это не лампочки, а сигнальные светодиоды с гибкими выводами для установки в отверстия, пригодные для быстрого прототипирования на макетной плате.

В этом примере мы узнаем, как построить базовую схему с внешним светодиодом и запрограммировать вывод GPIO для управления его свечением.

Скетч *03_gpio_out.ino*, содержащий код, описанный в этом примере, можно загрузить по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter02/ArduinoSketches/03_gpio_out.ino.

Подготовка

Чтобы выполнить этот пример, нам нужно знать, как работает светодиод и как запрограммировать периферийное устройство, управляющее периферийными устройствами GPIO микроконтроллера в режиме выхода.

Общепринятое международное название светодиода LED расшифровывается как **Light-Emitting Diode**, что и переводится как светоизлучающий диод. Он представляет собой полупроводниковый компонент, излучающий свет, когда через него протекает ток.

Сигнальные светодиоды имеют следующие основные конструктивные части:

- головку из прозрачного материала, из которой исходит свет. Головка может быть разного размера и формы. Круглые светодиоды обычно поставляется в размерах 3, 5 или 10 мм;
- две ножки разной длины для отличия положительного вывода (анода) от отрицательного (катода). Анод – это более длинный вывод.

На рис. 2.14 показан типовой внешний вид сигнального светодиода и его символическое обозначение в электронной схеме.

Как уже упоминалось, светодиод излучает свет, когда через него протекает ток. Однако, в отличие от резисторов, ток через светодиод течет только в одном направлении, а именно от анода к катоду. Этот ток обычно называют прямым током (I_f).

Яркость светодиода пропорциональна I_f, поэтому чем она выше, тем ярче он будет светиться. Светодиод имеет максимальный рабочий ток, который

мы не должны превышать, чтобы избежать его выхода из строя. Для стандартных сигнальных светодиодов в корпусе 5 мм максимальный ток обычно составляет 20 мА, поэтому значений от 4 до 15 мА должно быть достаточно, чтобы увидеть, как светодиод излучает свет[28].

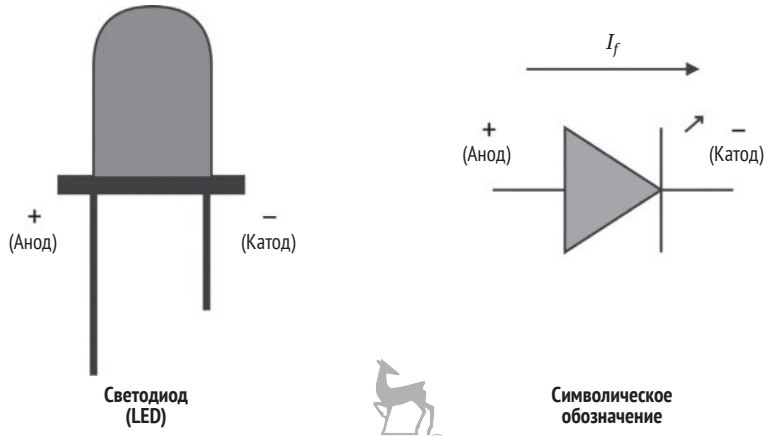


Рис. 2.14 ❖ Светодиод и его символическое обозначение на схемах

Чтобы обеспечить протекание тока, нам нужно подать определенное напряжение на выводы светодиода, называемое прямым напряжением (V_f). Мы определяем V_f в соответствии с формулой:

$V_f = V_{\text{anode}} - V_{\text{cathode}}$.

Типичный диапазон V_f для некоторых цветов свечения приведен в следующей таблице:

Таблица 2.1. Типовое прямое напряжение светодиода

Цвет свечения	**Прямое напряжение, В**
Красный	1.8–2.1
Оранжевый	1.9–2.2
Желтый	1.9–2.2
Зеленый	2.0–3.1
Синий	3.0–3.7
Белый	3.0–3.4

[28] Значение 20 мА для круглых светодиодов в корпусе 3 и 5 мм обычно соответствует номинальному току (т. е. току, при котором все параметры находятся в оговоренных пределах). Максимально допустимый ток для них обычно составляет 25–35 мА, причем импульсный ток (в зависимости от длительности импульса и продолжительности паузы) может быть в разы больше. Однако автор совершенно прав в оценке нормального рабочего диапазона в 4–15 мА – нет никакой нужды выдерживать светодиод близко к верхнему пределу токов, его свечение все равно будет хорошо заметно. Если зачем-то необходима увеличенная яркость (хотя для круглых светодиодов с корпусом-линзой высокая яркость без крайней нужды не рекомендуется), то надежнее просто поставить светодиод повышенной яркости свечения при том же токе.

Из приведенной таблицы мы можем сделать следующие выводы о диапазоне прямого напряжения:

- он зависит от цвета;
- он довольно узкий и в большинстве случаев меньше, чем типичное напряжение 3.3 В, необходимое для питания микроконтроллера.

Из этих наблюдений на ум приходят три вопроса.

- Для начала: как мы можем подать прямое напряжение на выводы светодиодов, поскольку обычно у нас есть только 3.3 В от микроконтроллера?
- Что произойдет, если мы подадим напряжение ниже минимального V_f?
- Что произойдет, если мы подадим напряжение выше максимального V_f?

Ответы основаны на рассмотрении вида физической зависимости между напряжением и током светодиода:

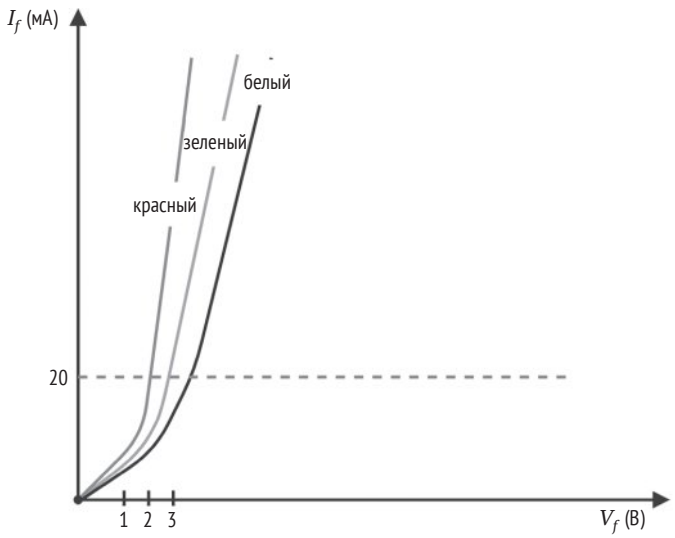


Рис. 2.15 ❖ Вольт-амперная характеристика светодиода

Из этого графика, где оси x и y отображают напряжение и ток, мы можем вывести следующее:

- если бы мы подали на светодиод напряжение намного ниже V_f, светодиод не включился бы, потому что ток был бы слишком низким;
- если бы мы подали на светодиод напряжение, намного превышающее V_f, светодиод был бы поврежден, поскольку ток превысил бы максимально допустимый предел.

Поэтому фиксация напряжения на требуемом рабочем значении V_f имеет решающее значение для обеспечения того, чтобы устройство работало и не было повреждено. Решение задачи простое и требует только резистора, подключенного последовательно со светодиодом, как показано на следующем рисунке:

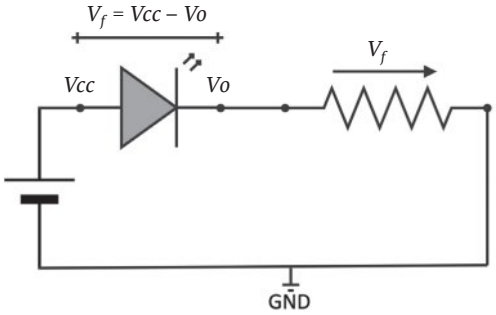


Рис. 2.16 ❖ Резистор, подключенный последовательно со светодиодом, ограничивает ток на нужном уровне

Из этого разъяснения должно быть ясно, почему мы включили резистор в схему предыдущего примера. Поскольку светодиод имеет фиксированное падение напряжения V_f, когда он излучает свет, резистор ограничивает ток на нужном нам значении, например в диапазоне 4–15 мА. Наличие резистора с правильно подобранным сопротивлением означает, что V_f не выйдет за пределы ожидаемого рабочего диапазона.

Мы можем рассчитать значение сопротивления резистора, используя следующую формулу:

$$R = \frac{Vcc - V_f}{I_f},$$

где:

- V_f – прямое напряжение;
- I_f – прямой ток;
- R – сопротивление резистора.

Информация о прямом напряжении, токе и яркости светодиода, как правило, доступна в техническом описании светодиодов[29].

Теперь давайте посмотрим, как мы можем контролировать состояние этого устройства с помощью управления периферийными устройствами GPIO.

[29] Из рассмотрения графика (рис. 2.15) видно, что прямое падение напряжения V_f сильно зависит от значения тока; кроме того, из табл. 2.1 следует, что сам график может иметь большой разброс для разных светодиодов одного цвета свечения. Поэтому точный расчет значения сопротивления невозможен и не требуется: разницу в свечении из-за этих неточностей вы не увидите, вполне можно обойтись приблизительными пределами сопротивлений, указанными на стр. 54 в второй сноске. Однако не следует забывать, что для синих и белых светодиодов значение прямого падения может превысить напряжение на выходном выводе микроконтроллера с питанием 3.3 В, и тогда светодиод вообще не загорится, поэтому их лучше предварительно проверить, подобрав рабочие пределы сопротивления экспериментально для конкретной схемы.

Представляем периферийное устройство GPIO

Универсальный ввод/вывод (**General-purpose input/output, GPIO**) – наиболее распространенное и универсальное периферийное устройство в микроконтроллерах. Как следует из названия, GPIO не имеет фиксированной функциональности. Вместо этого его основная функция заключается в установке (выводе) или считывании (вводе) цифровых сигналов (1 или 0) через внешние контакты, обычно называемые GPIO, IO или GP.

Микроконтроллер может интегрировать несколько периферийных устройств GPIO, где каждое из них может управлять выделенным выводом интегрированной микросхемы.

Поведение GPIO аналогично поведению `std::cout` и `std::cin` библиотеки `iostream` C++, но с той разницей, что он записывает и считывает фиксированные напряжения, а не символы.

Обычно применяемые напряжения для логических уровней 1 и 0 следующие:

Таблица 2.2. Логические уровни и соответствующие напряжения[30]

Логический уровень	**Напряжение (*V*)**
1 (HIGH)	Vcc (напряжение питания, например, 3.3 или 5 V)
0 (LOW)	GND

Рассмотренный выше пример мигания светодиода является типичным примером настройки периферийного устройства GPIO в режиме выхода для подачи 3.3 В (1) или 0 В (0) программным способом.

Существует два способа подключения светодиода к выводу GPIO, различающиеся направлением тока. Первый способ – это источник тока, при котором ток вытекает из платы микроконтроллера. Для этого нам нужно сделать следующее.

- Подключите анод светодиода к контакту GPIO.
- Последовательно подключите катод светодиода к резистору.
- Подключите оставшийся вывод резистора к GND.

Схема на рис. 2.17 показывает, как управлять светодиодом с помощью схемы источника тока.

Из показанной схемы мы можем заключить, что для включения светодиода вывод GPIO должен иметь логический уровень 1.

---

[30] Следует отметить, что в реальных (в том числе и микроконтроллерных) схемах уровни напряжения, соответствующие логическому нулю и единице, могут заметно отличаться от идеального случая совпадения с напряжением питания или нулем. Поэтому на практике в цифровых схемах для нуля (низкого уровня) и единицы (высокого уровня) обычно устанавливаются пороговые значения, при выходе за которые значение сигнала не определено. Пороговые значения всегда приводятся в документации на соответствующий компонент: например, для контроллера nRF52840 (Arduino Nano) в режиме GPIO входной уровень единицы не должен быть меньше 0,7×Vcc, а входной уровень нуля – больше 0,3×Vcc. Большинство современных компонентов общего применения имеют пороги, близкие к таким же значениям.

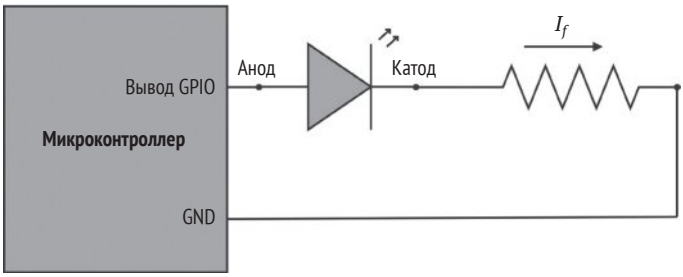


Рис. 2.17 ❖ Источник тока.
Ток вытекает из платы микроконтроллера

Второй способ противоположен первому – это приемник тока, при котором ток втекает в плату микроконтроллера. В этом случае нужно сделать следующее.

- Подключите катод светодиода к контакту GPIO.
- Последовательно подключите анод светодиода к резистору.
- Подключите оставшийся вывод резистора к напряжению 3.3 В.

Как мы можем видеть из следующей схемы, для включения светодиода вывод GPIO должен иметь логический уровень 0:

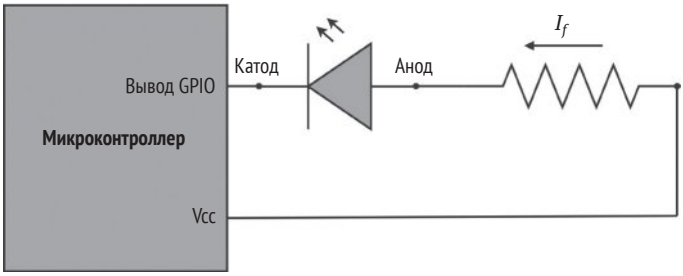


Рис. 2.18 ❖ Приемник тока.
Ток поступает в плату микроконтроллера

Какое бы решение мы ни приняли, важно иметь в виду, что вывод МК имеет ограничения по максимальному току, которые в принципе могут отличаться в зависимости от его направления. При проектировании схемы для управления светодиодом мы всегда должны учитывать эти ограничения, чтобы правильно работать и не повредить устройство.

Как это делается...

Отключите платы микроконтроллера от источника питания и оставьте светодиод и резистор на макетной плате, как в предыдущем примере (рис. 2.13). Однако отсоедините все провода-перемычки, кроме той, которая подключает плату к шине GND. На следующем рисунке показано, как должна выглядеть ваша макетная плата:

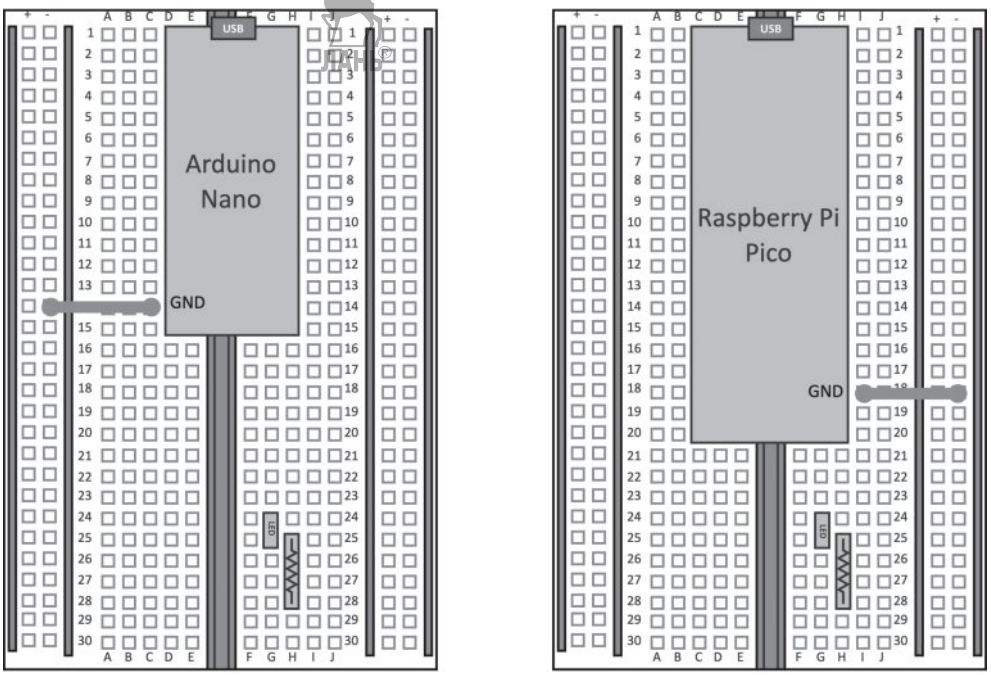


Рис. 2.19 ❖ Сохраняем плату микроконтроллера, светодиод и резистор от схемы светодиодного индикатора состояния на макетной плате

Поскольку катод светодиода подключен к выводу резистора, светодиод будет приводиться в действие цепью источника тока[31].

Следующие шаги покажут, как управлять свечением светодиода с помощью GPIO- периферии.

1. Выберите вывод GPIO для управления светодиодом. В следующей таблице представлен наш выбор:

Таблица 2.3. Вывод GPIO для управления светодиодом

Плата	**Вывод GPIO**
Arduino Nano	P0.23
Raspberry Pi Pico	GP22

[31] Следует учитывать, что для обоих случаев (втекающего или вытекающего тока) порядок включения резистора последовательно со светодиодом не имеет значения. Так, схему источника (рис. 2.17) можно переделать в схему приемника, просто переключив анод светодиода от GPIO к Vcc, а правый (по схеме) вывод резистора от GND к выводу GPIO. И наоборот: схема приемника (рис. 2.18) переделывается в схему источника, если катод светодиода вместо GPIO подключить к GND, а правый (по схеме) вывод резистора вместо Vcc к GPIO. Связку светодиод+резистор следует рассматривать как единый компонент, полярность которого зависит от ориентации светодиода относительно полярности управляющего напряжения, а не от расположения резистора «до» или «после» светодиода. Смотрите также практические примеры подключения на рис. 4.34 и 4.39.

2. Подсоедините анод светодиода к выбранному контакту GPIO с помощью перемычки:

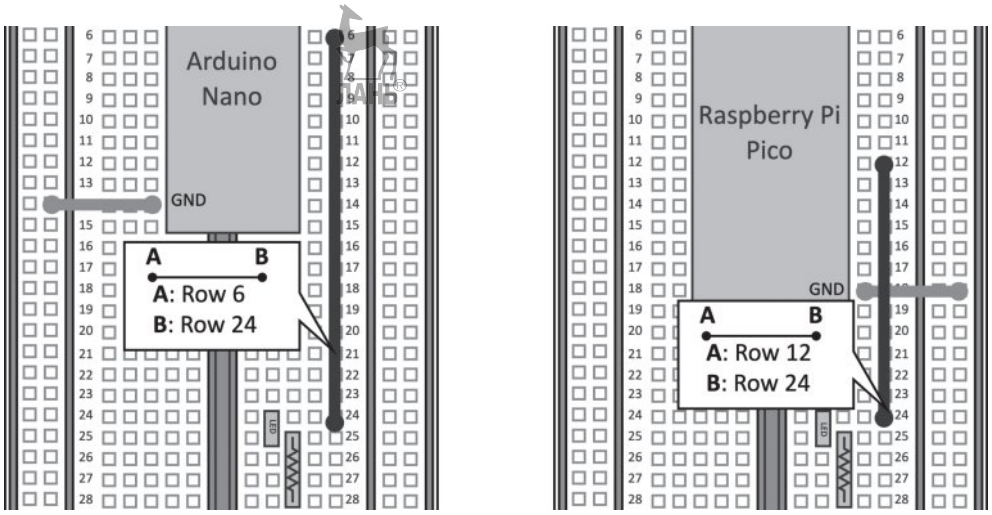


Рис. 2.20 ❖ Подсоедините анод светодиода к контакту GPIO

На Arduino Nano мы используем перемычку для соединения (J, 6) с (J, 24). На Raspberry Pi Pico мы используем перемычку для соединения (J, 12) с (J, 24).

3. Подключите вывод резистора к GND:

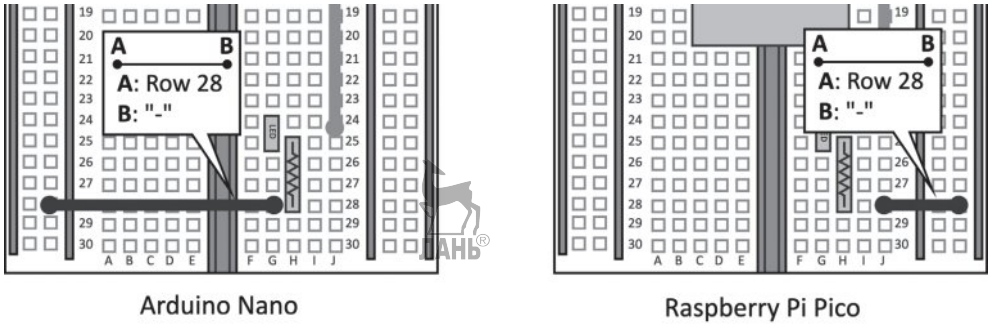


Рис. 2.21 ❖ Подключение резистора к GND

Как на Arduino Nano, так и на Raspberry Pi Pico мы подключаем (J, 28) к шине GND.

Резистор 220 Ом при напряжении 3.3 В обеспечивает ток светодиода ~ 5–7 мА, что ниже допустимого тока светодиода 20 мА и ниже максимального рекомендуемого выходного тока GPIO, указанного в следующей таблице:

Таблица 2.4. Максимальный рекомендуемый ток вывода GPIO

Плата	Максимальный рекомендуемый ток вывода GPIO
Arduino Nano	12
Raspberry Pi Pico	10

После подготовки схемы можно приступить к программированию GPIO.

4. Откройте Arduino IDE и создайте новый скетч. Объявите и инициализируйте глобальный объект `mbed::DigitalOut` с именем вывода, используемого для управления светодиодом.
   Для Arduino Nano у нас есть следующее:

```
mbed::DigitalOut led(p23);
```

   То же самое для Raspberry Pi Pico:

```
mbed::DigitalOut led(p22);
```

   Mbed, или, скорее, Mbed OS (https://os.mbed.com/), представляет собой операционную систему реального времени (RTOS) специально для процессоров ARM Cortex-M. Mbed OS предлагает функциональность типичной канонической ОС и драйверы для управления периферийными устройствами микроконтроллера. Все программы на плате Arduino Nano 33 BLE Sense и Raspberry Pi Pico построены поверх этой малой операционной системы. В этом примере мы используем объект Mbed OS `mbed::DigitalOutput` (https://os.mbed.com/docs/mbed-os/v6.15/apis/digitalout.html) для взаимодействия с периферийным устройством GPIO в режиме вывода. Для инициализации периферийного устройства требуется название вывода GPIO (`PinName`), подключенного к светодиоду. `PinName` всегда начинается с буквы p, за которой следует номер вывода. На Arduino Nano номер вывода равен величине `y`, указанной в обозначении вывода `P<x>.<y>`[32]. Следовательно, `PinName` – это `p23`.
   На Raspberry Pi Pico pin-код равен величине `y`, указанной в обозначении `GPy`. Следовательно, `PinName` – это `p22`.
5. Для включения светодиода установите в функции `loop()` для переменной `led` значение 1:

```
void loop() {
  led = 1;
}
```

Скомпилируйте скетч и загрузите программу в микроконтроллер.

---

[32] Смотрите полную схему разводки выводов Arduino Nano 33 BLE Sense, например http://wiki.amperka.ru/_media/products:arduino-nano-33-ble:arduino-nano-33-ble-pinout.png. Первая цифра (x в обозначениях автора) означает номер порта, к которому относится данный вывод.

Включение и выключение светодиода с помощью кнопки

В отличие от ПК, где клавиатура, мышь или даже сенсорный экран облегчают взаимодействие человека с программными приложениями, физическая кнопка представляет собой самый простой способ взаимодействия пользователя с микроконтроллером.

Этот пример научит вас, как запрограммировать GPIO для считывания состояния кнопки (нажата или отпущена) при управлении свечением светодиода.

Код этого примера содержится в файле *04_gpio_in_out.ino*, доступном по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter02/ArduinoSketches/04_gpio_in_out.ino.

Подготовка

Для подготовки к этому примеру нам нужно знать, как работает кнопка, и запрограммировать периферийное устройство GPIO в режиме ввода.

Тактовая кнопка – это тип кнопки, используемой в микроконтроллерах. Она ведет себя как логическое устройство, поскольку имеет только два состояния – может быть либо нажата (*true*), либо отпущена (*false*).

С точки зрения электроники кнопка – это устройство, которое соединяет (замыкает) или разрывает (размыкает) два проводника. Когда мы нажимаем кнопку, мы соединяем провода через механическую систему контактов, позволяя течь току. Однако это не похоже на стандартный выключатель света, который удерживает провода подключенными при отпускании. Пока мы не нажимаем на кнопку, провода разомкнуты и ток не течет.

Хотя стандартная тактовая кнопка имеет четыре металлические ножки, она является двухполюсным устройством, поскольку выводы на противоположных сторонах (1, 4 и 2, 3) составляют одно целое, как показано на следующем рисунке:

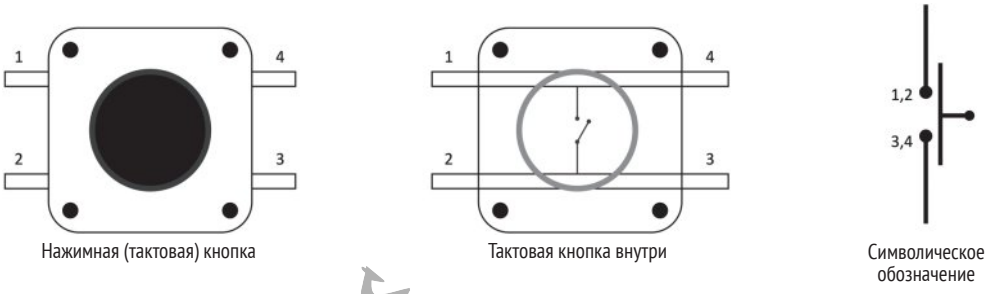


Рис. 2.22 ❖ Представление кнопки

При построении схемы с помощью этого компонента ножки на одной стороне (1,2 или 4,3 на предыдущем рисунке) отвечают за соединение двух

точек цепи. Эти две точки при нажатии кнопки будут иметь одинаковое напряжение.

Состояние кнопки может быть считано с помощью вывода GPIO в режиме входа. При настройке GPIO в режиме входа периферийное устройство считывает приложенное напряжение на выводе для определения логического уровня. По этому значению мы можем узнать, нажата ли кнопка.

На следующем рисунке напряжение на выводе GPIO равно GND, когда мы нажимаем кнопку.

Однако каково напряжение при отпускании кнопки?

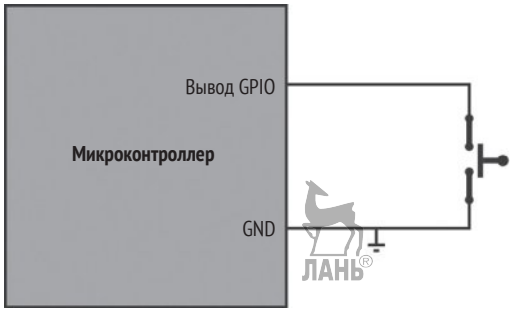


Рис. 2.23 ❖ Каково напряжение на выводе GPIO, когда кнопка отпущена?

Хотя сигнал на входе может принимать только два логических уровня, это может быть неверно в некоторых обстоятельствах. Вывод в режиме входа находится в плавающем (или высокоимпедансном) состоянии, иногда называемом третьим состоянием[33]. В этом состоянии логический уровень вывода не определен, поскольку напряжение произвольно скачет между 3.3 В и GND, и мы не можем знать, нажата ли кнопка или отпущена. Чтобы предотвратить эту проблему, мы должны включить в нашу схему резистор, чтобы всегда иметь определенный логический уровень при любых условиях.

В зависимости от того, какой логический уровень мы хотим получить при нажатии, резистор может быть подключен двумя способами:

- **подтягивающий** (**pull-up**) резистор соединяет вывод GPIO с напряжением 3.3 В. В таком подключении вывод GPIO считывает низкий уровень (LOW, 0) в нажатом состоянии кнопки и высокий (HIGH, 1) в отпущенном;
- **заземляющий** (**pull-down**) резистор соединяет вывод GPIO с GND. Таким образом, вывод GPIO считывает высокий уровень (HIGH, 1) в нажатом состоянии и низкий (LOW, 0) в отпущенном.

На следующем рисунке показана разница между конфигурациями pull-up и pull-down:

[33] Выводы логической схемы, способные находиться в трех состояниях, – ноль, единица или отключение выхода (высокоимпедансное состояние, Z-состояние), – называют трехстабильными (*tri-stated*).

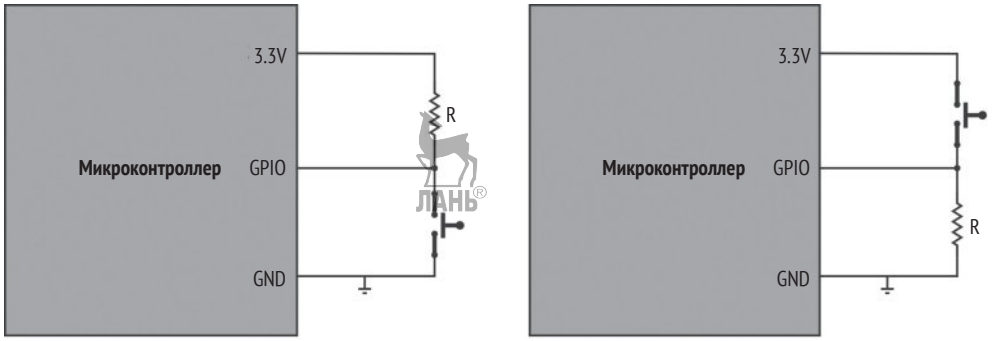


Рис. 2.24 ❖ Конфигурации с подтягивающим и заземляющим резисторами

Как правило, резистор 10 кОм должен подходить для обоих случаев. Однако большинство микроконтроллеров имеет внутренний программируемый подтягивающий резистор, поэтому внешний обычно не требуется[34].

Как это делается...

Сохраните все установленные компоненты на макетной плате из предыдущего примера управления светодиодом с помощью GPIO. Следующие шаги показывают, что нужно изменить в коде, чтобы управлять состоянием светодиода с помощью кнопки.

1. Выберите вывод GPIO для считывания состояния кнопки. В следующей таблице представлен наш выбор:

Таблица 2.5. Вывод GPIO для считывания состояния кнопки

Плата	**Входной вывод GPIO**
Arduino Nano	P0.30
Raspberry Pi Pico	GP10

2. Установите кнопку посередине платы между левыми и правыми рядами контактов:

[34] Тем не менее при необходимости получить высоконадежную схему, устойчивую к помехам и перепадам напряжения питания, внешние подтягивающие и заземляющие резисторы предпочтительнее встроенных. Чем ниже их сопротивление, тем выше защита от случайных срабатываний, но тем больше потребляемый ток при замыкании, поэтому оптимальным считается сопротивление в пределах 5–10 кОм. Сопротивление встроенных pull-up-резисторов может быть в разы больше: например, у Raspberry Pi Pico около 50 кОм, поэтому в документации на контроллер RP2040 имеется предупреждение о возможной недостаточности его сопротивления. В то же время для Arduino Nano 33 BLE (в отличие от обычных Arduino на контроллерах Atmel) сопротивление встроенных pull-up-резисторов снижено до ~13 кОм, и там этой проблемой в большинстве практических случаев можно не озадачиваться.

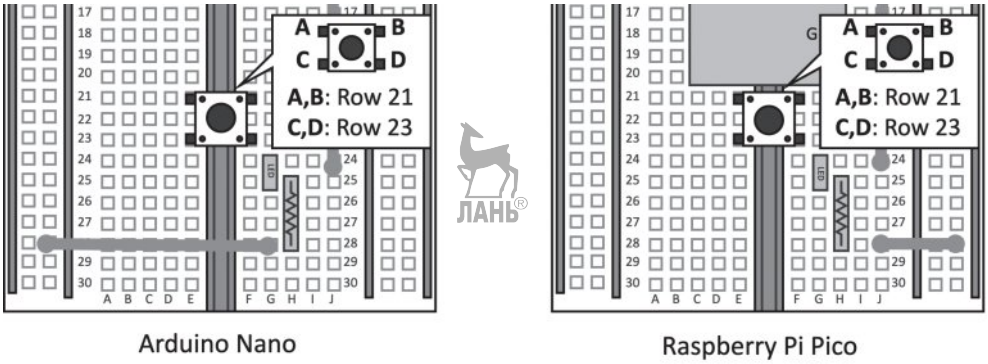


Рис. 2.25 ❖ Кнопка установлена между рядами контактов 21 и 23

Как мы можем видеть из рис. 2.25, используются контактные ряды, не занятые другими компонентами.

3. Подключите кнопку к выбранному контакту GPIO и GND:

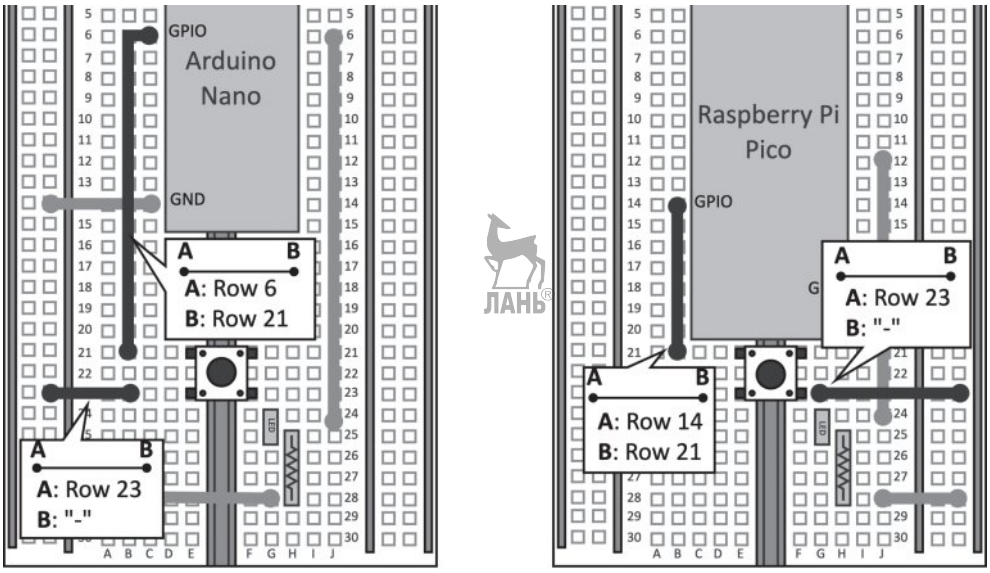


Рис. 2.26 ❖ Кнопка подключена только к контактам GPIO и GND

Неопределенного состояния входа не возникнет, потому что мы будем использовать подтягивающий резистор микроконтроллера.

4. Откройте скетч, разработанный в предыдущем примере. Объявите и инициализируйте глобальный объект `mbed::DigitalIn` с именем вывода, используемого для кнопки.
   Для Arduino Nano:

```
mbed::DigitalIn button(p30);
```

То же самое для Raspberry Pi Pico:

```
mbed::DigitalIn button(p10);
```

Объект `mbed::DigitalIn` (https://os.mbed.com/docs/mbed-os/v6.15/apis/digitalin.html) используется для взаимодействия с периферийным устройством GPIO в режиме ввода. Для инициализации требуется только имя вывода GPIO (`PinName`), подключенного к кнопке.

5. Установите режим кнопки с подключением подтягивающего резистора в функции `setup()`:

```
void setup() {
  button.mode(PullUp);
}
```

Показанный код подключает внутренний подтягивающий резистор микроконтроллера.

6. В функции `loop()` светодиод включается, когда на входе кнопки низкий уровень (LOW, 0):

```
void loop() {
  led = !button;
}
```

Здесь для вывода управления светодиодом просто устанавливается значение, противоположное значению, возвращаемому кнопкой, чтобы светодиод загорался при нажатии кнопки и гас при отпускании.

Скомпилируйте скетч и загрузите программу в микроконтроллер.

При нажатии кнопки замыкающие контакты могут генерировать ложные переходы логического уровня из-за механической природы компонента. Эта проблема называется дребезгом кнопки, потому что выход многократно перескакивает с высокого на низкий уровень в течение короткого времени. Вы можете рассмотреть возможность использования алгоритма «антидребезга» (например, https://os.mbed.com/teams/TVZ-Mechatronics-Team/wiki/Timers-interrupts-andtasks), чтобы предотвратить генерацию нескольких импульсов[35].

[35] Явление дребезга контактов в описанном случае простого включения светодиода никак не сказывается на работе схемы: глаз просто не успевает заметить мигания за время дребезга, составляющее обычно несколько миллисекунд максимум (аналогично тому, как не сказывается дребезг контактов сетевого выключателя при включении бытовых электроприборов). Однако в схемах, реагирующих не на уровень, а на каждое его изменение (перепад уровней – см. следующий пример), а также в схемах, где учитывается количество нажатий, дребезг абсолютно недопустим. Например, для счетчиков нажатий или при работе с клавиатурами ввода символов микроконтроллер не в состоянии отличить дребезг от действительного нажатия, и приходится принимать специальные «антидребезговые» меры, чтобы не допустить множественных срабатываний. К распространенным случаям такого рода относится часто практикуемое отключение и включение устройства одной и той же кнопкой: если не принимать мер борьбы с дребезгом, устройство при каждом нажатии может равновероятно оказаться в любом из двух состояний «вкл» или «выкл».

Использование прерываний для считывания состояния кнопки

В предыдущем примере рассказывалось, как считывать цифровые сигналы с помощью периферийного устройства GPIO. Однако предлагаемое решение неэффективно, поскольку процессор тратит впустую циклы ожидания нажатия кнопки, в то время как он мог бы сделать что-то еще за это время. Кроме того, может быть сценарий, в котором мы будем держать процессор в режиме низкого энергопотребления, и тогда он вообще не отреагирует на нажатие кнопки.

Этот пример научит нас, как эффективно считывать состояние кнопки с помощью прерываний на Arduino Nano.

Код этого примера содержится в файле *05_gpio_interrupt.ino*, доступном по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter02/ArduinoSketches/05_gpio_interrupt.ino.

Подготовка

Для воспроизведения этого примера сначала необходимо узнать, что такое прерывание и какой API Mbed OS можно использовать для считывания нажатия кнопки.

Прерывание – это сигнал, который временно приостанавливает основную программу для ответа на событие с помощью специальной функции, называемой обработчиком прерываний или процедурой обслуживания прерываний (**interrupt service routine**, **ISR**). Как только ISR завершает выполнение, процессор возобновляет основную программу с команды, на которой она была прервана, как показано на следующей блок-схеме:

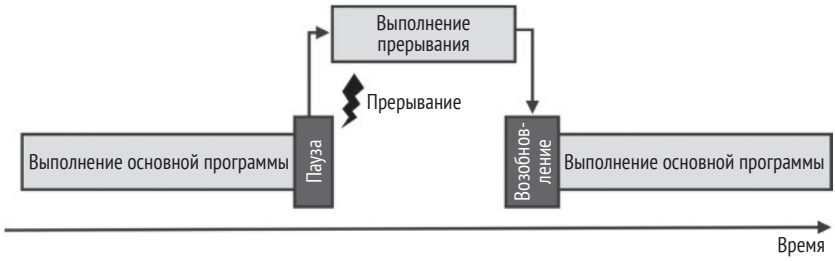


Рис. 2.27 ❖ Прерывание временно приостанавливает основную программу

Прерывание – мощный механизм экономии энергии, поскольку центральный процессор может перейти в спящий режим и дождаться события перед началом вычислений.

Микроконтроллер имеет несколько источников прерываний, и для каждого из них мы можем запрограммировать выделенную ISR.

Хотя ISR является функцией, существуют ограничения для ее реализации.

- У нее нет входных аргументов.
- Она не возвращает значение. Поэтому нам необходимо использовать глобальные переменные для сообщения об изменениях статуса.
- Она должна быть короткой, чтобы не отнимать слишком много времени у основной программы[36]. Мы хотим напомнить вам, что ISR не является отдельным потоком, поскольку процессор может возобновить вычисления только после завершения ISR.

Для периферийного устройства GPIO в режиме ввода мы можем использовать объект `mbed::InterruptIn` (https://os.mbed.com/docs/mbed-os/v6.15/apis/interruptin.html) для запуска события всякий раз, когда изменяется логический уровень на выводе:

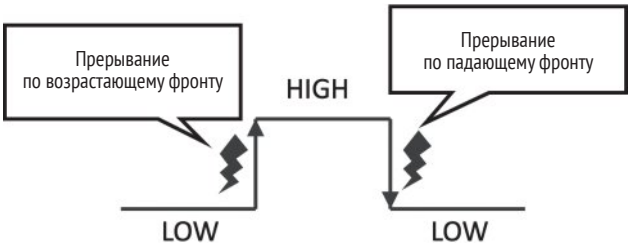


Рис. 2.28 ❖ Прерывание по возрастающему (*rising*) и по падающему (*falling*) фронтам

Как мы можем видеть из рисунка выше, `mbed::InterruptIn` может вызывать прерывания, когда логический уровень на выводе переходит от низкого к высокому уровню (*rising interrupts*, прерывания по возрастающему фронту) или от высокого к низкому (*falling interrupt*, прерывания по падающему фронту).

Как это делается...

Чтобы включить и выключить светодиод с помощью прерывания, откройте скетч, созданный в предыдущем примере, и выполните следующие действия.

1. Определите и инициализируйте объект `mbed::InterruptIn` с помощью имени вывода GPIO (`PinName`), подключенного к кнопке.
   Для Arduino Nano:

   ```
   mbed::InterruptIn button(p30);
   ```

   Для Raspberry Pi Pico:

   ```
   mbed::InterruptIn button(p10);
   ```

[36] Необходимо отметить, что для микроконтроллеров нередко применяется построение программы, в котором главный вычислительный поток представляет собой пустой или почти пустой цикл, а вся функциональность сосредоточена именно в прерываниях. К таким случаям относится в том числе и излагаемый здесь пример (см. далее).

Объект `mbed::DigitalIn` больше не требуется, поскольку `mbed::InterruptIn` также управляет интерфейсом с периферийным устройством GPIO в режиме ввода.

2. Создайте ISR-процедуру для обработки запроса на прерывание по возрастающему фронту входного сигнала (от низкого к высокому уровню):

```
void rise_ISR() {
  led = 0;
}
```

При вызове этой ISR светодиод выключается (`led = 0`).
Затем создайте ISR для обработки запроса на прерывание по падающему фронту (от высокого к низкому уровню) входного сигнала:


```
void fall_ISR() {
  led = 1;
}
```

При вызове этой ISR светодиод включается (`led = 1`).

3. Инициализируйте `button` в функции `setup()`:

```
void setup() {
  button.mode(PullUp);
  button.rise(&rise_ISR);
  button.fall(&fall_ISR);
}
```

Здесь мы настраиваем объект `mbed::InterruptIn` следующими действиями.

- Включение внутреннего подтягивающего резистора (`button.mode(PullUp)`).
- Подключение функции ISR для вызова при возникновении прерывания по возрастающему фронту (`button.rise(&rise_ISR)`).
- Подключение функции ISR для вызова при возникновении прерывания по падающему фронту (`button.fall(&fall_ISR)`).

4. Замените код в функции `loop()` на `delay(4000)`:


```
void loop() {
   delay(4000);
}
```

Теоретически можно было бы оставить функцию `loop()` пустой. Однако мы рекомендуем вызывать `delay()`, когда ничего не нужно делать, поскольку это может снизить энергопотребление в системе[37].

Скомпилируйте скетч и загрузите программу в микроконтроллер.

[37] Подчеркнем, что это не перевод в режим энергосбережения (для чего нужна специальная команда), а простая экономия за счет того, что функция `delay()` приостанавливает выполнение любых операций в главном цикле программы (в том числе и непрерывных переходов на начало цикла, как это было бы при ее отсутствии).

Питание микроконтроллеров от батарей

Для многих приложений TinyML батареи могут быть единственным источником питания. В последнем для этой главы примере мы узнаем, как питать микроконтроллеры от батареек типа AA.

Colab notebook[38] под названием *06_estimate_battery_life.ipynb* содержит код этого примера: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter02/ColabNotebooks/06_estimate_battery_life.ipynb.

Подготовка

Микроконтроллеры не имеют встроенной батареи, поэтому нам необходимо подключить внешнюю батарею, чтобы устройство работало, когда оно не подключено через кабель micro-USB.

Чтобы подготовиться к этому примеру, нам нужно знать, какие типы батарей нам нужны и как их правильно использовать для подачи питания.

Батареи являются источниками электроэнергии и имеют ограниченную энергетическую емкость[39]. Энергоемкость определяет количество накопленной энергии и для гальванических элементов измеряется в миллиампер-часах (мАч). Таким образом, более высокое значение мАч означает более длительное время автономной работы.

В следующей таблице приведены некоторые коммерческие батареи, которые можно использовать с микроконтроллерами:

Таблица 2.6. Коммерческие батареи, подходящие для микроконтроллерных устройств

Тип элемента	Номинальное напряжение, В	Энергетическая емкость, мАч
AAA	1.5	~1000
AA	1.5	~2400 (щелочные)
CR2032	3.6	~240
CR2016	3.6	~90

Выбор батареи зависит от требуемого напряжения микроконтроллера, а также от других факторов – необходимой энергоемкости, допустимых габаритов и рабочей температуры.

---

[38] Colab notebook – отдельный проект в сервисе Google Colab, см. описание на стр. 48.

[39] Следует подчеркнуть, что в данном случае речь идет о собственно батареях – одноразовых гальванических элементах (соединение которых в группы и называют батареями), а не об аккумуляторах, которые в англоязычной литературе также часто называют батареями. Для малопотребляющих или редко включающихся устройств (телевизионные пульты, переносные электроизмерительные приборы-тестеры, компьютерные мыши, автономные устройства IoT) аккумуляторы употреблять нецелесообразно, так как саморазряд у них оказывается гораздо больше среднего потребления, что требует частой дозарядки, в то время как одноразовые батарейки могут служить годами.

Как мы можем видеть из таблицы, батарея типа AA обеспечивает более высокую емкость, но она подает 1.5 В, что обычно недостаточно для микроконтроллеров. Как мы можем тогда питать микроконтроллеры от батареек типа AA?

В следующих подразделах мы покажем стандартные методы увеличения подаваемого напряжения или энергетической емкости.

Увеличение выходного напряжения при последовательном подключении батарей

При последовательном подключении батарей положительный контакт одного элемента соединяется с отрицательным контактом другого, как показано на следующем рисунке:

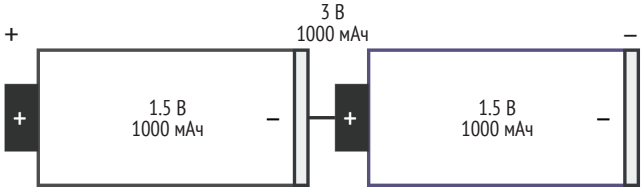


Рис. 2.29 ❖ Батареи при последовательном включении

Последовательное соединение приведет не к увеличению емкости батареи, а только к увеличению подаваемого напряжения.

Новое выходное напряжение (V_{new}) будет составлять:

$$V_{new} = V_{battery} \cdot N,$$

где N – количество последовательно подключенных батарей.

Например, поскольку одна батарея типа AA обеспечивает питание 1.5 В при емкости 2400 мАч, мы могли бы подключить последовательно две батарейки типа AA, чтобы они вырабатывали 3.0 В при той же энергоемкости.

Однако, если емкости батареи недостаточно для нашего приложения, как мы можем ее увеличить?

Увеличение энергетической емкости за счет параллельного подключения батарей

При параллельном подключении батарей все положительные выводы элементов соединяются между собой, то же самое относится и к отрицательным выводам, как показано на рис. 2.30.

Параллельное включение приведет не к увеличению выходного напряжения, а только к увеличению емкости аккумулятора.

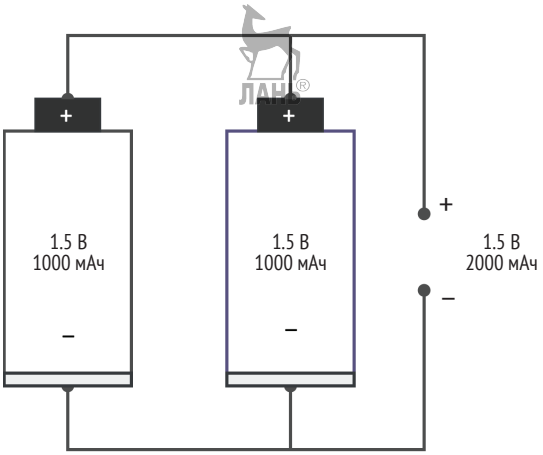


Рис. 2.30 ❖ Батареи, подключенные параллельно

Новая емкость батареи (BC_{new}) составит:

$$BC_{new} = BC_{battery} \cdot N,$$

где N – количество параллельно подключенных батарей.

Например, поскольку емкость одной батареи типа AA составляет 2400 мАч, мы могли бы подключить две батареи типа AA параллельно, чтобы увеличить емкость батареи в два раза[40].

Теперь, когда мы знаем, как соединить несколько батарей вместе, чтобы получить желаемое выходное напряжение и энергетическую емкость, давайте посмотрим, как мы можем использовать их для питания микроконтроллеров.

Подключение батарей к плате микроконтроллера

Микроконтроллеры имеют специальные контакты для подачи питания через внешние источники энергии, такие как батареи. Эти контакты имеют ограничения по напряжению, обычно указанные в техническом описании.

На Arduino Nano внешний источник питания подается через вывод VIN. Входное напряжение VIN может варьироваться от 5 до 21 В.

На Raspberry Pi Pico внешний источник питания подается через вывод VSYS. Входное напряжение VSYS может варьироваться от 1.8 до 5.5 В. На обеих платформах встроенный регулятор напряжения преобразует подаваемое напряжение в 3.3 В.

[40] В реальности для одноразовых гальванических элементов параллельное включение практикуют крайне редко. Такое включение подразумевает, что батарейки должны быть из одной партии, лучше совершенно новыми и хранившимися в одинаковым условиях. Малейший перекос в напряжениях между элементами приведет к тому, что элемент с большим напряжением начнет разряжаться через элемент с меньшим. При большой разнице это может даже привести к их повреждению, но в любом случае снизит емкость ниже расчетной и уменьшит срок службы. В отличие от одноразовых элементов аккумуляторы включают параллельно довольно часто, так как они при небольшом разбросе напряжений не повреждаются от взаимной перезарядки, но и в этом случае следует стараться подбирать элементы из одной партии.

Как это делается...

Отсоедините Arduino Nano и Raspberry Pi Pico от micro-USB, сохранив все компоненты на макетной плате.

Батарейный отсек, рассматриваемый для этого примера, соединяет батарейки типа AA последовательно. Мы не рекомендуем пока вставлять батарейки в батарейный отсек. Батарейки следует вставлять только после завершения формирования электрической цепи.

Следующие шаги покажут, как питать Arduino Nano и Raspberry Pi Pico от батарей.

1. Подсоедините положительный (красный) и отрицательный (черный) провода батарейных отсеков к шинам макетной платы + и – соответственно:

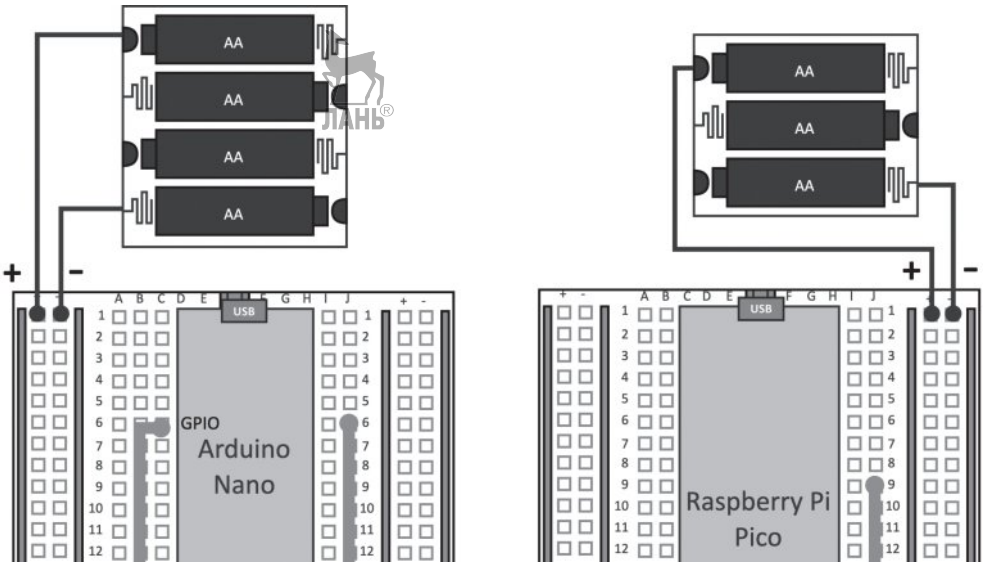


Рис. 2.31 ❖ Подсоедините батарейный отсек к шинам питания макетной платы

2. Arduino Nano и Raspberry Pi Pico имеют разные пределы напряжения для внешнего источника питания. Поэтому мы не можем использовать одинаковое количество батареек типа AA на обеих платформах. На самом деле для Raspberry Pi Pico достаточно трех батареек типа AA, но этого не хватит для Arduino Nano. Для Arduino Nano достаточно четырех батареек типа AA, но при этом для Raspberry Pi Pico напряжение выходит за пределы допустимого[41]. По этой причине мы используем

[41] Разница в том, что в Arduino Nano установлен простой аналоговый стабилизатор MPM3610A, требующий напряжения на входе 4.5 В минимум, а в Raspberry Pi Pico – преобразователь напряжения, выдающий 3.3 В даже при входном 1.8 В, но ограниченный по максимальному напряжению обычным для современных микросхем значением 5.5 В. Обратите внимание, что Raspberry Pi Pico при этом оказывается

батарейный отсек на 4 элемента AA для Arduino Nano, получая питание около 6 В, и батарейный отсек на 3 элемента AA для Raspberry Pi Pico с питанием 4.5 В.

3. Подключите внешний источник питания к плате микроконтроллера, как показано на следующей схеме:

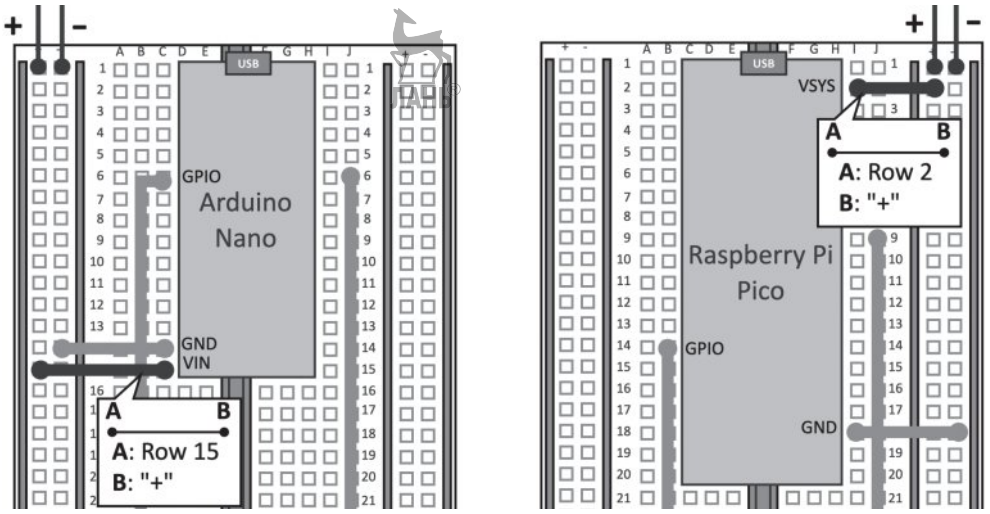


Рис. 2.32 ❖ Подключение контактов питания микроконтроллеров

Как вы можете видеть из рисунка, VIN (Arduino Nano) и VSYS (Raspberry Pi Pico) подключены к положительному выводу батарейного отсека через шину +.

4. Вставьте батарейки в батарейный отсек:
   - 4 батарейки типа AA для Arduino Nano;
   - 3 батарейки типа AA для Raspberry Pi Pico.

Теперь приложение управления светодиодом должно снова работать.

Однако одна вещь, которая может нас заинтересовать, – это как мы можем оценить срок службы приложения, работающего от батареи?

Дополнительно

После того как мы выбрали батарею для микроконтроллера, мы можем оценить ее срок службы по следующей формуле:

$BL = BC/IL$,

выгоднее не только из-за меньшего количества требуемых элементов, но и из-за более полного использования их емкости: щелочной элемент в конце разряда выдает около 0.9 В, что для трех элементов все еще намного превышает необходимый минимум в 1.8 В. А необходимые для Arduino Nano 4.5 В достигаются при разряде примерно до 1.1 В на каждый из четырех элементов, что соответствует при малом потреблении неистраченному остатку около четверти первоначальной емкости.

где:

Таблица 2.7. Физические величины в формуле оценки срока службы батареи

Величина	Единица измерения	Физический смысл
BL	Часы	Время жизни батарейки
BC	мАч	Емкость батарейки
IL	мА	Потребление тока платой микроконтроллера

Следующий код на Python вычисляет время автономной работы в часах и днях:

```
battery_cap_mah = 2400
i_load_ma = 1.5

battery_life_hours = battery_cap_mah / i_load_ma
battery_life_days = battery_life_hours / 24

print("Battery life:", battery_life_hours,"hours,", battery_life_days, "days")
```

Приведенный код оценивает время автономной работы для случая, когда емкость батареи (`battery_cap_mah`) составляет 2400 мАч, а ток нагрузки (`i_load_ma`) равен 1.5 мА.

Ожидаемый результат работы программы следующий:

```
Battery life: 1600.0 hours, 66.66666666666667 days
```

Рис. 2.33 ❖ Ожидаемый результат оценки времени автономной работы

Хотя приведенная выше формула является приблизительной и действительна в идеальных условиях, ее достаточно, чтобы понять, как долго может прослужить система. Лучшая модель могла бы включать другие факторы, такие как саморазряд батареи и температура[42].

[42] В задаче оценки времени работы от батареи для любого электронного устройства самую большую проблему представляет вычисление или измерение среднего потребляемого тока. Из-за того, что большинство приборов имеет периодический характер работы (например, радиодатчик обычно работает по циклу: измерение–передача–пауза), импульсное мгновенное потребление может в десятки раз превысить то, что показывает амперметр, успевающий отреагировать лишь на усредненное потребление в паузах. Опыт показывает, что даже внимательный анализ кривой потребления с помощью осциллографа и последующим расчетом среднего значения не дает достаточно точного результата. Наилучший способ – применение специальных анализаторов профиля потребления (Power Profiler), которые, однако, могут быть достаточно дороги: так, один из самых популярных Nordic Semiconductor Power Profiler Kit продается в зарубежных магазинах за цену порядка 100 долл., которая в российских условиях кратно возрастает.

Глава 3

Создание метеостанции с помощью библиотеки TensorFlow Lite for microcontrollers

В настоящее время благодаря подключению к интернету легко получить прогноз погоды с помощью смартфонов, ноутбуков и планшетов. Однако задумывались ли вы когда-нибудь о том, что бы вы сделали, если бы вам пришлось отслеживать погоду в отдаленном регионе без доступа в интернет?

Эта глава научит нас, как реализовать метеостанцию с помощью машинного обучения (ML) с использованием данных о температуре и влажности за последние три часа.

В этой главе мы сосредоточимся на подготовке набора данных и покажем, как получать исторические данные о погоде из WorldWeatherOnline. После этого объясним, как тренировать и тестировать модель TensorFlow (TF), а в последней части развернем модель на Arduino Nano и Raspberry Pi Pico с библиотекой TensorFlow Lite for microcontrollers (TFLu) и создадим приложение для прогнозирования того, выпадет ли снег.

Цель этой главы – провести вас через все этапы разработки приложения на базе TFLu и объяснить, как получать данные датчиков температуры и влажности.

В этой главе мы собираемся реализовать следующие примеры.

- Импорт данных о погоде из WorldWeatherOnline.
- Подготовка набора данных.
- Обучение модели с помощью TF.
- Оценка эффективности модели.
- Квантизация модели с помощью конвертера TFLite.
- Использование встроенного датчика температуры и влажности на Arduino Nano.
- Использование датчика DHT22 с Raspberry Pi Pico.
- Подготовка входных характеристик для просчета модели.
- Запуск на устройстве с помощью TFLu.

Чтобы выполнить все практические примеры этой главы, нам понадобится следующее:

- плата Arduino Nano 33 Sense;
- плата Raspberry Pi Pico;
- кабель micro-USB;
- беспаечная макетная плата половинного размера (только для Raspberry Pi Pico);
- модуль AM2302 с датчиком DHT22[43] (только для Raspberry Pi Pico);
- 5 соединительных проводов-перемычек штырь–штырь (только для Raspberry Pi Pico);
- ноутбук/ПК с Ubuntu 18.04+ или Windows 10 на x86-64.

Исходный код и дополнительные материалы доступны в папке *Chapter03* репозитория для этой книги по адресу https://github.com/PacktPublishing/TinyML-Cookbook/tree/main/Chapter03.

Импорт данных о погоде из WorldWeatherOnline

Эффективность алгоритмов ML в значительной степени зависит от данных, используемых для обучения. Следовательно, как мы обычно говорим, модель ML хороша настолько, насколько хорош набор данных. Основным требованием к хорошему набору данных является то, что входные данные должны достаточно полно представлять проблему, которую мы хотим решить. Так, мы знаем из физики, что температура и влажность влияют на образование снега.

Поэтому в данном примере мы покажем, как собирать исторические почасовые данные о температуре, влажности и снегопадах, чтобы создать набор данных для прогнозирования снега.

Файл Colab *preparing_model.ipynb* (в разделе *Importing weather data from WorldWeatherOnline*) содержит код, описанный в этом примере: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter03/ColabNotebooks/preparing_model.ipynb.

Подготовка

В интернете есть различные источники, из которых мы можем собирать почасовые данные о погоде, но большинство из них не являются бесплатными или имеют ограничения на использование.

Для этого примера нашим выбором стал сервис **WorldWeatherOnline** (https://www.worldweatheronline.com/developer/), имеющий бесплатный пробный период в течение 30 дней и предоставляющий следующее:

[43] В отличие от многочисленных датчиков температуры-влажности с I2C-интерфейсом датчик AM2302/DHT22 снабжен нестандартным однопроводным интерфейсом (см. далее), потому почти не имеет доступных аналогов для замены.

- простой API через HTTP-запросы для получения данных;
- исторические данные о погоде по всему миру;
- 250 запросов данных о погоде в день.

Ограничение на количество запросов данных о погоде в день никак не влияет на этот пример.

Вам нужно только зарегистрироваться на веб-сайте, чтобы начать получать данные.

WorldWeatherOnline имеет API под названием *Past Historical Weather API* (https://www.worldweatheronline.com/developer/premium-api-explorer.aspx), что позволяет нам собрать историю погодных условий с 1 июля 2008 года.

Однако мы не будем напрямую работать с его собственным API, а будем использовать пакет Python `wwo-hist` (https://github.com/ekapope/WorldWeatherOnline) для экспорта данных непосредственно в pandas DataFrame[44].

Как это делается...

Откройте Colab и создайте новый проект (notebook). В области кода выполните следующие действия.

1. Установите пакет `wwo-hist`:

```
!pip install wwo-hist
```

2. Импортируйте из `wwo-hist` функцию `retrieve_hist_data`:

```
from wwo_hist import retrieve_hist_data
```

   Функция `retrieve_hist_data` – единственная, необходимая для получения данных из WorldWeatherOnline, и может экспортироваться либо в pandas DataFrames, либо в CSV-файлы.

3. Получите данные за десять лет (с 01 января 2011 года по 31 декабря 2020 года) с почасовой периодичностью для Канацеи[45]:


```
frequency=1
api_key = 'YOUR_API_KEY'
location_list = [canazei]
df_weather = retrieve_hist_data(api_key, location_list, '01-JAN-2011',
                                '31-DEC-2020', frequency, location_label = False,
                                export_csv = False, store_df = True)
```

   Пакет `www-hist` экспортирует данные в `df_weather`, список pandas DataFrames.

---

[44] Pandas – библиотека Python с открытым исходным кодом для анализа больших массивов данных. Pandas DataFrame – наиболее широко используемая структура данных в Python pandas.

[45] Канацеи (Canazei) – местечко в Италии, часть горнолыжного курорта Валь-ди-Фасса в провинции Трентино (см. также далее).

На этом шаге мы задаем входные аргументы для функции `retrieve_hist_data`. Давайте рассмотрим их подробней.

- **API key**: ключ API берется на панели управления подпиской WorldWeatherOnline, замените им строку `YOUR_API_KEY`.
- **Location**: это список местоположений, для которых можно получить данные о погоде. Поскольку мы создаем набор данных для прогнозирования снегопада, нам следует рассмотреть места, где периодически идет снег. Поэтому в примере взят Канацеи (https://en.wikipedia.org/wiki/Canazei), расположенный на севере Италии, где снегопад может произойти в любой момент в период с декабря по март. Мы могли бы также добавить другие местоположения, чтобы сделать модель ML более универсальной.
- **Start date/End date**: даты начала и окончания определяют временной интервал, в течение которого необходимо собирать данные. Формат даты – *dd-mmm-yyyy*. Поскольку нам нужен большой репрезентативный набор данных, мы запрашиваем данные о погоде за 10 лет. Таким образом, временной интервал установлен на `'01-JAN-2011' - ' 31-DEC-2020'`.
- **Frequency**: определяет частоту выборок в час. Например, 1 означает каждый час, 3 – каждые три часа, 6 – каждые шесть часов и т. д. Мы выбираем периодичность в 1 ч, поскольку для прогноза снега нам нужны данные о температуре и влажности за последние три часа.
- **Location label**: поскольку нам может потребоваться получить данные из разных местоположений, эта метка привязывает полученные данные о погоде к месту. Мы устанавливаем этот параметр на `False`, потому что используем только одно местоположение.
- **export_csv**: это флаг для экспорта данных о погоде в CSV-файл. Мы установили его на `False`, потому что нам не нужно экспортировать данные в CSV-файл.
- **store_df**: это флаг для экспорта данных о погоде в pandas DataFrame. Для него устанавливаем значение `True`.

Как только данные о погоде будут получены, на выходе консоли появится сообщение «*export to canazei completed!*».

4. Экспортируйте температуру, влажность и количество выпадающего снега в отдельные списки:

```
t_list = df_weather[0].tempC.astype(float).to_list()
h_list = df_weather[0].humidity.astype(float).to_list()
s_list = df_weather[0].totalSnow_cm.astype(float).to_list()
```

Сгенерированный набор данных `df_weather[]` включает в себя несколько погодных параметров для каждой запрошенной даты и времени. Например, мы можем найти давление в миллибарах, облачность в процентах, видимость в километрах и, конечно же, физические величины, которые нас интересуют:

- `tempC`: температура в градусах Цельсия (°C);
- `humidity`: относительная влажность воздуха в процентах (%);

- `totalSnow_cm`: общее количество выпавшего снега в сантиметрах (см).

На этом заключительном шаге мы экспортируем почасовую температуру, влажность и снегопад в сантиметрах в три списка, используя метод `to_list()`.

Теперь у нас есть все, что нам нужно, чтобы подготовить набор данных для прогнозирования снега.

Подготовка набора данных

Подготовка набора данных является решающим этапом в любом проекте ML, поскольку это влияет на эффективность обученной модели.

В этом примере мы применим два метода, чтобы сделать набор данных более подходящим для получения более достоверной модели. Эти два метода приведут набор данных в соответствие со стандартами и приведут входные характеристики в один и тот же числовой диапазон.

Код, описанный в этом примере, содержится в разделе *Preparing the dataset* файла Colab *preparing_model.ipynb*: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter03/ColabNotebooks/preparing_model.ipynb.

Подготовка

Наши входные данные – температура и влажность за последние три часа. Мы используем погодные условия именно за последние три часа просто для того, чтобы увеличить количество входных данных и повысить достоверность (*accuracy*)[46] их классификации.

Чтобы подготовиться к подготовке набора данных, нам нужно знать, почему набор данных должен быть сбалансирован и почему необработанные входные функции не следует использовать для обучения. Эти два аспекта будут рассмотрены в следующих подразделах.

Подготовка сбалансированного набора данных

Несбалансированный набор данных – это набор данных, в котором один из классов параметров содержит значительно больше выборок, чем другие. Обучение с несбалансированным набором данных могло бы создать модель с высокой достоверностью, но неспособную решить нашу проблему. Например, рассмотрим набор данных, в котором один из двух классов содержит 99 % выборок. Если бы сеть неправильно классифицировала меньший класс, у нас все равно была бы достоверность 99 %, но модель была бы неэффективной.

[46] Термин *accuracy*, означающий общую долю правильных ответов алгоритма, здесь и далее переводится как «достоверность» в целях избежания путаницы с термином *precision*, о котором см. далее в разделе «*Оценка эффективности модели*».

Поэтому нам требуется сбалансированный набор данных примерно с одинаковыми входными выборками для каждой категории.

Балансировка набора данных может быть выполнена с помощью следующих методов.

- **Получение большего количества выборок данных** для меньшего класса, где их недостаточно: это должно быть первым, что мы должны сделать, чтобы убедиться, что мы правильно сгенерировали набор данных. Однако не всегда возможно собрать больше данных, особенно когда речь идет о нечастых событиях.
- **Увеличение выборки для меньшего класса**: мы могли бы случайным образом дублировать выборки для класса, где их недостаточно. Однако такой подход может увеличить риск переобучения для этого класса, если дублировать слишком много примеров.
- **Уменьшать выборку для большего класса**: мы могли бы случайным образом удалять выборки из класса, где их слишком много. Поскольку такой подход уменьшает размер набора данных, мы можем потерять ценную обучающую информацию.
- **Создание синтетических данных для меньшего класса**: мы могли бы разработать искусственное приготовление образцов данных. Наиболее распространенным алгоритмом для этого является **Synthetic Minority Over-sampling Technique** (**SMOTE**). SMOTE – метод повышения объема выборки, который создает новые данные вместо дублирования имеющихся экземпляров. Хотя этот метод снижает риск переобучения, сгенерированные синтетические данные могут быть неверными вблизи границы разделения классов, добавляя в набор данных нежелательный шум.

Как мы можем видеть, несмотря на разнообразие методов, в целом не существует наилучшего решения для исправления несбалансированного набора данных. Метод или методы, которые следует принять, будут зависеть от решаемой проблемы.

Масштабирование параметров с помощью Z-score

Наши входные характеристики существуют в разных числовых диапазонах. Например, влажность всегда находится в диапазоне от 0 до 100, в то время как температура по шкале Цельсия может быть отрицательной и имеет меньший числовой диапазон в абсолютных величинах, чем влажность.

Это типичный сценарий, когда имеешь дело с различными физическими величинами, и он может повлиять на эффективность обучения.

Как правило, если входные параметры имеют разные числовые диапазоны, модель ML может не обобщаться должным образом, поскольку на нее в большей степени будут влиять объекты с бóльшими величинами. Следовательно, входные параметры необходимо масштабировать, чтобы гарантировать, что каждый параметр вносит одинаковый вклад во время обучения. Кроме того, еще одним преимуществом масштабирования параметров в нейронных сетях является то, что оно помогает ускорять градиентный спуск к минимумам.

Z-score – распространенный метод масштабирования, принятый в нейронных сетях. Он определяется следующей формулой:

$$value_{new} = \frac{value_{old} - \mu}{\sigma}.$$

В этой формуле:

- μ: среднее значение входных характеристик,
- σ: стандартное отклонение входных характеристик.

Z-score может привести входные характеристики к одному и тому же числовому диапазону, но не обязательно между нулем и единицей.

Как это делается...

Чтобы сбалансировать набор данных и изменить масштаб входных объектов с помощью Z-score, для файла Colab выполните следующие действия.

1. Визуализируйте извлеченные физические измерения (температура, влажность и уровень снежного покрова) в 2D-графике. При этом учитывайте образование снега только тогда, когда количество выпавшего снега (из списка `s_list`) превышает 0.5 см:

```
def binarize(snow, threshold):
  if snow > threshold:
    return 1
  else:
    return 0
s_bin_list = [binarize(snow, 0.5) for snow in s_list]
cm = plt.cm.get_cmap('gray_r')
sc = plt.scatter(t_list, h_list, c=s_bin_list, cmap=cm, label="Snow")
plt.figure(dpi=150)
plt.colorbar(sc)
plt.legend()
plt.grid(True)
plt.title("Snow(T, H)")
plt.xlabel("Temperature - °C")
plt.ylabel("Humidity - %")
plt.show()
```

Этот код сгенерирует 2D-график (рис. 3.1).

На представленном графике ось *x* – температура, ось *y* – влажность, а черная точка – наличие снега. Как вы можете видеть из распределения черных точек, есть случаи, когда снег выпадает при температурах значительно выше 0 °C.

Чтобы упростить пример, мы можем проигнорировать эти случаи и считать 2 °C максимальной температурой для образования снега.

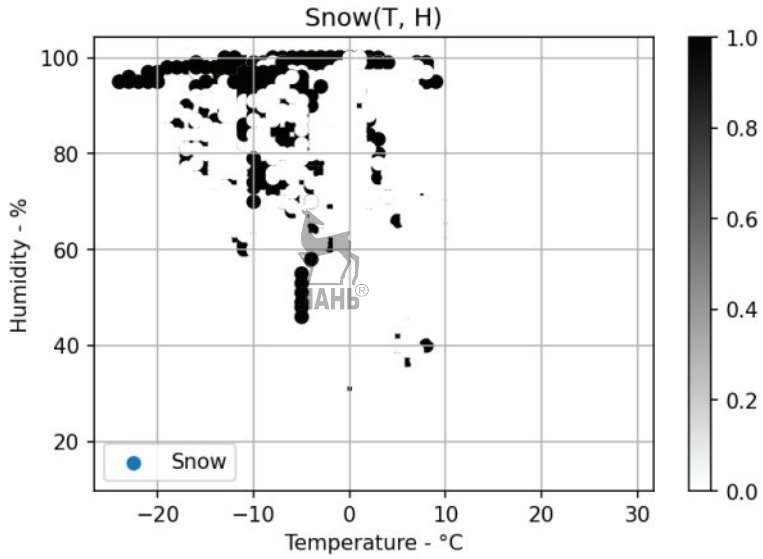


Рис. 3.1 ❖ Визуализация температуры, влажности и снега в 2D-графике. Данные предоставлены **WorldWeatherOnline.com**

2. Сгенерируйте сообщения для представления результата (`Yes` и `No`):

```
def gen_label(snow, temperature):
  if snow > 0.5 and temperature < 2:
    return "Yes"
  else:
    return "No"
snow_labels = [gen_label(snow, temp) for snow, temp in zip(s_list, t_list)]
```

Поскольку мы прогнозируем только снег, необходимы только два класса: `Yes` (да, идет снег) или `No` (нет, снега нет). В этой области мы преобразуем величину `totalSnow_cm` в соответствующий класс (`Yes` или `No`) с помощью функции `gen_label()`. Преобразующая функция присваивает значение `Yes`, когда значение `totalSnow_cm` превышает 0.5 см, а температура ниже 2 °C.

3. Создайте набор данных:

```
csv_header = ["Temp0", "Temp1", "Temp2", "Humi0", "Humi1", "Humi2", "Snow"]

df_dataset = pd.DataFrame(list(zip(t_list[:-2], t_list[1:-1], t_list[2:],
h_list[:-2], h_list[1:-1], h_list[2:], snow_labels[2:])), columns = csv_header)
```

Если *t0* – текущее время, то значения, сохраненные в наборе данных, следующие:

- `Temp0/Humi0`: температура и влажность в момент времени $t = t0 - 2$;
- `Temp1/Humi1`: температура и влажность в момент времени $t = t0 - 1$;
- `Temp2/Humi2`: температура и влажность в момент времени $t = t0$;
- `Snow`: сообщение, будет ли снег в момент $t = t0$.

Поэтому нам просто нужно имя (`zip`) и несколько индексных вычислений для построения набора данных.

4. Сбалансируйте набор данных, уменьшив выборку большего класса:

```
df0 = df_dataset[df_dataset['Snow'] == "No"]
df1 = df_dataset[df_dataset['Snow'] == "Yes"]

if len(df1.index) < len(df0.index):
  df0_sub = df0.sample(len(df1.index))
  df_dataset = pd.concat([df0_sub, df1])
else:
  df1_sub = df1.sample(len(df0.index))
  df_dataset = pd.concat([df1_sub, df0])
```

Исходный набор данных несбалансирован, поскольку в выбранном местоположении обычно выпадает снег в течение зимнего сезона, который длится только с декабря по март.
Следующая гистограмма показывает, что класс `No` представляет 87 % всех случаев, поэтому нам нужно применить один из методов, показанных в разделе «*Подготовка*», чтобы сбалансировать набор данных.

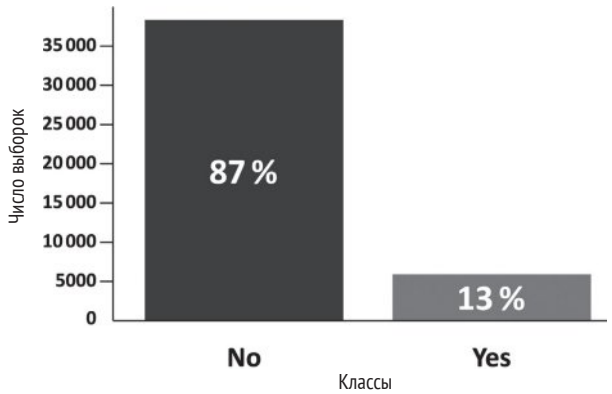


Рис. 3.2 ❖ Распределение выборок набора данных

Поскольку меньший класс имеет много выборок (~ 5000), мы можем случайным образом уменьшить количество выборок в большем классе, чтобы две категории имели одинаковое количество наблюдений.

5. Независимо масштабируйте входные объекты с помощью Z-score. Для этого извлеките все значения температуры и влажности:

```
t_list = df_dataset['Temp0'].tolist()
h_list = df_dataset['Humi0'].tolist()
t_list = t_list + df_dataset['Temp2'].tail(2).tolist()
h_list = h_list + df_dataset['Humi2'].tail(2).tolist()
```

Вы можете получить все значения температуры (или влажности) из столбца `Temp0` (или `Humi0`) и из последних двух записей столбца `Temp2` (или `Humi2`).

Затем вычислите среднее значение и стандартное отклонение входных параметров температуры и влажности:

```
t_avg = mean(t_list)
h_avg = mean(h_list)
t_std = std(t_list)
h_std = std(h_list)
print("COPY ME!")
print("Temperature - [MEAN, STD] ", round(t_avg, 5), round(t_std, 5))
print("Humidity - [MEAN, STD] ", round(h_avg, 5), round(h_std, 5))
```

Ожидаемый результат следующий:

```
COPY ME!
Temperature - [MEAN, STD]   2.05179 7.33084
Humidity - [MEAN, STD]      82.30551 14.55707
```

Рис. 3.3 ❖ Ожидаемые средние значения и стандартное отклонение

Скопируйте и сохраните значения среднего значения и стандартного отклонения, оказавшиеся в журнале вывода, поскольку они потребуются при развертывании приложения на Arduino Nano и Raspberry Pi Pico. Наконец, масштабируйте входные параметры с помощью Z-score:

```
def scaling(val, avg, std):
  return (val - avg) / (std)
df_dataset['Temp0']=df_dataset['Temp0'].apply(lambda x: scaling(x, t_avg, t_std))
df_dataset['Temp1']=df_dataset['Temp1'].apply(lambda x: scaling(x, t_avg, t_std))
df_dataset['Temp2']=df_dataset['Temp2'].apply(lambda x: scaling(x, t_avg, t_std))
df_dataset['Humi0']=df_dataset['Humi0'].apply(lambda x: scaling(x, h_avg, h_std))
df_dataset['Humi1']=df_dataset['Humi1'].apply(lambda x: scaling(x, h_avg, h_std))
df_dataset['Humi2']=df_dataset['Humi2'].apply(lambda x: scaling(x, h_avg, h_std))
```

На следующих графиках сравниваются распределения исходных (*Raw*) и масштабированных (*Scaled*) входных параметров:

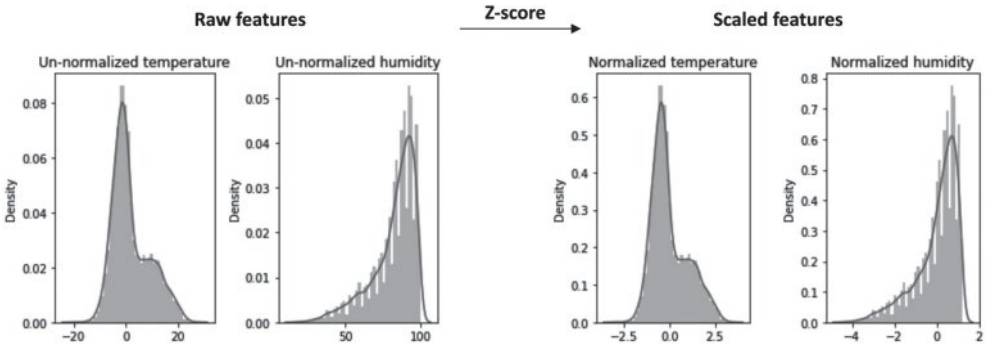


Рис. 3.4 ❖ Распределения исходных (слева) и масштабированных (справа) входных параметров

Как вы можете видеть из графиков, Z-score обеспечивает примерно одинаковый диапазон значений (ось *x*) для обоих параметров.

Теперь набор данных готов к использованию для обучения нашей модели прогноза снега!

Обучение модели с помощью TF

Модель, разработанная для прогнозирования снегопада, представляет собой бинарный классификатор, и может быть проиллюстрирована следующей схемой:

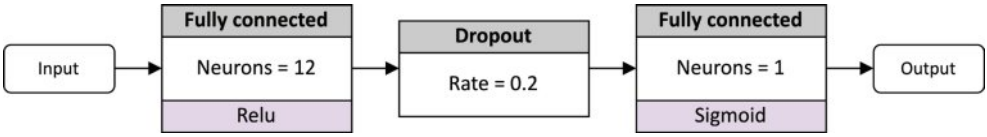


Рис. 3.5 ❖ Нейросетевая модель для прогнозирования снежного покрова

Нейронная сеть состоит из следующих слоев:

- один полносвязный слой с 12 нейронами и последующей функцией активации ReLU;
- один отсеивающий слой (dropout layer) с коэффициентом 20 % (0.2) для предотвращения переобучения;
- один полносвязный слой с одним выходным нейроном и последующей сигмоидной функцией активации.

В этом рецепте мы будем обучать предварительную модель с помощью TF.

Код, описанный в этом примере, содержится в разделе *Training the ML model with TF* файла Colab *preparing_model.ipynb*: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter03/ColabNotebooks/preparing_model.ipynb.

Подготовка

Модель, разработанная по этому описанию, имеет один узел входа и один выхода. Узел входа предоставляет сети шесть входных параметров: температуру и влажность за каждый из последних трех часов.

Модель использует входные параметры и возвращает вероятность класса через выходной узел. Поскольку выходные данные выдает функция сигмоидного вида, результат находится между нулем и единицей и считается `No`, когда он ниже 0.5, и `Yes` в противном случае.

В общем случае мы рассматриваем следующие четыре последовательных шага при обучении нейронной сети.

1. Кодирование названий выходных классов.
2. Разделение набора данных на обучающие, тестовые и проверочные наборы.
3. Создание модели.
4. Обучение модели.

В этом примере мы будем использовать библиотеки TF и *scikit-learn*.

Scikit-learn (https://scikit-learn.org/stable/) – это библиотека Python более высокого уровня для реализации общих алгоритмов ML, таких как SVM[47], random forests и логистическая регрессия. Это не специфичный для глубоких нейронных сетей (DNN) фреймворк, а скорее программная библиотека для широкого спектра алгоритмов ML.

Как это делается...

Следующие шаги показывают, как обучить модель, представленную в разделе «*Подготовка*», с помощью TF.

1. Извлеките входные объекты (`x`) и названия выходных сообщений (`y`) из набора `df_dataset` pandas DataFrame:

```
f_names = df_dataset.columns.values[0:6]
l_name = df_dataset.columns.values[6:7]
x = df_dataset[f_names]
y = df_dataset[l_name]
```

2. Выразите метки в числовых значениях:

```
labelencoder = LabelEncoder()
labelencoder.fit(y.Snow)
y_encoded = labelencoder.transform(y.Snow)
```

   Этот шаг преобразует выходные сообщения (`Yes` или `No`) в числовые значения, поскольку нейронные сети могут работать только с числами. Мы используем scikit-learn[48] для преобразования целевого сообщения к целочисленным значениям (ноль и единица). Для преобразования требуется вызвать следующие три функции:

   A) `LabelEncoder()` для инициализации модуля `LabelEncoder`;
   B) `fit()` для определения целевых целочисленных значений путем анализа названий выходных сообщений;
   C) `transform()` для преобразования выходных сообщений в числовые значения.

   После `transform()` закодированные сообщения доступны в `y_encoded`.

3. Разделите набор данных на обучающие, проверочные и тестовые наборы данных:

```
# Split 1 (85% vs 15%)
x_train, x_validate_test, y_train, y_validate_test = train_test_split(x, y_encoded,
test_size=0.15, random_state = 1)
# Split 2 (50% vs 50%)
x_test, x_validate, y_test, y_validate = train_test_split(x_validate_test, y_
validate_test, test_size=0.50, random_state = 3)
```

[47] Support Vector Machines, метод опорных векторов.

[48] Scikit-learn – один из наиболее широко используемых пакетов Python для подготовки и анализа данных. Scikit-learn основан на библиотеках SciPy и NumPy (см. далее).

На следующем рисунке показано, как мы разделяем наборы данных для обучения, проверки и тестирования:

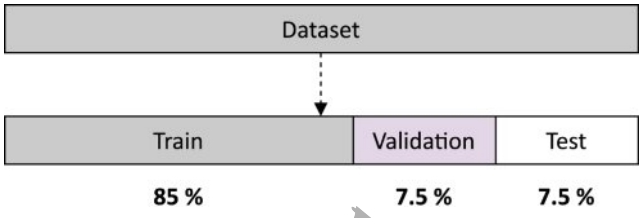


Рис. 3.6 ❖ Набор данных (*dataset*) разделен на обучающие (*train*), проверочные (*validation*) и тестовые (*test*) наборы данных

Эти три набора данных являются следующими.

- **Обучающий набор данных**: этот набор данных содержит образцы для обучения модели. Веса и смещения изучаются с помощью этих данных.
- **Проверочный набор данных**: этот набор данных содержит образцы для оценки точности модели на невидимых данных. Набор данных используется в процессе обучения, чтобы указать, насколько хорошо модель обобщается, поскольку она включает экземпляры, не включенные в обучающий набор данных. Однако, поскольку этот набор данных все еще используется во время обучения, мы могли бы косвенно повлиять на выходную модель, точно настроив некоторые обучающие гиперпараметры.
- **Тестовый набор данных**: этот набор данных содержит образцы для тестирования модели после обучения. Поскольку тестовый набор данных не используется во время обучения, он оценивает конечную модель без предвзятости.

Из исходного набора данных мы присваиваем 85 % набору данных для обучения, 7.5 % набору данных для проверки и 7.5 % набору данных для тестирования. При таком разделении набор данных проверки и тестирования будет содержать примерно 1000 выборок в каждом, чего достаточно, чтобы увидеть, работает ли модель должным образом[49].

[49] Следует подчеркнуть, что все три набора выборок взяты из одной последовательности, характерной для данного конкретного случая, и в других случаях модель будет неприменима или применима ограниченно. Для выбора местоположения это очевидно (и автор на это косвенно указывал, когда говорил о выборе курорта): скажем, в равнинной зоне механизмы снегообразования будут другими, чем в горах, поэтому та же модель окажется неработающей, например, в долине реки По, не слишком отдаленной от Канацеи. Менее очевидно, что прогноз обязательно начнет «разъезжаться» со временем: построенная модель будет хорошо работать в местности Канацеи в период 2011–2020 гг. Но в другой период в той же местности (например, из-за глобального потепления или просто из-за климатических флуктуаций) набор данных окажется другим, и построенная модель окажется ошибочной. Для преодоления этого недостатка модель необходимо все время корректировать в реальном времени, но, конечно, такой подход лежит за рамками концепции TinyML.

Разделение набора данных выполняется с помощью функции `train_test_split()` из библиотеки *scikit-learn*, которая разбивает набор данных на обучающий и тестовый наборы данных. Доля разделения определяется с помощью входного аргумента `test_size` (или `train_size`), представляющего процент входного набора данных, относящийся к тестированию (или обучению).

Мы вызываем эту функцию дважды, чтобы сгенерировать три разных набора данных. Первое разделение относит 85 % к обучающему набору данных, оставляя для тестирования `test_size=0.15`. Второе разделение создает наборы данных для проверки и тестирования, уменьшая вдвое 15%-ный набор данных из первого разделения.

4. Создайте модель с помощью Keras API:

```
model = tf.keras.Sequential()
model.add(layers.Dense(12, activation='relu', input_shape=(len(f_names),)))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
```

Представленный код выдает следующее:


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12)                84

 dropout (Dropout)           (None, 12)                0

 dense_1 (Dense)             (None, 1)                 13

=================================================================
Total params: 97
Trainable params: 97
Non-trainable params: 0
_________________________________________________________________
```

Рис. 3.7 ❖ Резюме модели, возвращаемое `model.summary()`

В резюме содержится полезная информация об архитектуре модели нейронной сети, такая как типы слоев, выходные формы и количество требуемых обучаемых весов.

В TinyML важно следить за количеством весов, потому что это связано с использованием памяти.

5. Скомпилируйте модель:

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

На этом шаге мы инициализируем следующие параметры обучения.

- **Функция потерь** (`loss`): тренинг направлен на поиск весов и отклонений, чтобы минимизировать функцию потерь. Потери указывают на то, насколько прогнозируемый результат отличается от ожидаемого результата, поэтому чем меньше потери, тем лучше модель. Метод **перекрестной энтропии** (**cross-entropy**) является стандартной функцией потерь для задач классификации, поскольку она обеспечивает более быстрое обучение с лучшим обобщением модели. Для бинарного классификатора мы должны использовать `binary_crossentropy`.
- **Показатели производительности** (`metrics`): показатели производительности оценивают, насколько хорошо модель предсказывает выходные классы. Мы используем достоверность, определяемую как отношение между количеством правильных прогнозов и общим количеством тестов:

$$accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ tests}.$$

- **Оптимизатор** (`optimizer`): это алгоритм, используемый для обновления весов сети во время обучения. Оптимизатор в основном влияет на время обучения. В нашем примере мы используем широко распространенный оптимизатор Adam.

Как только мы инициализируем параметры обучения, мы можем обучить модель.

6. Обучите модель:

```
NUM_EPOCHS=20
BATCH_SIZE=64
history = model.fit(x_train, y_train, epochs=NUM_EPOCHS, batch_size=BATCH_SIZE,
validation_data=(x_validate, y_validate))
```

Во время обучения TF сообщает о потерях и достоверности после каждого периода, как в наборах данных обучения, так и в наборах данных проверки:

```
loss: 0.3118 - accuracy: 0.8668 - val_loss: 0.3261 - val_accuracy: 0.8479
```

Рис. 3.8 ❖ Достоверность и потери сообщаются как в наборах данных обучения, так и в наборах данных проверки

Здесь `accuracy` и `loss` – это достоверность и потери в данных обучения, в то время как `val_accuracy` и `val_loss` – это достоверность и потери в данных проверки.

Лучше всего полагаться на достоверность и потерю данных проверки, чтобы предотвратить переобучение и посмотреть, как модель ведет себя на невидимых данных.

7. Постройте график достоверности и потерь за периоды обучения (`epochs`):

```
loss_train = history.history['loss']
loss_val = history.history['val_loss']
acc_train = history.history['accuracy']
acc_val = history.history['val_accuracy']
epochs = range(1, NUM_EPOCHS + 1)

def plot_train_val_history(x, y_train, y_val, type_txt):
  plt.figure(figsize = (10,7))
  plt.plot(x, y_train, 'g', label='Training'+type_txt)
  plt.plot(x, y_val, 'b', label='Validation'+type_txt)
  plt.title('Training and Validation'+type_txt)
  plt.xlabel('Epochs')
  plt.ylabel(type_txt)
  plt.legend()
  plt.show()

plot_train_val_history(epochs, loss_train, loss_val, "Loss")
plot_train_val_history(epochs, acc_train, acc_val, "Accuracy")
```

Представленный код отображает следующие два графика:

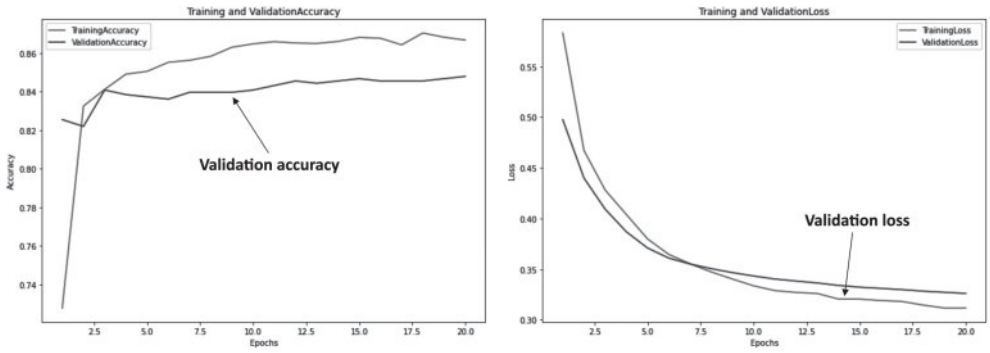


Рис. 3.9 ❖ График достоверности (слева) и потерь (справа) за периоды обучения

Из графиков достоверности и потерь во время обучения мы можем видеть тенденцию производительности модели. Тенденция подсказывает нам, следует ли нам тренироваться меньше, чтобы избежать переобучения, или больше, чтобы предотвратить недостаточную подготовку. В нашем случае лучшие результаты достоверности проверки и потерь достигаются примерно при десяти периодах. Поэтому нам следует рассмотреть возможность прекращения тренировки раньше, чтобы предотвратить переобучение. Для этого вы можете либо повторно обучить сеть в течение десяти периодов, либо использовать функцию Keras `EarlyStopping`, чтобы остановить обучение, когда отслеживаемый показатель производительности перестал улучшаться.

Вы можете узнать больше о `EarlyStopping` по следующей ссылке: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping.

8. Сохраните всю модель TF как *SavedModel*:

```
model.save("snow_forecast")
```

SavedModel – это каталог, содержащий следующее:
- модель TF в виде двоичного файла *protobuf* (с расширением файла *.pb*),
- контрольные точки TF (https://www.tensorflow.org/guide/checkpoint),
- параметры обучения, такие как оптимизатор, потери и показатели производительности.

Таким образом, приведенная выше команда создает папку *snow_forecast*, которую вы можете исследовать с помощью панели проводника слева на Colab.

Наконец-то у нас в руках модель для прогнозирования снегопада!

Оценка эффективности модели

Достоверности и потерь недостаточно, чтобы судить об эффективности модели. В целом достоверность является хорошим показателем качества, если набор данных сбалансирован, но она не говорит нам о сильных и слабых сторонах нашей модели. Например, какие классы мы распознаем с высокой степенью уверенности? Какие частые ошибки допускает модель?

Посмотрим, как мы можем оценить эффективность модели, наглядно изобразив матрицу ошибок и оценив показатели полноты (*recall*), точности (*precision*) и критерия F-score.

Код, описанный в этом примере, содержится в разделе *Evaluating the model's effectiveness* файла Colab *preparing_model.ipynb*: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter03/ColabNotebooks/preparing_model.ipynb.

Подготовка

Чтобы выполнить этот пример, нам нужно знать, что такое матрица ошибок и какие показатели качества мы можем использовать, чтобы понять, нормально ли работает модель.

В следующих подразделах будут рассмотрены эти показатели.

Наглядное представление эффективности с помощью матрицы ошибок

Матрица ошибок (*confusion matrix*) – это матрица N×N, сообщающая о количестве правильных и неправильных прогнозов в тестовом наборе данных.

Для нашей модели бинарной классификации у нас есть матрица 2×2, подобная приведенной на следующем рисунке:

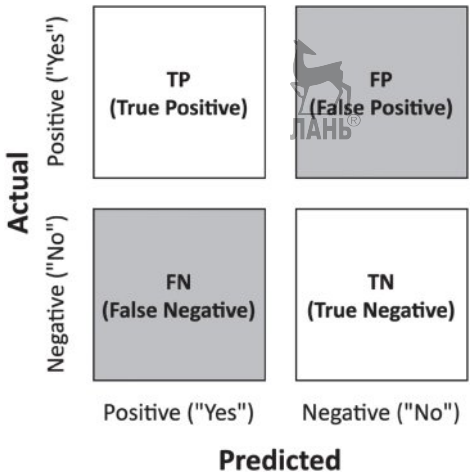


Рис. 3.10 ❖ Матрица ошибок

Четыре значения, указанные в матрице ошибок, следующие:

- истинно положительный результат (**true positive,** TP): количество прогнозируемых положительных результатов, которые на самом деле являются положительными;
- истинно отрицательный результат (**true negative**, TN): количество прогнозируемых отрицательных результатов, которые на самом деле являются отрицательными;
- ложноположительный результат (**false positive**, FP): количество прогнозируемых положительных результатов, которые на самом деле являются отрицательными;
- ложноотрицательный результат (**false negative**, FN): количество прогнозируемых отрицательных результатов, которые на самом деле являются положительными[50].

В идеале мы хотели бы иметь 100%-ную достоверность, т. е. значения, равные нулю в серых ячейках (FN и FP) матрицы ошибок, представленной на рис. 3.10. Исходя из матрицы ошибок, мы можем рассчитать фактическую достоверность по следующей формуле:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \in [0, 1].$$

Однако, как упоминалось ранее, нас больше интересуют альтернативные показатели. Эти показатели эффективности описаны в следующем подразделе.

[50] В статистике ложноположительный результат называют ошибкой I рода (засчитывание попадания при промахе), ложноотрицательный – ошибкой II рода (засчитывание промаха при попадании).

Оценка полноты (recall), точности (precision) и критерия F-score

Первым оцениваемым показателем производительности является **полнота** (*recall*), определяемая следующим образом:

$$Recall = \frac{TP}{TP + FN} \in [0, 1].$$

Этот показатель показывает нам, сколько из всех положительных (*Yes*) выборок мы предсказали правильно. Показатель полноты должен быть как можно выше.

Однако этот показатель не учитывает неправильную классификацию отрицательных выборок. Короче говоря, модель могла бы превосходно классифицировать положительные образцы, но не способна классифицировать отрицательные.

По этой причине существует еще один показатель, который учитывает отрицательные результаты FP. Это **точность** (*precision*), определяемая следующим образом:

$$Precision = \frac{TP}{TP + FP} \in [0, 1].$$

Этот показатель показывает нам, сколько предсказанных положительных классов (*Yes*) на самом деле были положительными. Точность должна быть как можно выше.

Еще один ключевой показатель производительности сочетает полноту и точность в рамках единой формулы. Это критерий **F-score**, определяемый следующим образом:

$$F\text{-}score = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}.$$

Эта формула помогает нам оценить показатели полноты и точности одновременно. Высокое значение F-score подразумевает хорошую эффективность модели.

Как это делается...

Следующие шаги научат нас, как наглядно представить матрицу ошибок и рассчитать показатели полноты, точности и F-score.

1. Графическое представление матрицы ошибок:

```
y_test_pred = model.predict(x_test)

y_test_pred = (y_test_pred > 0.5).astype("int32")

cm = sklearn.metrics.confusion_matrix(y_test, y_test_pred)
```

```
index_names = ["Actual No Snow", "Actual Snow"]
column_names = ["Predicted No Snow", "Predicted Snow"]

df_cm = pd.DataFrame(cm, index = index_names, columns = column_names)

plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='d', cmap="Blues")
plt.figure(figsize = (10,7))
```

Представленный код выдает следующее:

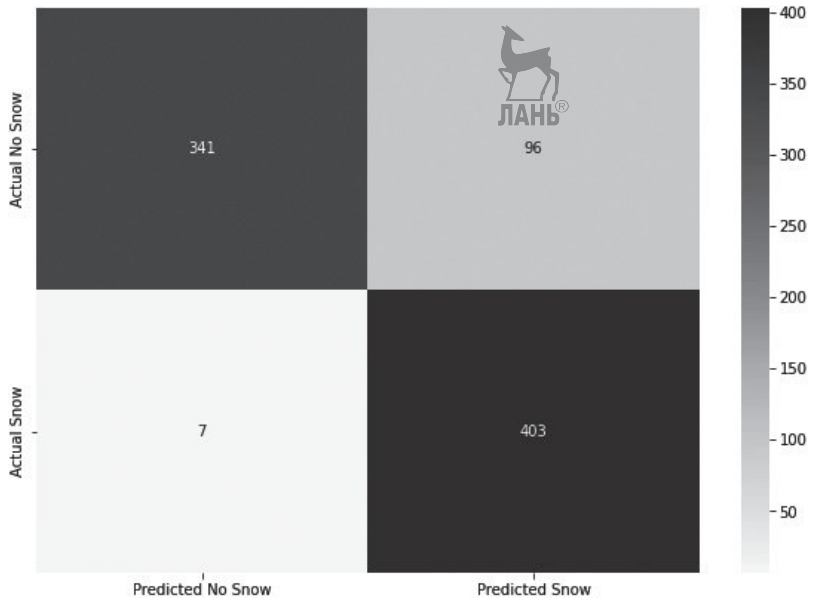


Рис. 3.11 ❖ Матрица ошибок для модели прогноза снега

Матрица ошибок получается с помощью следующих двух шагов.

A. Предсказание сообщения в тестовом наборе данных с помощью `model.predict()` при пороговом значении выходного результата в 0.5. Пороговое значение требуется, потому что `model.predict()` возвращает выходные данные в виде функции сигмоиды, представляющей собой значение от нуля до единицы.
B. Используйте функцию `confusion_matrix()` из библиотеки *scikit-learn* для вычисления матрицы ошибок (`cm`).

Из рис. 3.11 мы можем видеть, что выборки в основном распределены по ведущей диагонали, и там больше FP, чем FN. Поэтому, хотя модель подходит для обнаружения снега, мы должны ожидать некоторых ложных обнаружений.

2. Рассчитайте показатели полноты, точности и F-score:

```
TN = cm[0][0]
TP = cm[1][1]
```

```
FN = cm[1][0]
FP = cm[0][1]

precision = TP / (TP + FP)
recall = TP / (TP + FN)
f_score = (2 * recall * precision) / (recall + precision)

print("Recall: ", round(recall, 3))
print("Precision: ", round(precision, 3))
print("F-score: ", round(f_score, 3))
```

Представленный код выводит следующую информацию:

```
Recall:     0.983
Precision:  0.808
F-score:    0.887
```

Рис. 3.12 ❖ Ожидаемые результаты по полноте, точности и F-score

Как мы можем видеть из ожидаемых результатов, полнота равна 0.983, поэтому наша модель может прогнозировать снег с высокой степенью достоверности. Однако точность ниже 0.808. Эта величина показывает, что мы должны ожидать некоторых ложных срабатываний от нашей модели. Наконец, значение 0.887, полученное для F-score, говорит нам о том, что полнота и точность сбалансированы. Таким образом, у нас в руках хорошая ML-модель, способная прогнозировать снегопад с предоставленными входными функциями.

Теперь модель обучена и проверена. Следовательно, пришло время сделать ее пригодной для развертывания в микроконтроллере.

Квантизация модели с помощью конвертера TFLite

При экспорте обученной сети в форме `SavedModel` сохраняются исходные зависимости, такие как архитектура сети, веса, обучающие переменные и контрольные точки. Таким образом, сгенерированная модель TF идеально подходит для совместного использования или возобновления сеанса обучения, но не подходит для развертывания на микроконтроллерах по следующим причинам:

- веса хранятся в формате с плавающей запятой;
- модель хранит информацию, которая не требуется для расчетов.

Поскольку наше целевое устройство имеет ограничения в вычислительной мощности и памяти, крайне важно преобразовать обученную модель во что-то более компактное.

Этот пример научит, как квантизовать и преобразовать обученную модель в легкий, экономичный по объему памяти и простой в разборе формат экспорта с помощью **TensorFlow Lite** (**TFLite**).

Затем сгенерированная модель будет преобразована в C-байтовый массив, подходящий для развертывания на микроконтроллере.

Код, описанный в этом примере, содержится в разделе *Quantizing the model with TFLite converter* файла Colab *preparing_model.ipynb*: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter03/ColabNotebooks/preparing_model.ipynb.

Подготовка

Основные особенности этого примера – применение конвертера TFLite и квантизация.

TFLite (https://www.tensorflow.org/lite) – это платформа глубокого обучения, специально предназначенная для выполнения расчетов модели на периферийных устройствах, таких как смартфоны или встроенные микроконтроллерные устройства.

Как показано на следующем рисунке, TFLite предоставляет набор инструментов для достижения следующих целей:

- преобразования модели TF в облегченное представление;
- эффективного выполнения модели на целевом устройстве.

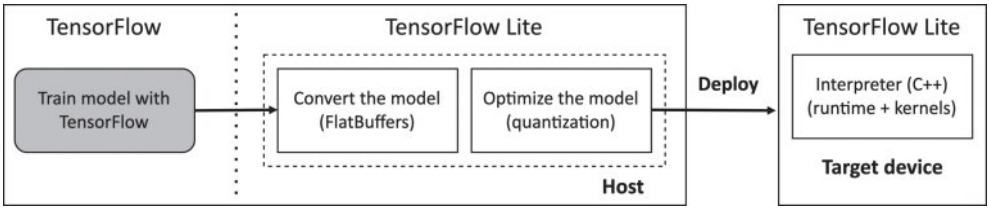


Рис. 3.13 ❖ Составляющие TFLite

Облегченное представление модели, используемое TFLite, идентифицируется расширением *.tflite*, и оно внутренне представлено в виде структур **FlatBuffers** (https://google.github.io/flatbuffers/). Формат FlatBuffers предлагает гибкую, простую в анализе и экономичную в использовании памяти структуру. Конвертер TFLite отвечает за преобразование модели TF в FlatBuffers с оптимизацией, основанной на 8-битной целочисленной квантизации, в целях уменьшения размера модели и снижения задержки.

Квантизация входной модели

Незаменимым методом для того, чтобы сделать модель подходящей для микроконтроллеров, является квантизация.

Квантизация имеет три существенных преимущества:

- уменьшает размер модели за счет преобразования всех весов к более низкой битовой точности;

- снижает энергопотребление за счет уменьшения ширины и пропускной способности каналов памяти;
- повышает производительность расчетов за счет использования целочисленной арифметики для всех операций.

Этот широко распространенный метод применяет квантизацию после обучения и преобразует 32-разрядные веса с плавающей запятой до 8-разрядных целых значений. Чтобы понять, как работает квантизация, рассмотрим следующую C-подобную функцию, которая аппроксимирует 32-разрядное значение с плавающей запятой при помощи 8-разрядного значения:

```
float dequantize(int8 x, float zero_point, float scale) {
  return ((float)x - zero_point) * scale;
}
```

В этом фрагменте `x` – квантизованное значение, представленное в виде 8-битного целого числа со знаком, `scale` и `zero_point` являются параметрами квантизации. Параметр `scale` используется для отображения квантизованного значения на диапазон с плавающей запятой; `zero_point` (нулевая точка) – смещение, которое следует учитывать для квантизованного диапазона.

Чтобы понять, почему `zero_point` не может быть равно нулю, рассмотрим следующее распределение входных данных с плавающей точкой, которое мы хотим масштабировать до 8-битного диапазона:

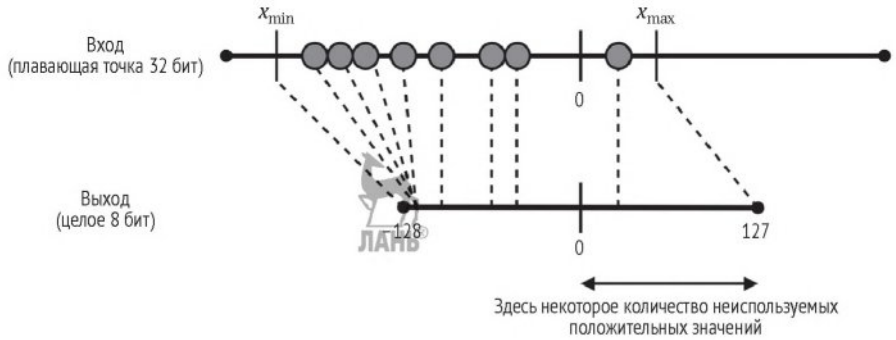


Рис. 3.14 ❖ Пример, в котором распределение значений смещено в сторону отрицательного диапазона

Следующий рисунок показывает, что входное распределение с плавающей запятой не центрировано по нулю, а сдвинуто в сторону отрицательного диапазона. Следовательно, если бы мы просто масштабировали значения с плавающей запятой до 8 бит, мы могли бы получить следующее:

- несколько отрицательных входных значений с одним и тем же 8-разрядным аналогом;
- много неиспользуемых положительных 8-битных значений.

Следовательно, было бы неэффективно присваивать ноль нулевой точке, поскольку мы могли бы выделить больший диапазон для отрицательных

значений, чтобы уменьшить их ошибку квантизации, определяемую следующим образом:

$$\varepsilon = X_{\text{real}} - Z_{\text{quantised}}.$$

Когда нулевая точка не равна нулю, мы обычно называем квантизацию асимметричной, так как назначаем разный диапазон значений для положительных и отрицательных значений:

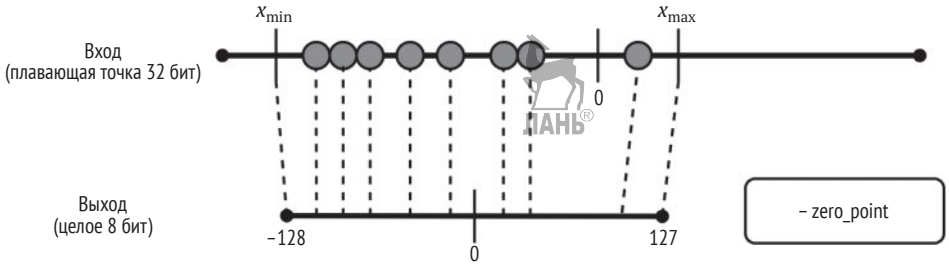


Рис. 3.15 ❖ Асимметричная квантизация

Когда нулевая точка равна нулю, мы обычно называем квантизацию симметричной, так как она симметрична относительно нуля:

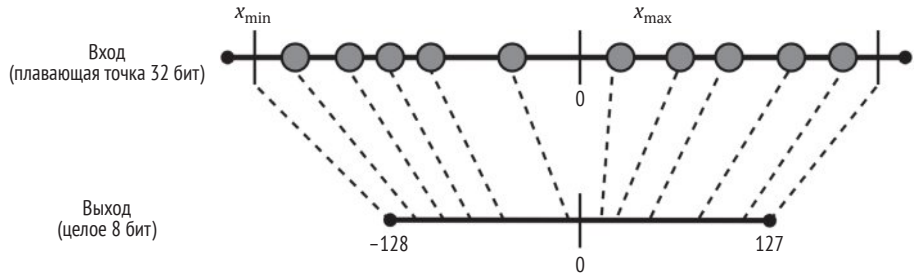


Рис. 3.16 ❖ Симметричная квантизация

Обычно мы применяем симметричную квантизацию к весам модели и асимметричную ко входу и выходу слоев.

Единственные параметры, необходимые для квантизации, – значения `scale` и `zero_point`, и обычно они предоставляются следующими способами.

- **Потензорная (per-tensor)**: параметры квантизации одинаковы для всех тензорных элементов.
- **Поканальная (per-channel)**: параметры квантизации различны для каждого набора признаков тензора.

Следующая иллюстрация наглядно описывает потензорную и поканальную квантизацию:

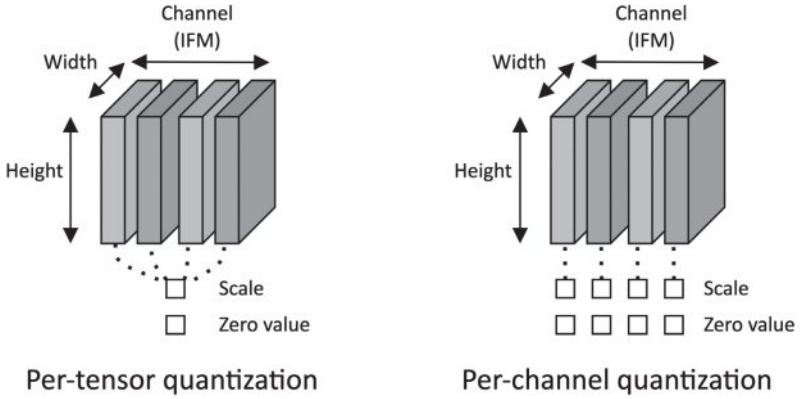


Рис. 3.17 ❖ Квантизация потензорная и поканальная

Обычно мы используем тензорный подход, за исключением весов и смещений слоев свертки (*convolution*) и свертки по глубине (*depth-wise convolution*).

Как это делается...

Использование конвертора TFLite для квантизации и создания модели, подходящей для микроконтроллеров, предполагает следующие шаги.

1. Выберите несколько сотен выборок случайным образом из тестового набора данных для калибровки квантизации:

```
def representative_data_gen():
  for i_value in tf.data.Dataset.from_tensor_slices(x_test).batch(1).take(100):
  i_value_f32 = tf.dtypes.cast(i_value, tf.float32)
  yield [i_value_f32]
```

   Этот шаг обычно называется созданием репрезентативного набора данных, и он необходим для минимизации риска снижения точности квантизации. Фактически конвертор использует этот набор выборок, чтобы определить диапазон входных значений, а затем оценить параметры квантизации. Как правило, достаточно ста выборок, которые могут быть взяты из тестового или обучающего набора данных. В нашем случае мы использовали тестовый набор данных.
2. Импортируйте каталог `SavedModel` в TFLite converter:

```
converter = tf.lite.TFLiteConverter.from_saved_model("snow_forecast")
```

3. Инициализируйте конвертор TFLite для 8-битной квантизации:

```
# Representative dataset
converter.representative_dataset = tf.lite.RepresentativeDataset(representati
ve_data_gen)
# Optimizations
```

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
# Supported ops
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
# Inference input/output type
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8
```

На этом шаге мы настраиваем конвертор TFLite для применения 8-битной квантизации.

Входные аргументы, передаваемые инструменту, следующие:

- **Representative dataset**: репрезентативный набор данных, созданный на первом шаге;
- **Optimizations** (*Оптимизация*): определяет стратегию оптимизации, которую следует принять. На данный момент поддерживается только оптимизация по умолчанию, которая пытается оптимизировать как размер, так и задержку, сводя к минимуму снижение достоверности;
- **Supported ops** (*Поддерживаемые операции*): здесь вынуждает использовать только целочисленные 8-разрядные операторы во время конверсии. Если наша модель имеет неподдерживаемые ядра, преобразование не будет успешным;
- **Inference input/output type** (*Расчетный тип ввода/вывода*): здесь используется 8-битный формат квантизации для входных и выходных величин сети. Следовательно, нам нужно будет снабдить модель ML квантизованными входными характеристиками, чтобы правильно выполнить запуск расчетов.

Как только мы инициализируем конвертер TFLite, мы можем выполнить преобразование:

```
tflite_model_quant = converter.convert()
```

4. Сохраните преобразованную модель в файл типа *.tflite*:

```
open("snow_forecast_model.tflite", "wb").write(tflite_model_quant)
```

5. Преобразуйте модель TFLite в C-байтовый массив с помощью `xxd`:

```
!apt-get update && apt-get -qq install xxd
!xxd -i snow_forecast_model.tflite > model.h
```

Эта команда выводит заголовочный файл C (опция -i), содержащий модель TFLite в виде массива `unsigned char` с шестнадцатеричными числами. Однако в разделе «*Подготовка*» мы упомянули, что модель представляет собой файл с расширением *.tflite*. Итак, зачем нам это дополнительное преобразование? Преобразование в C-байтовый массив имеет решающее значение для развертывания модели на микроконтроллерах, поскольку формат *.tflite* требует дополнительной библиотеки программного обеспечения в приложении для загрузки файла из памяти. Нам нужно помнить, что большинство микроконтроллеров не имеет поддержки операционной системы и встроенной файловой системы. Следовательно, формат C-байтового массива позволяет нам

интегрировать модель непосредственно в приложение. Другой важной причиной этого преобразования является то, что файл *.tflite* не позволяет сохранять веса в памяти программы. Поскольку важен каждый байт, а емкость SRAM ограничена, хранение модели в памяти программы, как правило, более эффективно, когда веса постоянны.

Теперь вы можете загрузить сгенерированный файл *model.h* с помощью левой панели Colab. Модель TFLite хранится в массиве `snow_forecast_model_tflite`.

Использование встроенного датчика температуры и влажности на Arduino Nano

Как мы знаем, Arduino Nano и Raspberry Pi Pico обладают уникальными аппаратными возможностями, которые делают их идеальными для различных сценариев применения. Например, Arduino Nano имеет встроенный датчик температуры и влажности, так что для нашего проекта с этой платой не требуются внешние компоненты[51].

В этом примере мы покажем, как считывать данные датчика температуры и влажности, размещенного на плате Arduino Nano.

Скетч Arduino *06_sensor_arduino_nano.ino*, содержащий код, описываемый в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter03/ArduinoSketches/06_sensor_arduino_nano.ino.

Подготовка

Нет никаких особых новых сведений, которые нужно знать для выполнения этой задачи. На этот раз в разделе «*Подготовка*» будет дан лишь обзор основных характеристик встроенного датчика температуры и влажности Arduino Nano.

Плата Arduino Nano интегрирует датчик HTS221 (https://www.st.com/resource/en/datasheet/HTS221.pdf) от ST (https://www.st.com/) для измерения относительной влажности и температуры.

Датчик ультракомпактен (2×2 мм) и обеспечивает измерения с помощью двух последовательных интерфейсов на выбор: I²C или SPI. В следующей таблице приведены основные характеристики чувствительного элемента датчика:

[51] С той оговоркой, что встроенный датчик Arduino Nano 33 BLE измеряет температуру и влажность непосредственно на поверхности платы. Эти величины могут значительно отличаться от параметров окружающего воздуха, так как плата содержит нагревающиеся компоненты (стабилизаторы питания, микроконтроллеры) в непосредственной близости от датчика. Разница в параметрах дополнительно усугубляется в случае установки платы в корпус, изолирующий ее от окружающей среды.

Таблица 3.1. Основные характеристики датчика температуры и влажности HTS221

Диапазон относительной влажности	0–100 %
Диапазон температур	–40...+120 °C
Погрешность влажности	±3.5 %
Погрешность температуры	±0.5 °C
Потребляемый ток	2 мкА при измерениях 1 раз в секунду

Как мы можем видеть из таблицы, датчик обладает чрезвычайно низким энергопотреблением, поскольку ток потребления находится в диапазоне мкА.

Как это делается...

Чтобы инициализировать и протестировать датчик температуры и влажности на Arduino Nano, создайте новый скетч в Arduino IDE и выполните следующие действия.

1. Включите в скетч заголовочный файл *Arduino_HTS221.h*:

```
#include <Arduino_HTS221.h>
```

2. Создайте функциональные макросы для считывания температуры и влажности:

```
#define READ_TEMPERATURE() HTS.readTemperature()
#define READ_HUMIDITY() HTS.readHumidity()
```

   Причина определения двух предыдущих макросов С заключается в том, что Raspberry Pi Pico будет использовать отличающиеся функции для считывания температуры и влажности с датчика. Поэтому более практично иметь общий интерфейс, чтобы приложения Arduino Nano и Raspberry Pi Pico могли совместно использовать большую часть своего кода.

3. Инициализируйте последовательный порт и датчик HTS221 в функции `setup()`:

```
void setup() {
  Serial.begin(9600);
  while (!Serial);
  if (!HTS.begin()) {
    Serial.println("Failed initialization of HTS221!");
    while (1);
  }
}
```

   Последовательный порт будет использоваться для представления результатов измерения.

Как сообщается в FAQ по плате Arduino Nano 33 BLE Sense, когда плата питается от USB, из-за разогрева датчик HTS221 становится ненадежным и имеет ошибку в каждом показании, зависящую от внешней температуры.

Мы рекомендуем отсоединить USB-кабель и запитать плату батарейками через вывод VIN для получения надежных измерений. Смотрите главу 2 «Прототипирование на микроконтроллерах», чтобы узнать, как включить Arduino Nano с батарейками[52].

Использование датчика DHT22 с Raspberry Pi Pico

В отличие от Arduino Nano, Raspberry Pi Pico для измерения температуры и влажности требует внешнего модуля датчика и дополнительной библиотеки. В этом примере мы покажем, как с Raspberry Pico для измерения температуры и влажности использовать датчик DHT22.

Скетч Arduino *07_sensor_rasp_pico.ino*, содержащий код, описываемый в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter03/ArduinoSketches/07_sensor_rasp_pico.ino.

Подготовка

Модуль датчика температуры и влажности, рассматриваемый для Raspberry Pi Pico, – это недорогой AM2302, который вы можете приобрести либо у Adafruit (https://www.adafruit.com/product/393), либо на Amazon[53].

Как показано на рис. 3.18, модуль AM2302 представляет собой компонент с тремя выводами, в который встроен датчик температуры и влажности DHT22:

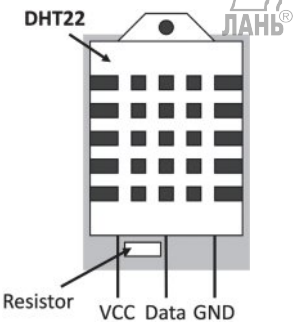


Рис. 3.18 ❖ Модуль AM2302 с датчиком DHT22

[52] Что, однако, проблему, изложенную в предыдущей сноске на стр. 114, совершенно не решает – стабилизатор питания при новых батарейках будет греться даже больше, чем при питании от USB, так как входное напряжение выше (см. в главе 2 подраздел «Подключение батарей к плате микроконтроллера»).

[53] DHT22 широко представлен в российских интернет-магазинах, ориентированных на компоненты для Arduino.

В следующей таблице приведены основные характеристики датчика DHT22:

Таблица 3.2. Основные характеристики датчика температуры и влажности DHT22

Диапазон относительной влажности	0–100 %
Диапазон температур	-40…+80 °C
Погрешность влажности	2–5 %
Погрешность температуры	±0.5 °C
Потребляемый ток	2 мА при запросе данных

DHT11 – еще один популярный датчик температуры и влажности из семейства DHT. Однако мы не можем использовать его в этом примере, потому что он имеет хорошую точность измерения температуры только между 0 и 50 °C.

В отличие от датчика HTS221 на Arduino Nano, DHT22 имеет специальный протокол для считывания значений температуры и влажности. Протокол должен быть реализован через вывод периферийного устройства GPIO и требует точных отсчетов времени для считывания данных. К счастью, Adafruit разработала библиотеку программного обеспечения (https://github.com/adafruit/DHT-sensor-library) для датчиков DHT, так что нам не нужно беспокоиться об этом. Библиотека будет иметь дело с деталями низкоуровневого программного обеспечения и предоставит API для считывания температуры и влажности.

Как это делается...

Чтобы использовать датчик DHT22 с Raspberry Pi Pico, создайте новый скетч в Arduino IDE и выполните следующие действия.

1. Подключите датчик DHT22 к Raspberry Pi Pico. Для передачи данных DHT22 используйте вывод GPIO G10 (строка 14 платы), рис. 3.19.
   Загрузите последнюю версию библиотеки программного обеспечения для датчика DHT по адресу https://www.arduino.cc/reference/en/libraries/dht-sensor-library/.
   В среде IDE Arduino импортируйте ZIP-файл библиотеки, нажав на вкладку **Libraries** на левой панели, как показано на рис. 3.20.
   Всплывающее окно сообщит нам, что библиотека была успешно импортирована.
3. Включите заголовочный файл *DHT.h* в скетч:

```
#include <DHT.h>
```

4. Определите глобальный объект DHT для взаимодействия с датчиком DHT22:

```
const int gpio_pin_dht_pin = 10;
DHT dht(gpio_pin_dht_pin, DHT22);
```

   Объект `DHT` инициализируется с помощью вывода GPIO, используемого линией передачи данных DHT22 (G10), и типа датчика DHT (DHT22).

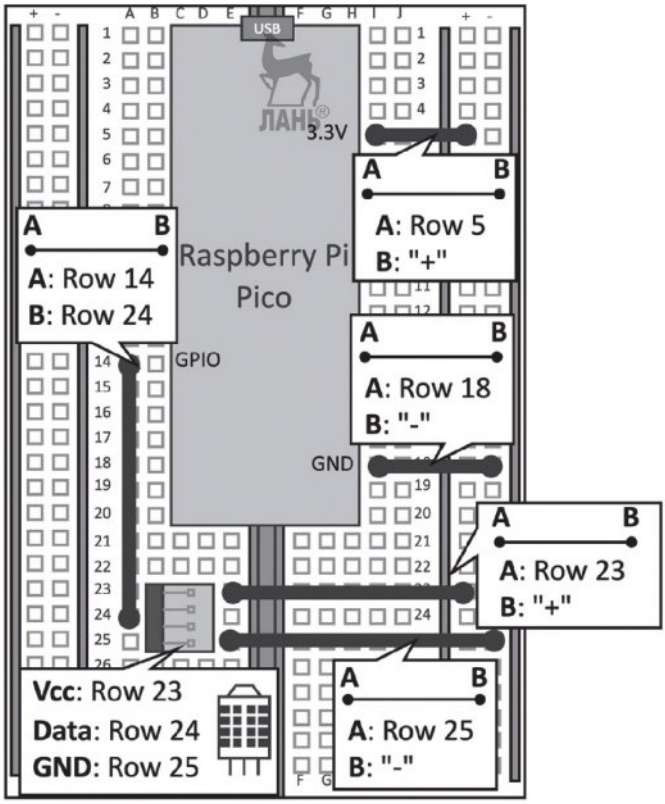


Рис. 3.19 ❖ Полная схема с Raspberry Pi Pico и модулем датчика AM2302

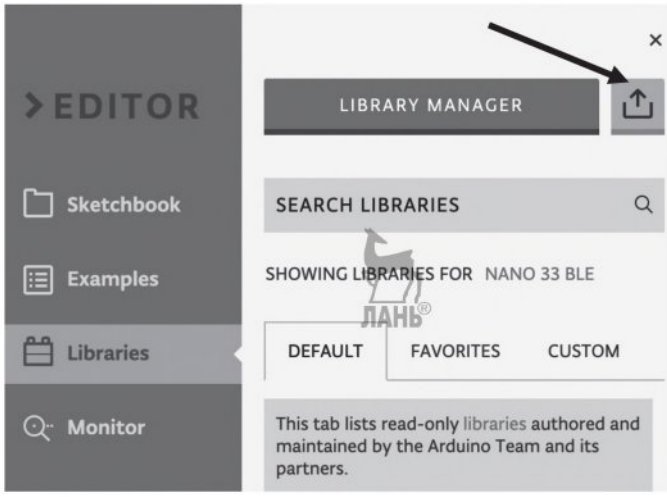


Рис. 3.20 ❖ Импорт библиотеки датчиков DHT в редакторе Arduino Web Editor

5. Создайте функциональные макросы для считывания температуры и влажности:

```
#define READ_TEMPERATURE() dht.readTemperature()
#define READ_HUMIDITY() dht.readHumidity()
```

   Имя функции должно совпадать с именем в предыдущем примере. Этот шаг обеспечивает общий функциональный интерфейс для измерения температуры и влажности на Arduino Nano и Raspberry Pi Pico.
6. Инициализируйте последовательный порт и датчик DHT22 в функции `setup()`:

```
void setup() {
  Serial.begin(9600);
  while(!Serial);
  dht.begin();
  delay(2000);
}
```

   DHT22 может возвращать новые данные только через две секунды. По этой причине мы в ожидании готовности датчика используем `delay(2000)`.

Теперь Raspberry Pi Pico может считывать данные датчика температуры и влажности.

Подготовка входных характеристик для просчета модели

Как мы знаем, входными характеристиками модели являются масштабированные и квантизованные значения температуры и влажности за последние три часа. Используя эти данные, модель ML может спрогнозировать, будет ли идти снег.

В этом примере мы увидим, как подготовить входные данные для ввода в нашу модель ML. В частности, этот пример научит нас, как получать, масштабировать и квантизовать измерения датчиков и сохранять их в порядке получения по времени, используя циклический буфер.

Скетч Arduino *08_input_features.ino*, содержащий код, описываемый в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter03/ArduinoSketches/08_input_features.ino.

Подготовка

Чтобы получить необходимые входные характеристики для модели, наше приложение будет получать данные о температуре и влажности каждый час. Однако как мы можем сохранить последние три измерения во временном порядке, чтобы передать в нейронную сеть правильные входные данные?

В этом примере мы будем использовать кольцевой буфер (*circular buffer*), структуру данных фиксированного размера, которая реализует структуру **First-In-First-Out** («*первый вошел – первый вышел*», **FIFO**). Эта структура данных хорошо подходит для буферизации потоков данных и может быть реализована с помощью массива и указателя, который указывает, где хранить элемент в памяти. На следующем рисунке показано, как работает кольцевой буфер с тремя элементами:

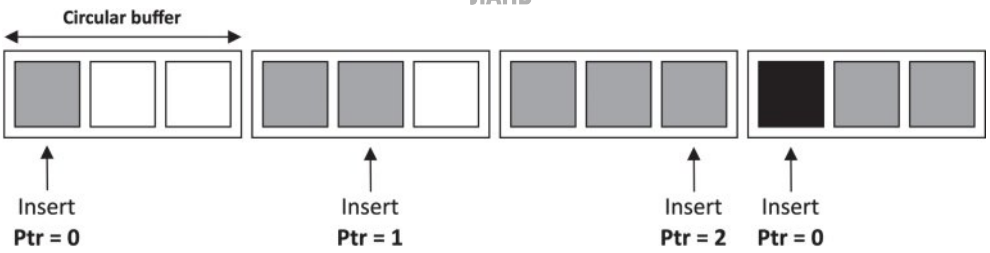


Рис. 3.21 ❖ Кольцевой буфер с тремя элементами

Как вы можете видеть из рисунка, эта структура данных имитирует кольцо, поскольку указатель (*Ptr*) увеличивается после каждого добавления данных и переходит в начало, когда достигает конца.

Как это делается...

Инструкции, приведенные в этом разделе, применимы как к Arduino Nano, так и к Raspberry Pi Pico. Чтобы создать циклический буфер и подготовить входные данные для запуска расчета модели, выполните следующие действия.

1. Определите два глобальных массива `int8_t` размером 3 и целочисленную переменную для реализации циклической структуры данных буфера:

```
#define NUM_HOURS 3
int8_t t_vals [NUM_HOURS] = {0};
int8_t h_vals [NUM_HOURS] = {0};
int cur_idx = 0;
```

   Эти два массива будут использоваться для сохранения масштабированных и квантизованных измерений температуры и влажности в порядке поступления по времени.

2. Определите параметры квантизации входных параметров – две переменные для `scale` (типа `float`) и `zero point` (типа `int32_t`):

```
float tflu_i_scale = 0.0f;
int32_t tflu_i_zero_point = 0;
```

   Следующий пример извлечет эти параметры квантизации из модели TF.

Пожалуйста, обратите внимание, что масштаб (переменная `tflu_i_scale`) является числом с плавающей запятой, в то время как нулевая точка (`tflu_i_zero_point`) является 32-разрядным целым числом.

3. Возьмите среднее значение трех образцов температуры и влажности, записываемых каждые три секунды в функции `loop()`:

```
constexpr int num_reads = 3;
void loop() {
  float t = 0.0f;
  float h = 0.0f;
  for(int i = 0; i < num_reads; ++i) {
    t += READ_TEMPERATURE();
    h += READ_HUMIDITY();
    delay(3000);
  }
  t /= (float)num_reads;
  h /= (float)num_reads;
```

Захват более чем одной выборки, как правило, является хорошим способом проведения надежных измерений.

4. Масштабируйте данные о температуре и влажности с помощью Z-score в функции `loop()`:

```
constexpr float t_mean = 2.05179f;
constexpr float h_mean = 82.30551f;
constexpr float t_std = 7.33084f;
constexpr float h_std = 14.55707f;
t = (t - t_mean) / t_std;
h = (h - h_mean) / h_std;
```

Z-score требует среднего значения и стандартного отклонения, которые мы рассчитали во втором разделе этой главы.

5. Квантизация входных характеристик в функции `loop()`:

```
t_vals[cur_idx] = (t / tflu_i_scale) + tflu_i_zero_point;
h_vals[cur_idx] = (h / tflu_i_scale) + tflu_i_zero_point;
```

Выборки квантизуются с использованием входных параметров `tflu_i_scale` и `tflu_i_zero_ point`. Помните, что входные данные модели используют схему квантизации для каждого тензора, поэтому все входные объекты должны быть квантизованы с одинаковыми масштабом и нулевой точкой.

6. Храните данные температуры и влажности в кольцевом буфере:

```
t_vals[cur_idx] = t;
h_vals[cur_idx] = h;
cur_idx = (cur_idx + 1) % NUM_HOURS;
delay(2000);
```

Указатель кольцевого буфера (`cur_index`) обновляется после каждой вставки данных с помощью следующей формулы:

$index_{new} = (index_{current} + 1)$ % $length_{array}$.

В этой формуле $length_{array}$ – размер циклического буфера, $index_{current}$ и $index_{new}$ – значения указателя до и после вставки данных.

В конце кода у нас есть задержка в две секунды, но в реальном приложении она должна составлять один час. Двухсекундная пауза используется для того, чтобы избежать слишком долгого ожидания в наших экспериментах.

Запуск на устройстве с помощью TFLu

Вот мы и пришли к нашему первому приложению ML для микроконтроллеров.

В этом примере мы, наконец, узнаем, как использовать **TensorFlow Lite for microcontrollers** (TFLu) для просчета модели TFLite на Arduino Nano и Raspberry Pi Pico.

Скетч Arduino *09_classification.ino*, содержащий код, описываемый в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter03/ArduinoSketches/09_classification.ino.

Подготовка

Чтобы выполнить этот последний пример, необходимо знать, как модель просчитывается с помощью TFLu.

TFLu был представлен в главе 1 «*Начало работы с TinyML*» и представляет собой программный компонент, который запускает модели TFLite на микроконтроллерах.

Просчет с помощью TFLu обычно состоит из следующего.

1. **Загрузка и синтаксический разбор (парсинг) модели**: TFLu анализирует веса и сетевую архитектуру, хранящиеся в C-байтовом массиве.
2. **Преобразование входных данных**: входные данные, полученные от датчика, преобразуются в нужный формат в соответствии с требованиями модели.
3. **Выполнение модели**: TFLu выполняет модель, используя оптимизированные функции DNN.

При работе с микроконтроллерами необходимо оптимизировать каждую строку нашего кода, чтобы свести к минимуму объем памяти при максимальной производительности.

По этой причине TFLu также интегрирует библиотеки программного обеспечения для получения наилучшей производительности от различных целевых процессоров. Например, TFLu поддерживает CMSIS-NN (https://www.keil.com/pack/doc/CMSIS/NN/html/index.html), бесплатную библиотеку с открытым исходным кодом, разработанную ARM для оптимизированных операторов DNN на архитектурах ARM Cortex-M. Эти оптимизации имеют отношение к критическим примитивам DNN, таким как свертка, свертка по глубине

и полносвязный слой, и совместимы с ARM-процессорами в Arduino Nano и Raspberry Pi Pico.

На данный момент у вас может возникнуть один вопрос: как мы можем использовать TFLu с библиотекой CMSIS-NN?

Нам не нужно устанавливать дополнительные библиотеки, потому что TFLu для Arduino содержит CMSIS-NN. Следовательно, Arduino автоматически включит CMSIS-NN для более быстрого просчета приложения при использовании TFLu.

Как это делается...

Инструкции в этом разделе применимы как к Arduino Nano, так и к Raspberry Pi Pico. Следующие шаги покажут, как использовать TFLu для запуска модели прогноза снега TFLite на наших платах.

1. Импортируйте файл *model.h* в проект Arduino. Как показано на следующем рисунке, нажмите на вкладке кнопку с перевернутым треугольником и выберите **Import File into Sketch**:

Рис. 3.22 ❖ Импорт файла *model.h* в проект Arduino

Появится окно папки, из которого вы можете перетащить файл модели TFLu.

Как только файл будет импортирован, включите в скетч C-заголовок:

```
#include "model.h"
```

2. Включите заголовочные файлы, требуемые TFLu:

```
#include <TensorFlowLite.h>
#include <tensorflow/lite/micro/all_ops_resolver.h>
#include <tensorflow/lite/micro/micro_error_reporter.h>
#include <tensorflow/lite/micro/micro_interpreter.h>
#include <tensorflow/lite/schema/schema_generated.h>
#include <tensorflow/lite/version.h>
```

Основными заголовочными файлами являются следующие:

- `all_ops_resolver.h`: для загрузки операторов DNN, необходимых для запуска модели ML;

- `micro_error_reporter.h`: для вывода отладочной информации, возвращаемой средой выполнения TFLu;
- `micro_interpreter.h`: для загрузки и выполнения модели ML;
- `schema_generated.h`: для схемы формата TFLite FlatBuffer;
- `version.h`: для управления версиями схемы TFLite.

Для получения дополнительной информации о файлах заголовков мы рекомендуем прочитать руководство по началу работы с микроконтроллером в документации TF (https://www.tensorflow.org/lite/microcontrollers/get_started_low_level).

3. Объявите переменные, требуемые TFLu:

```
const tflite::Model* tflu_model           = nullptr;
tflite::MicroInterpreter* tflu_interpreter = nullptr;
TfLiteTensor* tflu_i_tensor                = nullptr;
TfLiteTensor* tflu_o_tensor                = nullptr;
tflite::MicroErrorReporter tflu_error;
constexpr int tensor_arena_size = 4 * 1024;
byte tensor_arena[tensor_arena_size] __attribute__((aligned(16)));
```

Глобальные переменные, объявленные на этом шаге, следующие:

- `tflu_model`: модель, анализируемая анализатором TFLu;
- `tflu_interpreter`: указатель на интерпретатор TFLu;
- `tflu_i_tensor`: указатель на входной тензор модели;
- `tflu_o_tensor`: указатель на выходной тензор модели;
- `tensor_arena`: память, требуемая интерпретатором TFLu. TFLu не использует динамическое распределение. Следовательно, мы должны предоставить фиксированный объем памяти для входных, выходных и промежуточных тензоров. Размер памяти зависит от модели и определяется только экспериментами. В нашем случае 4096 более чем достаточно.

Объявленные переменные, как правило, требуются во всех приложениях на основе TFLu.

4. Загрузите в функции `setup()` модель TFLite из C-байтового массива `snow_forecast_model_tflite`:

```
tflu_model = tflite::GetModel(snow_forecast_model_tflite);
```

Определите объект `tflite::AllOpsResolver` в функции `setup()`:

```
tflite::AllOpsResolver tflu_ops_resolver;
```

Интерпретатор TFLu будет использовать этот интерфейс для поиска указателей функций для каждого DNN-оператора.

6. Создайте в функции `setup()` интерпретатор TFLu:

```
tflu_interpreter = new tflite::MicroInterpreter(tflu_model, tflu_ops_resolver,
tensor_arena, tensor_arena_size, &tflu_error);
```

7. В функции `setup()` выделите память, необходимую для модели, и получите указатель памяти входных и выходных тензоров:

```
tflu_interpreter->AllocateTensors();
tflu_i_tensor = tflu_interpreter->input(0);
tflu_o_tensor = tflu_interpreter->output(0);
```

8. В функции `setup()` получите параметры квантизации для входного и выходного тензоров:

```
const auto* i_quantization = reinterpret_cast<TfLiteAffineQuantization*>(tflu_i_
tensor->quantization.params);
const auto* o_quantization = reinterpret_cast<TfLiteAffineQuantization*>(tflu_o_
tensor->quantization.params);
tflu_i_scale = i_quantization->scale->data[0];
tflu_i_zero_point = i_quantization->zero_point->data[0];
tflu_o_scale = o_quantization->scale->data[0];
tflu_o_zero_point = o_quantization->zero_point->data[0];
```

   Параметры квантизации возвращаются в объекте TfLiteAffineQuantization, содержащем два массива для параметров масштаба и нулевой точки. Поскольку как входные, так и выходные тензоры используют потензорную квантизацию, каждый массив хранит одно значение.

9. Инициализируйте в функции `loop()` входной тензор с помощью квантизованных входных функций:

```
const int idx0 = cur_idx;
const int idx1 = (cur_idx - 1 + NUM_HOURS) % NUM_HOURS;
const int idx2 = (cur_idx - 2 + NUM_HOURS) % NUM_HOURS;
tflu_i_tensor->data.int8[0] = t_vals[idx2];
tflu_i_tensor->data.int8[1] = t_vals[idx1];
tflu_i_tensor->data.int8[2] = t_vals[idx0];
tflu_i_tensor->data.int8[3] = h_vals[idx2];
tflu_i_tensor->data.int8[4] = h_vals[idx1];
tflu_i_tensor->data.int8[5] = h_vals[idx0];
```

   Поскольку нам нужны последние три выборки, мы используем следующую формулу для считывания элементов из циклического буфера:

   $index_{\text{past}} = (index_{\text{current}} - N + length_{\text{array}}) \% length_{\text{array}}$.

   В предыдущей формуле N указывает момент выборки, а $index_{\text{past}}$ есть указатель соответствующего циклического буфера. Например, если $t0$ является текущим моментом, $N = 0$ означает выборку в момент времени $t = t0$, $N = 1$ – выборку в момент времени $t = t0 - 1$, и $N = 2$ – выборку в момент времени $t = t0 - 2$.

10. Запустите расчеты в функции loop():

```
tflu_interpreter->Invoke();
```

11. В функции loop() деквантизуйте выходной тензор и прогнозируйте погодные условия:

```
int8_t out_int8 = tflu_o_tensor->data.int8[0];
float out_f = (out_int8 - tflu_o_zero_point) * tflu_o_scale;
```

```
if (out_f > 0.5) {
  Serial.println("Yes, it snows");
}
else {
  Serial.println("No, it does not snow");
}
```

Деквантизация выходных данных выполняется с помощью параметров `tflu_o_scale` и `tflu_o_zero_point`, заданных в функции `setup()`. В представлении с плавающей запятой выходные данные рассматриваются отрицательными (*No*), если они меньше 0.5, в противном случае положительными (*Yes*).

Теперь скомпилируйте и загрузите программу на плату микроконтроллера. Терминал последовательного порта в Arduino IDE сообщит «*Yes*» (пойдет снег) или «*No*» (снега не будет) в зависимости от того, прогнозируется ли снег.

Чтобы проверить, может ли приложение прогнозировать снег, вы можете просто выставить температуру на –10 °C и влажность на 100 %. Модель через терминал последовательного порта должна вернуть «*Yes*» (снег пойдет).

Глава 4

Голосовое управление светодиодами с помощью Edge Impulse

Распознавание ключевых слов (**Keyword spotting**, **KWS**) – технология, применяемая в широком спектре повседневных приложений, позволяющая работать с устройством без помощи рук. Распознавание известных слов для пробуждения «*OK Google*», «*Alexa*», «*Hey Siri*» или «*Cortana*» представляет собой пример использования этой технологии, когда умный помощник непрерывно прослушивает звуковой фон в ожидании волшебной фразы, прежде чем подключить взаимодействие с устройством.

Поскольку KWS нацелено на распознавание высказываний из речи в реальном времени, оно должно быть встроено в устройство, всегда включено и при этом эффективно работать в системе с низким энергопотреблением.

В этой главе мы продемонстрируем использование KWS с помощью **Edge Impulse**, создав приложение для голосового управления цветом **RGB**-светодиода (**red, green, blue** – красный, зеленый и синий) и количеством миганий (один, два и три раза).

Это приложение TinyML может найти место в интеллектуальных развивающих игрушках для изучения словарного запаса названий цветов и цифр со спокойной душой в отношении конфиденциальности и безопасности, поскольку оно не требует подключения к интернету.

В этой главе основное внимание будет уделено подготовке набора данных – будет показано, как получать аудиоданные с помощью мобильного телефона. Далее мы разработаем модель, основанную на **Mel-frequency cepstral coefficients** (**MFCC**), одной из самых популярных функций для распознавания речи[54]. В этих примерах мы покажем, как извлекать MFCC из аудиосемплов, обучать ML-модель и оптимизировать производительность с помощью **EON Tuner**. В конце главы мы сосредоточимся на доработке приложения KWS для Arduino Nano и Raspberry Pi Pico.

Цель этой главы – показать, как разработать сквозное (**end-to-end**, **E2E**) приложение KWS с помощью Edge Impulse, ознакомиться со сбором аудио-

[54] Об MFCC и алгоритмах на их основе подробнее см. https://habr.com/ru/post/140828/.

данных и таким периферийным устройством, как аналого-цифровой преобразователь (АЦП, **analog-to-digital converter, ADC**).

В этой главе мы собираемся выполнить примеры по следующим темам.

- Сбор аудиоданных с помощью смартфона.
- Извлечение параметров MFCC из аудиосемплов.
- Пример проектирования и обучения нейронной сети (NN).
- Настройка эффективности модели с помощью EON Tuner.
- Классификация в реальном времени с помощью смартфона.
- Классификация в реальном времени с помощью Arduino Nano.
- Непрерывное распознавание на Arduino Nano.
- Схема на Raspberry Pi Pico для голосового управления светодиодами.
- Выборка звука на Raspberry Pi Pico с помощью АЦП и прерываний по таймеру.

Технические требования

Чтобы выполнить на практике все примеры этой главы, понадобится следующее:

- плата Arduino Nano 33 BLE Sense,
- плата Raspberry Pi Pico,
- смартфон (телефон Android или Apple iPhone),
- кабель Micro Universal Serial Bus (USB),
- беспаечная макетная плата половинного размера,
- усилитель электретного микрофона – MAX9814 (только для Raspberry Pi Pico),
- 11 соединительных проводов-перемычек (штырь–штырь, только для Raspberry Pi Pico),
- 2 резистора 220 Ом (только для Raspberry Pi Pico),
- 1 резистор 100 Ом (только для Raspberry Pi Pico),
- 1 красный светодиод (только для Raspberry Pi Pico),
- 1 зеленый светодиод (только для Raspberry Pi Pico),
- 1 синий светодиод (только для Raspberry Pi Pico),
- 1 тактовая кнопка (только для Raspberry Pi Pico),
- ноутбук/ПК с Ubuntu 18.04+ или Windows 10 на x86-64.

Исходный код и дополнительные материалы доступны в папке *Chapter04* репозитория GitHub по адресу https://github.com/PacktPublishing/TinyML-Cookbook/tree/main/Chapter04.

Сбор аудиоданных с помощью смартфона

Как и для всех задач ML, сбор данных – первый шаг, который необходимо предпринять, и Edge Impulse предлагает несколько способов сделать это непосредственно из веб-браузера.

В этом примере мы узнаем, как получать звуковые семплы с помощью мобильного телефона.

Подготовка

Получение аудиосемплов с помощью смартфона – самый простой способ сбора данных, предлагаемый Edge Impulse, поскольку для этого требуется только телефон (Android phone или Apple iPhone) с подключением к интернету.

Однако сколько образцов нам нужно для обучения модели?

Сбор звуковых семплов для KWS

Количество образцов полностью зависит от характера проблемы – поэтому ни одна оценка не подходит для всех задач. Для такой ситуации, как в этом примере, 50 образцов каждого класса может быть достаточно, чтобы получить базовую модель. Однако, как правило, рекомендуется 100 или более для получения лучших результатов. Мы хотим предоставить вам полную свободу в этом выборе. Однако не забудьте про *равное количество образцов для каждого класса, чтобы получить сбалансированный набор данных*.

Какой бы размер набора данных вы ни выбрали, попробуйте включить в образцы речи различные вариации акцентов, интонаций, тембров, произношений и высоты тона. Эти вариации сделают модель способной распознавать слова, произносимые разными носителями языка. Как правило, аудиозапись, записанная лицами разного возраста и пола, должна охватывать все эти случаи.

Хотя предполагается шесть выходных классов для идентификации (*red, green, blue, one, two* и *three*), мы должны рассмотреть дополнительный класс *unknown* для случаев, когда в речи встречаются неизвестные слова.

Как это делается...

Откройте панель управления Edge Impulse и дайте название своему проекту (например, *voice_controlling_leds*).

В этом примере *N* будет использоваться для обозначения количества образцов для каждого выходного класса.

Выполните следующие действия, чтобы получить аудиоданные с помощью микрофона мобильного телефона.

1. Выберите **Let's collect some data** (*Давайте соберем некоторые данные*) в разделе **Acquire data** (*Сбор данных*).
   Затем выберите **Show QR code** (*Показать QR-код*) в разделе меню **Use your mobile phone** (*Использовать свой мобильный телефон*).

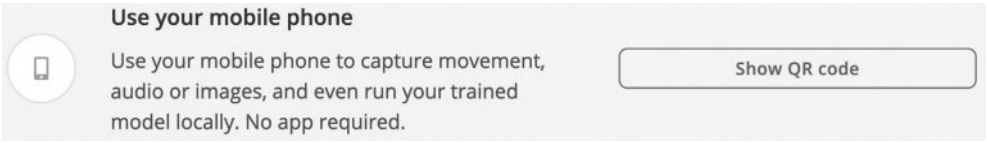


Рис. 4.1 ❖ Выберите **Show QR code** для сопряжения мобильного телефона с Edge Impulse

Отсканируйте QR-код с помощью смартфона, чтобы подключить устройство к Edge Impulse. Всплывающее окно на вашем телефоне подтвердит, что устройство подключено, как показано на следующем скриншоте:

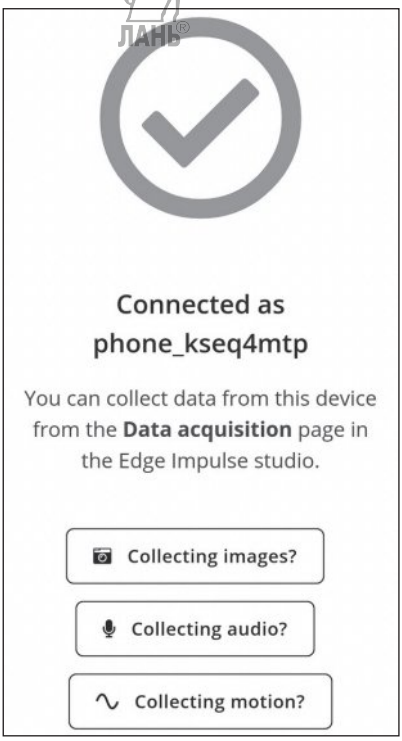


Рис. 4.2 ❖ Сообщение Edge Impulse на вашем телефоне

На своем мобильном телефоне нажмите на **Collecting audio?** (*Собирать аудио?*) и дайте разрешение использовать микрофон.

Поскольку не обязательно иметь ноутбук и смартфон в одной сети, мы можем собирать аудиосемплы где угодно. Как можно догадаться, этот подход хорошо подходит для записи звуков из различных источников, поскольку для этого требуется только телефон с подключением к интернету.

2. Запишите *N* (например, 50) звучаний для каждого класса (*red, green, blue, one, two* и *three*). Прежде чем нажать на кнопку **Start recording** (*Начать запись*), установите категорию обучения и введите одно из следующих названий в поле **Label** (*Метка*), в зависимости от произносимого слова:

Class	Red	Green	Blue	One	Two	Three
Label	00_red	01_green	02_blue	03_one	04_two	05_three

Рис. 4.3 ❖ Метки выходных категорий

Поскольку кодировка названий присваивает целочисленное значение на основе алфавитного порядка каждой категории вывода, предлагаемые нами имена меток (`00_red`, `01_green`, `02_blue`, `03_one`, `04_two` и `05_three`) помогут легко узнать, есть ли у нас цвет или число по индексу метки. Например, если индекс метки меньше 3, у нас есть цвет.

Мы рекомендуем повторять одно и то же высказывание несколько раз в одной записи, чтобы избежать загрузки слишком большого количества файлов в Edge Impulse. Например, вы можете записать звук продолжительностью 20 с, в котором вы повторяете одно и то же слово 10 раз с паузой в 1 с между ними.

Записи будут доступны в разделе **Data acquisition** (*Сбор данных*). Нажав на файл, вы можете наглядно просмотреть соответствующую форму звукового сигнала:

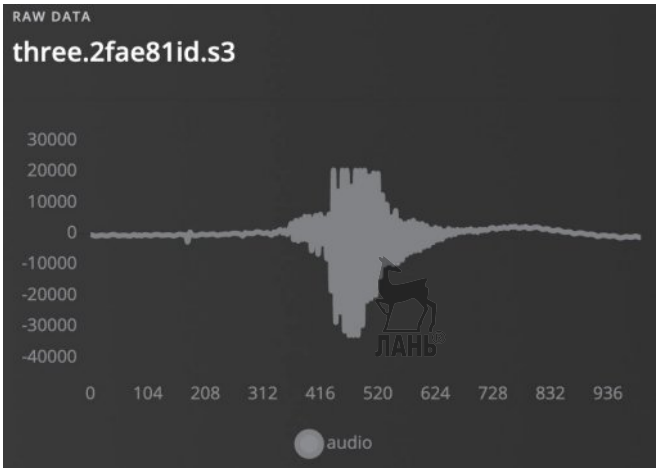


Рис. 4.4 ❖ Форма звукового сигнала

Необработанная звуковая форма представляет собой сигнал, записанный микрофоном, и графически описывает изменение звукового давления с течением времени. По вертикальной оси откладывается

амплитуда сигнала, по горизонтальной – время. Более высокая амплитуда сигнала подразумевает более громкий звук, воспринимаемый человеческим ухом.

3. Разделите записи, содержащие повторы высказывания, на отдельные семплы, нажав на : рядом с именем файла, а затем нажав на **Split sample** (*Разделить образец*), как показано на следующем скриншоте:

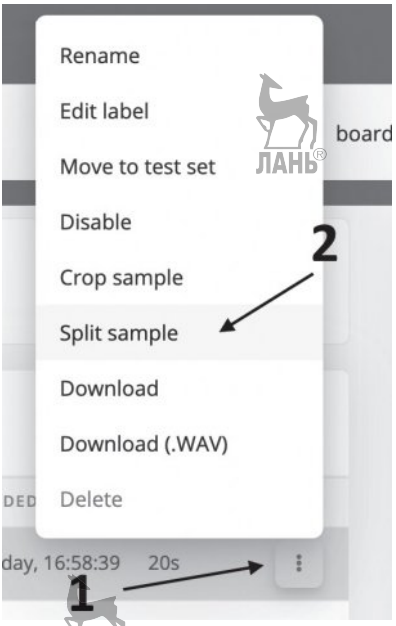


Рис. 4.5 ❖ Разделение выборки

Edge Impulse автоматически распознает произносимые слова, как вы можете видеть на следующем рисунке:

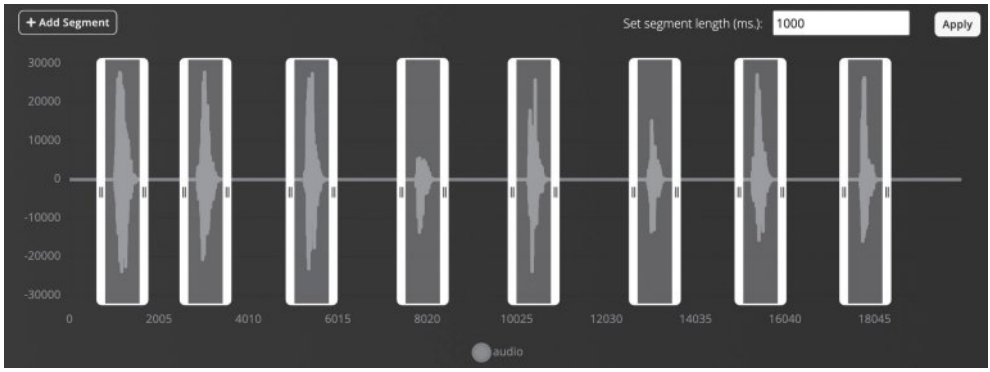


Рис. 4.6 ❖ Форма звукового сигнала
с повторениями одного и того же высказывания

Установите длину сегмента равной 1000 миллисекунд (мс) (= 1 с) и убедитесь, что все образцы расположены по центру в пределах окна нарезки. Затем нажмите на **Split** (*Разделить*), чтобы получить отдельные образцы.

4. Загрузите набор ключевых слов из Edge Impulse (https://cdn.edgeimpulse.com/datasets/keywords2.zip) и распакуйте файл. Импортируйте *N* случайных выборок из неизвестного (*unknown*) набора данных в проект Edge Impulse. Перейдите в раздел **Data acquisition** и нажмите на кнопку **Upload existing data** (*Загрузить существующие данные*) в меню **Collect data** (*Собранные данные*).

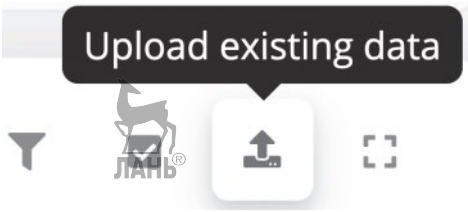


Рис. 4.7 ❖ Кнопка для загрузки существующих данных для обучения

На странице загрузки данных **Upload data** (*Загрузить данные*) выполните следующие действия:

- установите **Upload category** – категорию загрузки для обучения;
- напишите *unknown* (*неизвестно*) в поле **Enter label** (*Ввести метку*).

Нажмите на кнопку **Begin upload** (*Начать загрузку*), чтобы импортировать файлы в набор данных.

5. Разделите выборки между обучающими и тестовыми наборами данных, нажав на кнопку **Perform train / test split** (*Выполнить разделение обучения/теста*) в области **Danger zone** (*Опасная зона*) на панели инструментов:

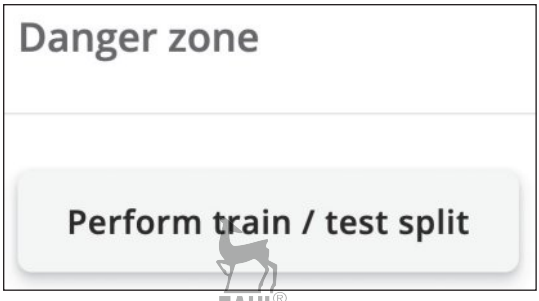


Рис. 4.8 ❖ Danger zone в Edge Impulse

Edge Impulse дважды спросит вас, уверены ли вы в этом действии, поскольку перетасовка данных необратима.

Теперь у вас должно быть 80 % образцов, назначенных набору обучения/проверки, и 20 % – набору тестирования.

Извлечение параметров MFCC из аудиосемплов

При создании приложения ML с помощью Edge Impulse за всю обработку данных, такую как извлечение параметров и расчеты модели, отвечает вычислительный блок, именуемый *Impulse*.

В этом примере мы увидим, как создать *Impulse* для извлечения параметров MFCC из аудиосемплов.

Подготовка

Давайте начнем этот пример с обсуждения того, что такое *Impulse*, и изучения функций MFCC, используемых для нашего KWS-приложения.

Impulse в Edge Impulse отвечает за обработку данных и состоит из двух вычислительных блоков, в основном выполняющих следующее.

- **Блок обработки** (**processing block**): предварительный шаг в любом приложении ML, и он направлен на подготовку данных для алгоритма ML.
- **Обучающий блок** (**learning block**): блок, реализующий ML, целью которого является изучение шаблонов на основе данных, предоставляемых блоком обработки.

Блок обработки определяет эффективность ML, поскольку необработанные входные данные часто не подходят для прямой подачи в модель. Например, входной сигнал может быть зашумленным или содержать нерелевантную и избыточную информацию для обучения модели.

Таким образом, Edge Impulse предлагает несколько готовых функций обработки, включая возможность подключения пользовательских функций.

В нашем случае мы будем использовать блок обработки для извлечения параметров MFCC, и следующие подразделы помогут нам узнать об этом больше.

Анализ звука в частотной области

В отличие от приложений для работы с изображениями, где сверточные нейросети (CNN) могут сделать извлечение характерных параметров частью процесса обучения, типичные модели распознавания речи плохо работают с необработанными аудиоданными. Следовательно, предварительное извлечение параметров является обязательным и должно быть частью блока обработки.

Мы знаем из физики, что звук – это колебание молекул воздуха, которое распространяется в виде волны. Например, если бы мы воспроизводили чистый одиночный тон, микрофон записал бы синусоидальный сигнал:

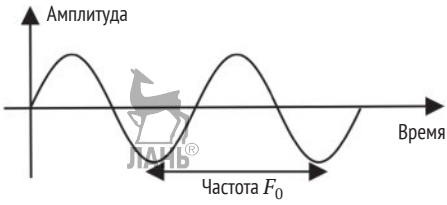


Рис. 4.9 ❖ Синусоидальная волновая форма

Хотя звуки в природе далеки от чистоты, каждый звук может быть выражен как сумма синусоидальных волн с различными *частотами* и *амплитудами*.

Поскольку частота (*frequency*) и амплитуда (*amplitude*) характеризуют синусоидальные волны, мы обычно представляем компоненты в частотной области через спектр мощности (*power spectrum*) звуковой волны:

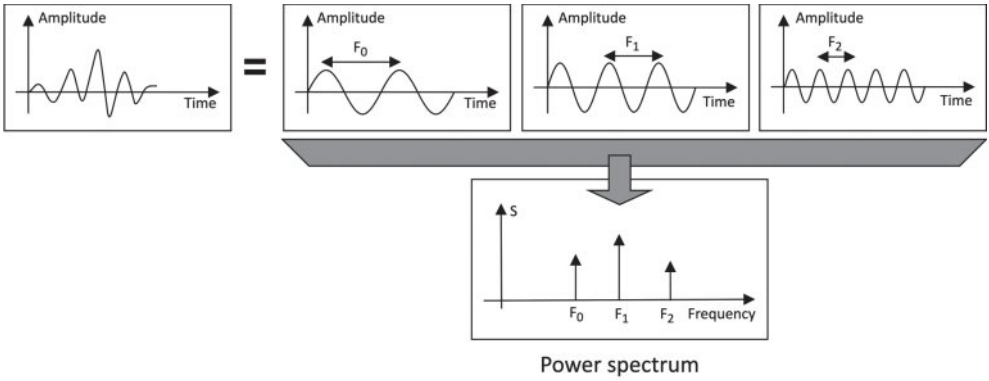


Рис. 4.10 ❖ Представление сигнала в частотной области

Спектр мощности отображает частоту на горизонтальной оси и мощность, связанную с каждым компонентом на вертикальной оси[55].

Дискретное преобразование Фурье (DFT) является необходимым математическим инструментом для разложения цифрового аудиосигнала на все составляющие его синусоидальные волны, обычно называемые компонентами.

Теперь, когда мы знакомы с частотным представлением аудиосигнала, давайте посмотрим, что можно сгенерировать в качестве входной функции для CNN.

---

[55] В данном случае термин автора «*power*» для краткости переводится как «*мощность*», хотя в спектральном представлении звуковых колебаний фигурирует более сложная величина: *спектральная плотность звуковой энергии*, или *спектральная плотность звуковой мощности*. Подробнее см. статью в «Википедии» «Акустический спектр». В реальных спектрах по оси ординат часто применяют относительные единицы – децибелы, см. далее (определения акустических терминов собраны воедино, например, в кн.: *Эфрусси М. М.* Громкоговорители и их применение. М.: Энергия, 1971. http://www.radioman-portal.ru/mrb/files/mrb0769.djvu).

Генерация Mel-спектрограммы

Спектрограмму можно рассматривать как изображение аудиосигнала, поскольку она показывает изменения спектра мощности с течением времени.

Спектрограмма получается путем разделения аудиосигнала на более мелкие сегменты и применения DFT-преобразования к каждому из них, как показано на следующем рисунке:

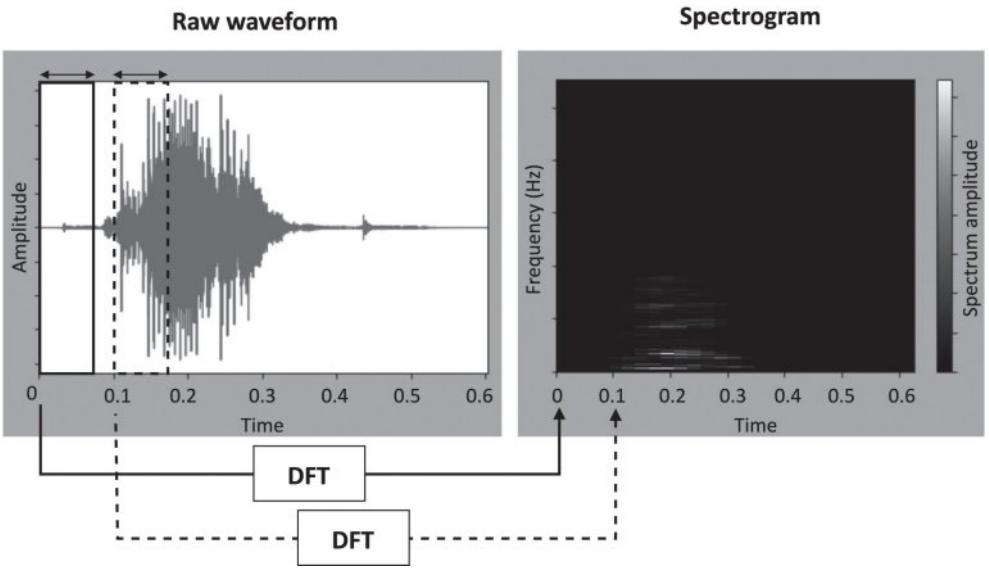


Рис. 4.11 ❖ Форма звукового сигнала (*waveform*) и его спектрограмма (*spectrogram*) для звучания слова «*red*»

На спектрограмме каждый вертикальный срез представляет спектр мощности, связанный с каждым моментом, в частности:

- горизонтальная координата представляет время;
- вертикальная координата представляет частоту;
- цвет отображает амплитуду спектра мощности, поэтому более яркий цвет подразумевает более высокую амплитуду.

Однако спектрограмма, полученная таким образом, была бы неэффективна для распознавания голосовой речи, поскольку соответствующие особенности не подчеркиваются. На самом деле, как мы можем наблюдать из этой картинки, спектрограмма темная почти во всех областях.

Поэтому спектрограмма корректируется с учетом того, что люди воспринимают частоты и громкость не линейно, а в логарифмическом масштабе. Корректировка заключается в следующем.

- **Масштабирование частоты** (герц, или Гц) до **mel**[56] с помощью банка фильтров **Mel scale filter bank**. Шкала mel изменяет частоты таким

[56] Mel (мел) – единица высоты звука, основанная на восприятии этого звука нашими органами слуха.

образом, чтобы они были различимы и воспринимались равноудаленно друг от друга. Например, если воспроизводить чистые тона от 100 до 200 Гц с шагом 1 Гц, мы могли бы отчетливо воспринимать все 100 частот. Однако, если провести тот же эксперимент на более высоких частотах (например, между 7500 и 7600 Гц), мы едва могли бы различить все тоны. Следовательно, не все частоты одинаково важны для наших ушей.

Шкала mel обычно вычисляется с использованием треугольных фильтров, наложенных друг на друга (что и представляет собой банк фильтров) в частотной области.

- **Масштабирование амплитуд** с использованием шкалы децибел (дБ): человеческий мозг воспринимает амплитуду не линейно, а логарифмически, как и в случае с частотами. Поэтому мы масштабируем амплитуды в логарифмических единицах, чтобы они были видны на спектрограмме.

Спектрограмма, полученная путем применения предыдущих преобразований, представляет собой Mel-спектрограмму или Mel-частотную энергию (**Mel-frequency energy**, **MFE**). MFE слова «*red*» с использованием 40 треугольных фильтров демонстрируется на следующем скриншоте, где теперь мы можем четко заметить интенсивность частотных составляющих:

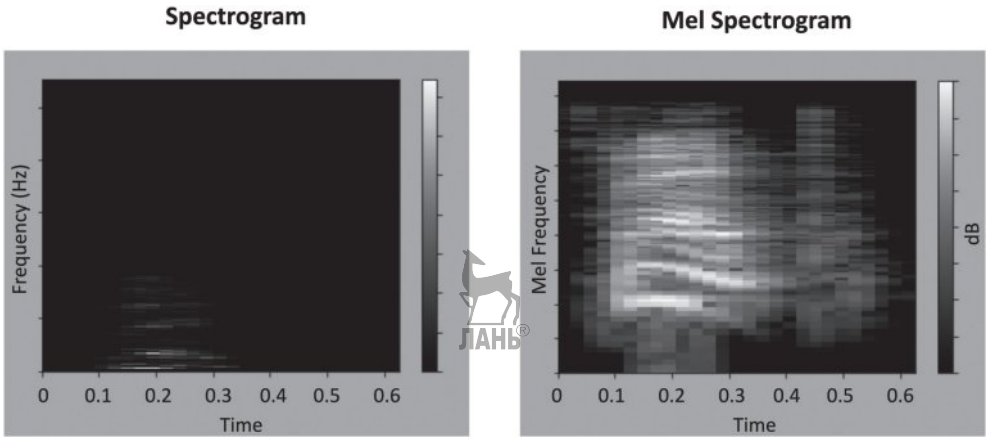


Рис. 4.12 ❖ Обычная спектрограмма и Mel-спектрограмма слова «*red*»

Хотя Mel-спектрограмма хорошо работает с моделями распознавания звука, существует также нечто еще более эффективное для распознавания человеческой речи в отношении количества входных параметров – MFCC.

Извлечение MFCC

MFCC стремится извлечь из Mel-спектрограммы меньшее количество не связанных между собой коэффициентов.

В Mel-банке фильтров используются перекрывающиеся фильтры, что обеспечивает высокую корреляцию компонентов. Если мы имеем дело с че-

ловеческой речью, мы можем декоррелировать их, применяя **дискретное косинусное преобразование** (**Discrete Cosine Transform**, **DCT**).

DCT предоставляет сжатую версию блока фильтров. Из выходных данных DCT мы можем сохранить первые 2–13 коэффициентов (так называемые *кепстральные* коэффициенты) и отбросить остальные, поскольку они не несут дополнительной информации для распознавания человеческой речи. Следовательно, результирующая спектрограмма будет иметь меньше частот, чем Mel-спектрограмма (13 против 40).

Как это делается...

Для создания нашего первого *Impulse* следует нажать на опцию **Create impulse** (*Создать Impulse*) в меню слева, как показано на следующем рисунке:

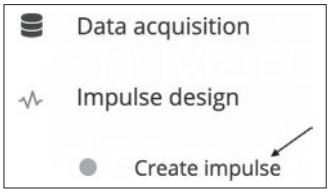


Рис. 4.13 ❖ Создание *Impulse*

В разделе Create impulse убедитесь, что для данных временных последовательностей в поле **Window size** (*Размер окна*) установлено значение 1000 мс, а в поле **Window increase** (*Расширение окна*) – значение 500 мс.

Window increase – параметр, специально предназначенный для непрерывных приложений KWS, где есть непрерывный аудиопоток, и мы не знаем, когда начинается произнесение. В этом сценарии мы разделяем аудиопоток на окна (или сегменты) равной длины и выполняем просчет ML для каждого из них. **Window size** – это длина окна во времени, в то время как **Window increase** – это временно́е расстояние между двумя последовательными сегментами, как показано на следующем рисунке:

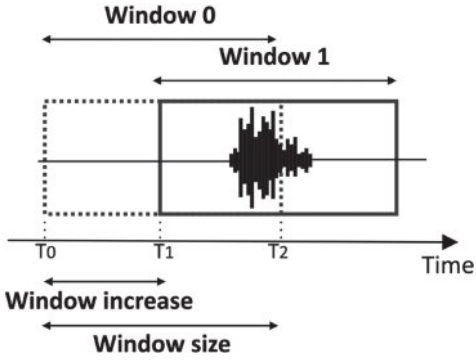


Рис. 4.14 ❖ *Window size* и *Window increase*

Значение *Window size* зависит от длины обучающей выборки (1 с) и может повлиять на результирующую достоверность модели. Напротив, значение *Window increase* не влияет на результаты обучения, но влияет на шансы получить правильное начало высказывания. Фактически меньшее значение *Window increase* подразумевает более высокую вероятность. Однако подходящее значение *Window increase* будет зависеть от задержки при работе модели.

Следующие шаги показывают, как настроить блок обработки для извлечения параметров MFCC из записанных аудиосемплов.

1. Нажмите на кнопку **Add a processing block** (*Добавить блок обработки*) и добавьте **Audio (MFCC)**.
2. Нажмите на кнопку **Add a learning block** (*Добавить обучающий блок*) и добавьте **Classification (Keras)**.
   Блок **Output features** (*Выходные параметры*) должен сообщать о семи выходных классах для распознавания (*00_red*, *01_green*, *02_blue*, *03_one*, *04_two*, *05_three* и *unknown*), как показано на следующем рисунке:

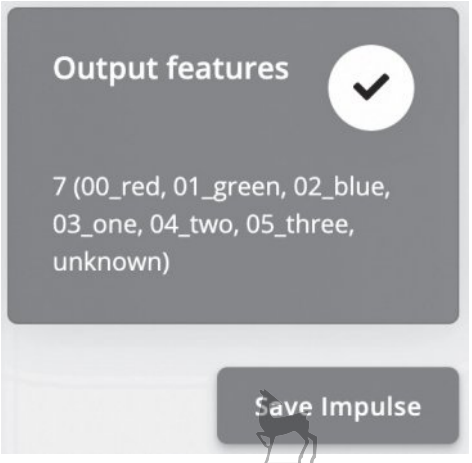


Рис. 4.15 ❖ Выходные параметры

   Сохраните *Impulse*, нажав на кнопку **Save Impulse**.
3. Нажмите на **MFCC** в категории **Impulse design**. В новом окне мы можем поиграть с параметрами, влияющими на извлечение объектов MFCC, такими как количество кепстральных коэффициентов, количество треугольных фильтров, применяемых для шкалы Mel, и т. д. Все установленные параметры MFCC сохраняются в их значениях по умолчанию.
   В нижней части страницы также есть два параметра для этапа предварительного акцентирования. Стадия предварительного акцентирования выполняется перед формированием спектрограммы для уменьшения эффекта шума за счет увеличения энергии на самых высоких частотах. Если значение величины **Coefficient** равно 0, то предварительный акцент во входном сигнале отсутствует. Параметры предварительного акцентирования сохраняются в их значениях по умолчанию.

4. Извлеките параметры MFCC из каждого обучающего образца, нажав на кнопку **Generate features** (*Сгенерировать параметры*):

Рис. 4.16 ❖ Кнопка генерации параметров

В конце этого процесса Edge Impulse в выводе консоли вернет «*Job completed*» (*Задание выполнено*).

Параметры MFCC теперь извлечены из всех записанных аудиосемплов.

Дополнительно

Как только параметры MFCC будут извлечены, мы можем использовать инструмент **Feature explorer** (*Обозреватель параметров*) для изучения сгенерированного обучающего набора данных на трехмерном (3D) графике, как показано на следующем рисунке:

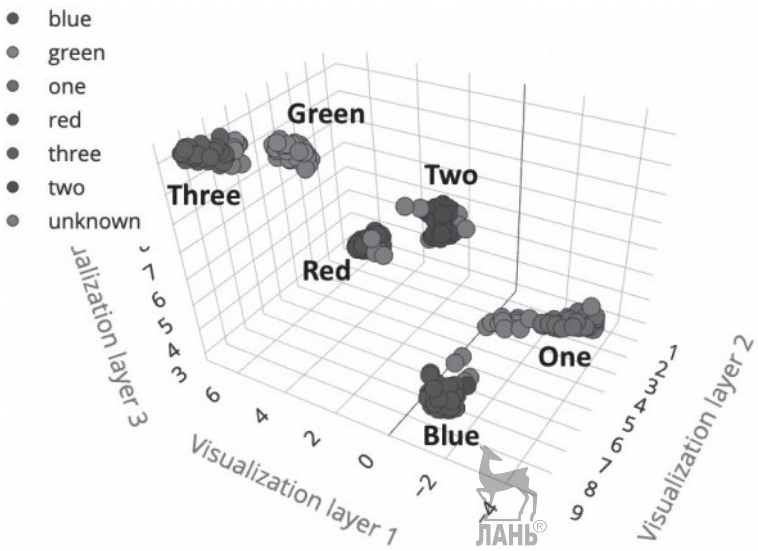


Рис. 4.17 ❖ *Feature explorer*, показывающий семь выходных классов

Из диаграммы *Feature explorer* мы должны сделать вывод, подходят ли входные параметры для нашей задачи. Если это так, то выходные классы (за исключением выходной категории *unknown*) должны быть четко разделены.

Под расположением **Feature explorer** мы находим раздел **On-device performance** (*Производительность на устройстве*), относящийся к MFCC:

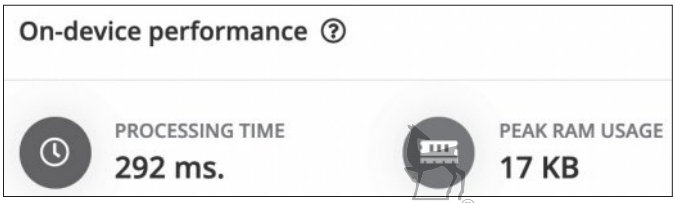


Рис. 4.18 ❖ Производительность MFCC на плате Arduino Nano 33 BLE Sense

PROCESSING TIME (*Время операции*) и **PEAK RAM USAGE** (*Пиковое использование ОЗУ*) оцениваются с учетом целевого устройства, выбранного в информационной панели в пункте **Project info** (*Информация о проекте*):

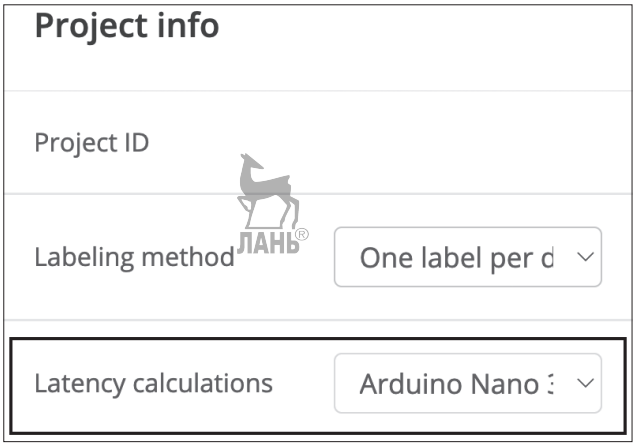


Рис. 4.19 ❖ Целевое устройство в пункте **Project info**

В **Project info** вы можете изменить целевое устройство для оценки производительности.

К сожалению, Edge Impulse пока не поддерживает Raspberry Pi Pico[57], поэтому здесь расчетная производительность будет основана только на Arduino Nano.

В информации о проекте вы можете изменить целевое устройство для оценки производительности.

[57] Edge Impulse анонсировал официальную поддержку Raspberry Pi Pico 28 февраля 2022 года.

Пример проектирования и обучения нейронной сети (NN)

В этом примере для распознавания слов мы будем использовать следующую архитектуру NN:

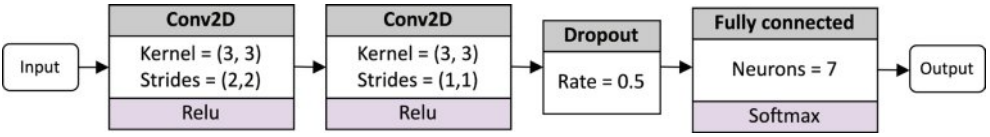


Рис. 4.20 ❖ Архитектура NN

Модель состоит из двух двумерных (2D) слоев свертки, одного отсеивающего слоя и одного полносвязного слоя, за которым следует активация *softmax*[58].

Входным сигналом сети является параметр MFCC, извлеченный из односекундного звукового семпла.

Подготовка

Чтобы подготовиться к этому примеру, нам нужно знать, как построить и обучить NN в Edge Impulse.

В зависимости от выбранного блока обучения Edge Impulse использует различные базовые фреймворки ML. Для блока обучения классификации фреймворк использует TensorFlow совместно с Keras. Построение модели может быть выполнено двумя способами.

- **Visual mode** (*Визуальный режим* – простой способ): это самый быстрый способ и выполняется через пользовательский интерфейс (UI). Edge Impulse предоставляет некоторые базовые строительные блоки NN и предустановки архитектуры, которые полезны, если вы только начали экспериментировать с глубоким обучением (DL).
- **Keras code mode** (*Режим кода Keras* – экспертный режим): если мы хотим больше контролировать сетевую архитектуру, мы можем редактировать код Keras непосредственно из веб-браузера.

Как только мы разработаем модель, то сможем запустить обучение из того же окна.

Как это делается...

Нажмите на **Neural Network (Keras)** в разделе **Impulse design** и выполните следующие шаги для построения и обучения NN, представленные на рис. 4.20.

[58] О функции активации *softmax* подробнее см.: https://habr.com/ru/post/155235/.

1. Выберите предустановленную архитектуру **2D Convolutional** и удалите отсеивающий (**Dropout**) слой между двумя слоями свертки:

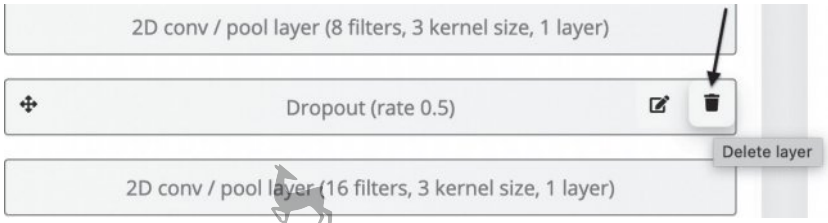


Рис. 4.21 ❖ Удаление отсеивающего слоя между двумя 2D-слоями свертки

2. Переключитесь в режим **Keras (expert)**, нажав на ⋮. В области кодирования удалите слои *MaxPooling2D*:

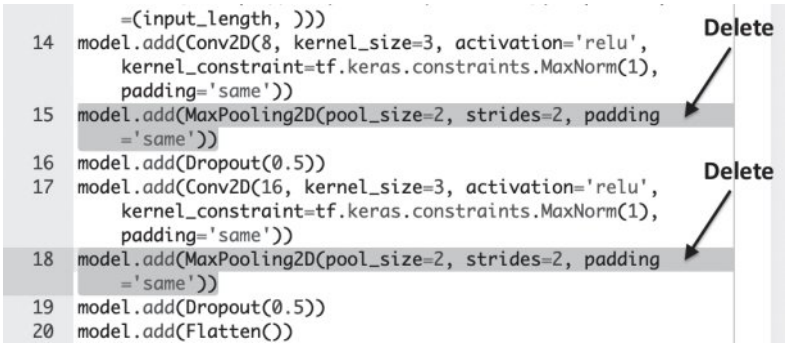


Рис. 4.22 ❖ Удаление двух слоев *MaxPooling2D* из кода Keras

Установите шаги (`strides`) первого слоя свертки равными (2,2):

```
model.add(Conv2D(8, strides=(2,2), kernel_size=3, activation='relu', kernel_
constraint=tf.keras.constraints.MaxNorm(1), padding='same'))
```

Объединение слоя – это метод прореживания, который уменьшает количество информации, распространяемой по сети, и снижает риск переобучения. Однако этот оператор может увеличить задержку при использовании оперативной памяти (RAM). В устройствах с ограниченным объемом памяти, таких как микроконтроллеры, память является ценным ресурсом, и мы должны использовать ее как можно эффективнее. Следовательно, идея состоит в том, чтобы использовать неединичные шаги (*non-unit strides*) в слоях свертки, чтобы уменьшить пространственную размерность. Этот подход, как правило, более производителен, потому что мы полностью пропускаем вычисление полносвязного слоя и можем иметь дело с более быстрыми сверточными слоями при меньшем количестве выходных элементов для обработки.

3. Запустите обучение, нажав на кнопку **Start training**:

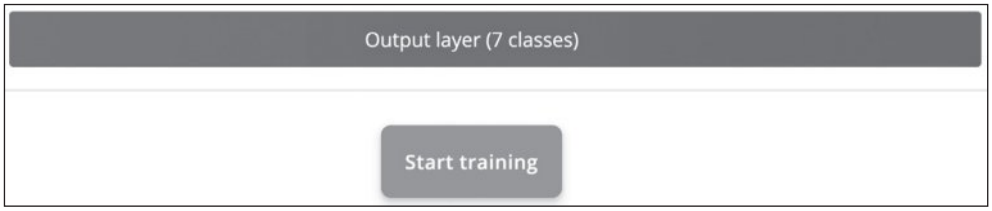


Рис. 4.23 ❖ Кнопка **Start training**

Вывод на консоль будет сообщать о достоверности и потерях в наборах данных обучения и проверки во время обучения после каждого этапа.

В конце обучения мы на той же странице можем оценить эффективность модели (достоверность и потери), матрицу ошибок и предполагаемую производительность на выбранном устройстве.

Если вы достигаете 100%-ной достоверности, это признак того, что модель, скорее всего, переобучена. Чтобы избежать этой проблемы, вы можете либо добавить больше данных в обучающий набор, либо снизить скорость обучения.

Если вас не устраивает достоверность модели, мы рекомендуем собрать больше данных и снова обучить модель.

Настройка эффективности модели с помощью EON Tuner

Разработка наиболее эффективного конвейера ML для конкретного приложения всегда является сложной задачей.

Один из способов сделать это – прибегнуть к помощи итеративных экспериментов. Например, мы можем оценить, как изменяются некоторые целевые показатели (задержка, объем памяти и достоверность) в зависимости от генерации входных функций и архитектуры модели. Однако этот процесс отнимает много времени, поскольку существует несколько комбинаций, и каждая из них должна быть протестирована и оценена. Кроме того, этот подход требует знакомства с цифровой обработкой сигналов и архитектурами NN, чтобы знать, что настраивать.

В этом примере мы будем использовать EON Tuner, чтобы найти лучший ML-конвейер для Arduino Nano.

Подготовка

EON Tuner (https://docs.edgeimpulse.com/docs/eon-tuner) – инструмент для автоматизации поиска наилучшего решения на основе ML для данной целевой

платформы. Однако это не просто инструмент **automated ML** (**AutoML**), поскольку блок обработки также является частью задачи оптимизации. Таким образом, EON Tuner является оптимизатором E2E[59] для определения наилучшей комбинации блока обработки и модели ML для заданного набора ограничений, таких как задержка, использование оперативной памяти и достоверность.

Как это делается...

Нажмите на **EON Tuner** в меню слева и следуйте следующим шагам, чтобы узнать, как найти наиболее эффективный конвейер на основе ML для нашего приложения.

1. Настройте EON Tuner, нажав на значок шестеренки в **Target area** (*целевой области*):

Рис. 4.24 ❖ Настройки EON Tuner

   Edge Impulse откроет новое окно для настройки EON Tuner. В этом окне задайте **Dataset category** (*Категория набора данных*), **Target device** (*Целевое устройство*) и **Time per inference** (*Периодичность вывода*) следующим образом:
   - **Dataset category**: **Keyword spotting** (*распознавание ключевых слов*);
   - **Target device**: **Arduino Nano 33 BLE Sense (Cortex-M4F 64MHz)**;
   - **Time per inference (ms)**: **100**.

   Поскольку Edge Impulse пока не поддерживает Raspberry Pi Pico, мы можем настроить производительность только для платы Arduino Nano 33 BLE Sense.

   Мы устанавливаем значение периодичности вывода, равное 100 мс, чтобы находить более быстрые решения, чем полученные ранее в примере «*Пример проектирования и обучения нейронной сети (NN)*».
2. Сохраните настройки EON Tuner, нажав на кнопку **Save**.
3. Запустите EON Tuner, нажав на кнопку **Start EON Tuner**. Процесс может занять от нескольких минут до 6 ч, в зависимости от размера набора данных. Инструмент покажет индикатор выполнения и сообщит об обнаруженных архитектурах в том же окне, как показано на следующем рисунке:

[59] Сквозное тестирование E2E (End-to-end) – проверка всего программного обеспечения от начала до конца на предмет зависимостей, целостности данных, связи с другими системами и интерфейсами в целях успешного выполнения полного производственного сценария.

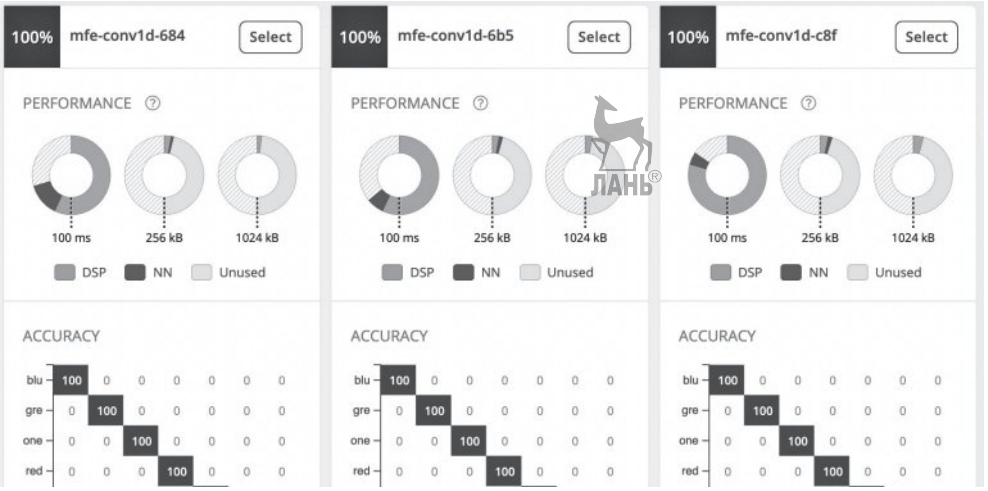


Рис. 4.25 ❖ EON Tuner демонстрирует матрицу ошибки для каждого предлагаемого решения ML

Как только EON Tuner завершит этап поиска, у вас будет коллекция решений на основе ML (обработка + модель ML) на выбор.

4. Выберите архитектуру с более высокой достоверностью и меньшим расширением временно́го окна, нажав на кнопку **Select**. Выбранная нами архитектура имеет расширение окна на 250 мс, использует MFE в качестве функции ввода и 1D-слои свертки.

   Как вы можете заметить, функция ввода не является MFCC. EON Tuner предлагает альтернативный блок обработки, поскольку он учитывает задержку всего конвейера ML, а не только просчет модели. Верно, что MFE может замедлить процесс просчета модели, поскольку он возвращает спектрограмму с большим количеством функций, чем MFCC. Однако MFE работает значительно быстрее, чем MFCC, поскольку не требует извлечения компонентов дискретного косинусного преобразования DCT.

   Как только вы выберете архитектуру, Edge Impulse попросит вас обновить основную модель. Нажмите на **Yes**, чтобы переопределить архитектуру, подготовленную в предыдущем разделе «*Пример проектирования и обучения нейронной сети (NN)*». Появится всплывающее окно, подтверждающее, что основная модель была обновлена.

В конце нажмите на **Retrain model** (*Повторное обучение модели*) на левой боковой панели и нажмите на **Train model** (*Обучение модели*), чтобы снова обучить сеть.

Классификация в реальном времени с помощью смартфона

Когда мы говорим о тестировании модели, мы обычно имеем в виду оценку обученной модели на тестовом наборе данных. Однако тестирование модели в Edge Impulse – это нечто большее.

В этом примере мы узнаем, как протестировать эффективность модели на тестовом наборе, а также покажем способ выполнения классификации в реальном времени с помощью смартфона.

Подготовка

Единственное, что нам нужно знать, прежде чем приступать к этому примеру, – как мы можем оценить эффективность модели в Edge Impulse.

В Edge Impulse мы можем оценить обученную модель двумя способами:

- **тестированием модели**: мы оцениваем достоверность, используя тестовый набор данных. Тестовый набор данных обеспечивает объективную оценку эффективности модели, поскольку входящие в него семплы не используются прямо или косвенно во время обучения;
- **классификацией в реальном времени**: это уникальная функция Edge Impulse, благодаря которой мы можем записывать новые семплы либо со смартфона, либо с поддерживаемого устройства (например, Arduino Nano).

Подход классификации в реальном времени выигрывает в сравнении с тестированием обученной модели перед обязательным развертыванием приложения на целевой платформе.

Как это делается...

Чтобы оценить эффективность модели с помощью тестового набора данных и инструмента классификации в реальном времени, выполните следующие шаги.

1. Нажмите на **Model testing** (*Тестирование модели*) на левой панели и нажмите на **Classify all** (*Классифицировать все*).
   Edge Impulse позаботится об извлечении MFE из тестового набора, запуске обученной модели и представлении отчетов об эффективности в виде матрицы ошибок.
2. Нажмите на **Live classification** (*Классификация в реальном времени*) на левой панели и убедитесь, что смартфон указан в списке устройств:

Рис. 4.26 ❖ Список устройств, показывающий, что мобильный телефон подключен к Edge Impulse

Выберите **Microphone** из выпадающего списка **Sensor** в разделе **Live classification** и установите значение длительности выборки (мс) равным 10 000. Сохраните для **Frequency** (*Частота*) значение по умолчанию (16 000 Гц).

3. Нажмите **Start sampling** (*Начать выборку*), а затем предоставьте доступ к микрофону на вашем телефоне, нажав **Give access to the Microphone**. Запишите любое из наших шести высказываний (`red`, `green`, `blue`, `one`, `two` или `three`). Образец аудио будет загружен на Edge Impulse, как только вы завершите запись.

   На этом этапе Edge Impulse разделит запись на образцы продолжительностью 1 с и протестирует обученную модель на каждой из них. Результаты классификации будут представлены на той же странице в следующих формах.

   - **Общая сводка**: здесь сообщается о количестве распознаваний для каждой категории выходных данных:

CATEGORY	COUNT
blue	1
green	1
one	1
red	2
three	1
two	1
unknown	29
uncertain	1

Рис. 4.27 ❖ Общая сводка, сообщающая о количестве распознаваний для каждого ключевого слова

- **Детальный анализ**: здесь сообщается о вероятности классов для каждой временно́й метки, как показано на следующем рисунке:

TIMESTAMP	BLUE	GREEN	ONE	RED	THREE	TWO	UNKNOWN
1000	0	0	0	0	0	0	1.00
1250	0.94	0	0.05	0	0	0	0
1500	0.02	0	0	0	0	0	0.98
1750	0	0	0	0	0	0	1.00
2000	0	0	0	0	0	0	1.00

Рис. 4.28 ❖ Детальный анализ, показывающий вероятность классов для каждой временной метки

Если вы нажмете на какую-то запись в таблице, Edge Impulse покажет соответствующую форму звукового сигнала в отдельном окне, как показано на рис. 4.28.

Классификация в реальном времени с помощью Arduino Nano

Если вам показалась полезной классификация в реальном времени с помощью смартфона, еще более полезной окажется классификация в реальном времени с помощью Arduino Nano.

Этот пример покажет, как подключить Arduino Nano к Edge Impulse для выполнения классификации в реальном времени непосредственно с нашей целевой платформы.

Подготовка

Чтобы иметь больше уверенности в достоверности классификации, тестирование эффективности модели с помощью датчика, используемого в конечном приложении, является хорошей практикой. Благодаря Edge Impulse можно выполнить классификацию в реальном времени на Arduino Nano с помощью нескольких простых шагов. Их описание вы можете найти также по следующей ссылке: https://docs.edgeimpulse.com/docs/arduino-nano-33-ble-sense.

Как это делается...

Для классификации в реальном времени с помощью встроенного микрофона на Arduino Nano требуется установка дополнительного программного обе-

спечения на вашем компьютере. Различные инструменты работают в Linux, macOS и Windows, и перечислены здесь:

- **интерфейс командной строки** Edge Impulse (**command-line interface**, **CLI**): https://docs.edgeimpulse.com/docs/cli-installation;
- **интерфейс Arduino CLI**: https://arduino.github.io/arduino-cli/0.19/.

После установки зависимостей выполните следующие действия, чтобы подключить платформу Arduino Nano к Edge Impulse.

1. Запустите `arduino-cli core install arduino:mbed_nano` из приглашения командной строки (Command Prompt) или через терминал.
2. Подключите плату Arduino Nano к вашему компьютеру и дважды нажмите на ней кнопку сброса, чтобы перевести устройство в режим загрузчика.
   Встроенный светодиод должен начать мигать, чтобы подтвердить, что платформа находится в режиме загрузчика.
3. Загрузите прошивку Edge Impulse для Arduino Nano с https://cdn.edgeimpulse.com/firmware/arduino-nano-33-ble-sense.zip и распакуйте файл. Прошивка потребуется для отправки звуковых семплов с Arduino Nano на Edge Impulse.
4. В распакованной папке запустите flash-скрипт для загрузки прошивки на Arduino Nano. Вы должны использовать скрипт соответственно с вашей операционной системой (ОС) – например, *flash_linux.sh* для Linux. Как только прошивка будет загружена на Arduino Nano, вы можете нажать кнопку сброса (**Reset**), чтобы запустить программу.
5. Запустите `edge-impulse-daemon` из командной строки или терминала. Мастер попросит вас войти в систему и выбрать проект Edge Impulse, над которым вы работаете.

Теперь Arduino Nano должен быть подключен к Edge Impulse. Вы можете проверить подключение, нажав на **Devices** (*Устройства*) на левой боковой панели, как показано на следующем рисунке:

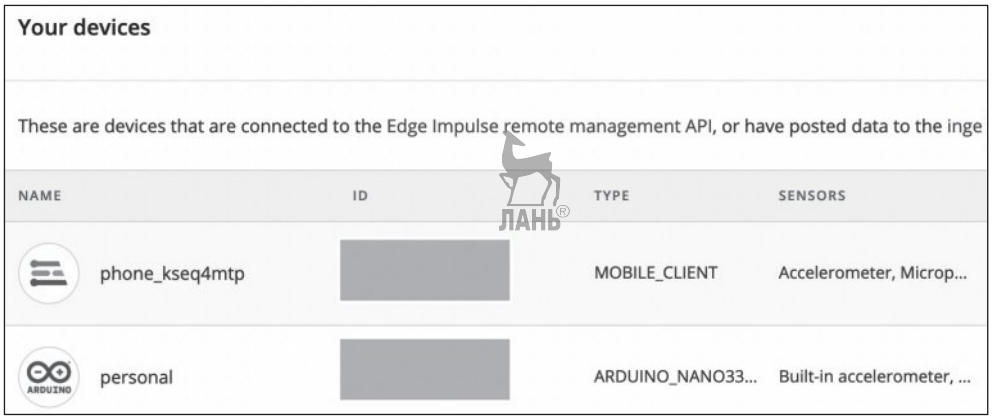


Рис. 4.29 ❖ Список устройств, подключенных к Edge Impulse

Как вы можете видеть на предыдущем скриншоте, Arduino Nano (*personal*) указан в разделе **Your devices** (*Ваши устройства*).

Теперь перейдите к классификации в реальном времени и выберите **Arduino Nano 33 BLE Sense board** из выпадающего списка устройств. Теперь вы можете записать звуковые семплы с помощью Arduino Nano и проверить, работает ли модель.

Если вы обнаружите, что модель работает не так, как ожидалось, мы рекомендуем добавить звуковые семплы, записанные с помощью микрофона Arduino Nano, в обучающий набор данных. Для этого нажмите на **Data acquisition** (*сбор данных*) и запишите новые данные с помощью устройства Arduino Nano (на правой боковой панели).

Непрерывное распознавание на Arduino Nano

Как вы можете догадаться, развертывание приложений на Arduino Nano и Raspberry Pi Pico отличается, поскольку устройства имеют разные аппаратные возможности. В этом примере мы покажем, как реализовать непрерывную работу приложения распознавания ключевых слов на Arduino Nano.

Скетч Arduino *07_kws_arduino_nano_ble33_sense.ino*, содержащий код, описываемый в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter04/ArduinoSketches/07_kws_arduino_nano_ble33_sense.ino.

Подготовка

Приложение на Arduino Nano будет основано на *nano_ble33_sense_microphone_continuous.cpp* – примере, реализующем приложение KWS в реальном времени, предоставленном Edge Impulse. Прежде чем изменять код, мы должны изучить, как работает этот пример.

Изучение примера приложения KWS в реальном времени

Приложение KWS в режиме реального времени – например, то, которое используется в «умных помощниках» – должно захватывать и обрабатывать все фрагменты аудиопотока, чтобы никогда не пропустить ни одного события. Таким образом, приложение должно записывать аудио и запускать расчет результатов распознавания одновременно, чтобы мы не пропустили какую-либо информацию.

На микроконтроллере параллельные задачи могут выполняться двумя способами:

- с помощью операционной системы реального времени (RTOS). В этом случае мы можем использовать два потока для захвата и обработки аудиоданных;
- с выделенным периферийным устройством, таким как прямой доступ к памяти (DMA), подключенным к АЦП. DMA позволяет передавать данные без вмешательства в основную программу, запущенную на процессоре.

В этом примере мы не будем касаться непосредственно этого аспекта. На самом деле в примере *nano_ble33_sense_microphone_continuous.cpp* уже представлено приложение, в котором аудиозапись и просчет выполняются одновременно с помощью механизма двойной буферизации. Двойная буферизация использует два буфера фиксированного размера, предназначенных для следующего:

- один из буферов предназначен для задачи выборки звука,
- второй буфер предназначен для задачи обработки (извлечение параметров и просчет модели ML).

В каждом буфере хранится количество звуковых семплов, необходимых для записи с увеличением временного окна.

Таким образом, размер буфера может быть рассчитан по следующей формуле:

$$Buffer_{size} = SF\ (\text{Гц}) \cdot WI\ (\text{с}).$$

Эта формула представляет собой произведение следующих двух величин:

- *SF* (Гц): частота дискретизации в Гц (например, 16 килогерц (кГц) = 16 000 Гц),
- *WI* (с): расширение окна в секундах (например, 250 мс = 0.250 с).

Например, если мы выберем аудиосигнал с частотой 16 кГц и расширение окна составит 250 мс, каждый буфер будет иметь емкость 4000 выборок.

Эти два буфера непрерывно переключаются между задачами записи (*recording*) и обработки (*processing*), и следующая блок-схема наглядно показывает, как именно:

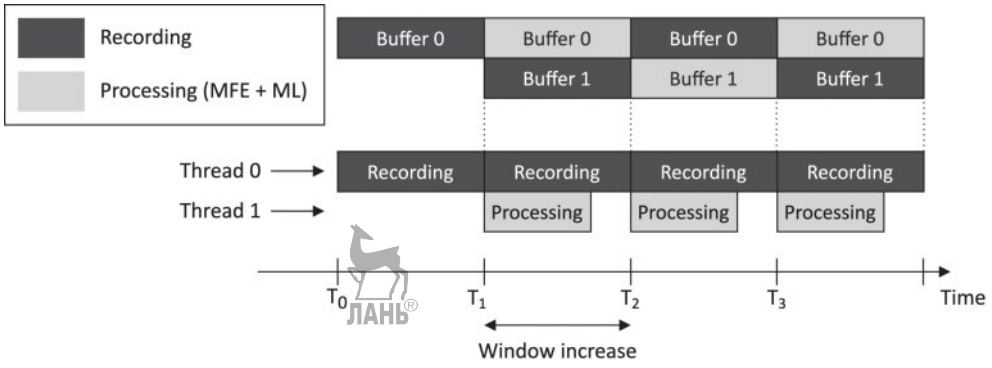


Рис. 4.30 ❖ Задачи записи и обработки, выполняемые одновременно

Из представленной схемы мы можем заключить следующее.

1. Задача записи начинает заполнять буфер **Buffer 0** в момент $t = T_0$.
2. В момент $t = T_1$ **Buffer 0** заполнен. Следовательно, задача обработки может приступить к расчетам, используя данные из **Buffer 0**. Тем временем задача записи продолжает захват аудиоданных в фоновом режиме, используя буфер **Buffer 1**.

3. В момент $t = T_2$ **Buffer 1** заполнен. За это время задача обработки должна была завершить предыдущее вычисление, прежде чем начинать новое.

Сохранение как можно более короткого времени расширения окна дает следующие преимущества:

- увеличивает вероятность получения правильного начала высказывания;
- сокращает время вычислений для извлечения параметров, поскольку вычисления идут только во время, на которое расширено окно.

Однако время расширения окна должно быть достаточно длительным, чтобы гарантировать, что задача обработки может быть завершена за это время.

На этом этапе у вас может возникнуть один вопрос: если у нас есть расширение окна на 250 мс, а модель ожидает односекундный звуковой семпл, как двойные буферы могут передавать данные нейросети NN?

Двойные буферы – это не вход NN, а вход для дополнительного буфера, содержащего семплы односекундного аудио. В этом буфере хранятся данные по принципу «первый вошел – первый вышел» (FIFO). Буфер предоставляет фактические входные данные для модели ML по принципу, показаному на следующей схеме:

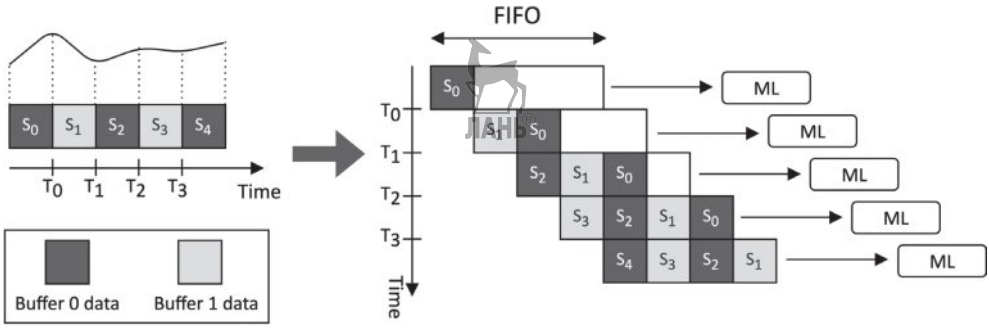


Рис. 4.31 ❖ Использование FIFO-буфера для загрузки модели NN

Поэтому каждый раз, когда мы запускаем новую задачу обработки, отобранные данные копируются в очередь FIFO перед запуском расчета модели.

Как это делается...

С помощью следующих шагов мы внесем некоторые изменения в файл *nano_ble33_sense_microphone_continuous.cpp* для голосового управления встроенными RGB-светодиодами на Arduino Нано.

1. В Edge Impulse нажмите на **Deployment** (*Развертывание*) в меню слева и выберите **Arduino Library** (*Библиотека Arduino*) из опций **Create library** (*Создание библиотеки*), как показано на следующем рисунке:

Рис. 4.32 ❖ Выбор параметров библиотеки в Edge Impulse

Затем нажмите на кнопку **Build** (*Создать*) в нижней части страницы и сохраните ZIP-файл на своем компьютере. ZIP-файл представляет собой библиотеку Arduino, содержащую приложение KWS, процедуры извлечения параметров (MFCC и MFE) и несколько готовых к использованию примеров для платы Arduino Nano 33 BLE Sense.

2. Откройте среду разработки Arduino (IDE) и импортируйте библиотеку, созданную Edge Impulse. Для этого перейдите на вкладку **Libraries** (*Библиотеки*) на левой панели, а затем нажмите кнопку **Import**, как показано на рис. 4.33.

Рис. 4.33 ❖ Импорт библиотеки в веб-редакторе Arduino

После импорта откройте пример *nano_ble33_sense_microphone_continuous* из папки *Examples | FROM LIBRARIES | <name_of_your_project>_INFERENCING*.

В нашем случае *<name_of_your_project>* означает `VOICE_CONTROLLING_LEDS`, что соответствует имени, данному нашему проекту в Edge Impulse.

В файле макрос `EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW` определяет расширение окна в терминах количества кадров, обрабатываемых в окне модели. Мы можем оставить его на значении по умолчанию.

3. Объявите и инициализируйте глобальный массив объектов `mbed::DigitalOut` для управления встроенными RGB-светодиодами:

```
mbed::DigitalOut rgb[] = {p24, p16, p6};
#define ON 0
#define OFF 1
```

Для инициализации `mbed::DigitalOut` требуются имена выводов, к которым подключены RGB-светодиоды. Названия выводов можно найти на схеме платы Arduino Nano 33 BLE Sense (https://content.arduino.cc/assets/NANO33BLE_V2.0_sch.pdf):

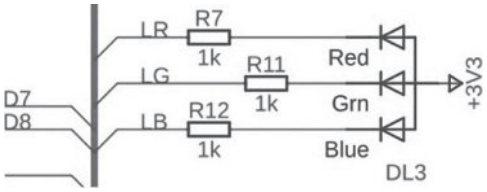


Рис. 4.34 ❖ Встроенные RGB-светодиоды подключены к схеме приемника тока

RGB-светодиоды, обозначенные LR, LG и LB, управляются схемой с втекающим током[60] и подключаются к выводам P0.24, P0.16 и P0.06 (рис. 4.35)

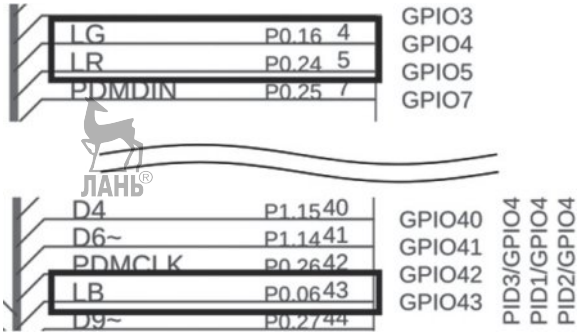


Рис. 4.35 ❖ RGB-светодиоды подключены к P0.24, P0.16 и P0.06

Следовательно, контакт ввода/вывода общего назначения (GPIO) должен подавать напряжение 0 В (LOW), чтобы включить светодиод. Чтобы избежать использования числовых значений, для включения и выключения светодиодов мы используем C-определения `#define ON 0` и `#define OFF 1`.

4. Определите целочисленную глобальную переменную (`current_color`) для отслеживания последнего распознанного цвета. Инициализируйте ее равной 0 (*red*):

```
size_t current_color = 0;
```

[60] Смотрите также рис. 2.18 и примечание в сноске на стр. 68.

5. Инициализируйте встроенные RGB-светодиоды в функции `setup()`, включив тот, для которого задан `current_color`:

```
rgb[0] = OFF; rgb[1] = OFF; rgb[2] = OFF; rgb[current_color] = ON;
```

6. В функции `loop()` установите значение `false` для флага скользящей средней (**moving average**, **MA**) в функции `run_classifier_continuous()`:

```
run_classifier_continuous(&signal, &result, debug_nn, false);
```

   Функция `run_classifier_continuous()` отвечает за запуск просчета модели. MA отключается путем передачи значения false после параметра debug_nn. Однако почему мы отключаем эту функциональность?
   MA – это эффективный метод для фильтрации ложных обнаружений при небольшом увеличении окна. Например, рассмотрим слово «*синяя птица*». Это слово содержит синий цвет, но это не то высказывание, которое мы хотим распознать. Однако при выполнении непрерывного расчета с небольшим расширением окна есть преимущество обработки небольших фрагментов звучания за раз. Таким образом, слово «синяя» может быть обнаружено с высокой степенью уверенности в одном фрагменте, но не в других. Цель MA – усреднить результаты классификаций с течением времени, чтобы избежать таких ложных обнаружений.
   Как мы можем догадаться, выходной класс должен иметь несколько классификаций с высоким рейтингом при использовании MA. Следовательно, что произойдет, если расширение окна будет значительным? Когда расширение окна значительно (например, больше 100 мс), мы обрабатываем меньше сегментов в секунду, и тогда скользящее среднее может отфильтровать все классификации. Поскольку наше расширение окна будет составлять от 250 до 500 мс (в зависимости от выбранной архитектуры ML), мы рекомендуем вам отключить его, чтобы избежать фильтрации правильных результатов.
7. Удалите код после `run_classifier_continuous()` до конца функции `loop()`.
8. В функции `loop()` после `run_classifier_continuous()` впишите код для вывода класса, имеющего более высокую вероятность:

```
size_t ix_max = 0;
float pb_max = 0.0f;
for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++)
{
  if(result.classification[ix].value > pb_max) {
    ix_max = ix;
    pb_max = result.classification[ix].value;
  }
}
```

   В этом фрагменте кода мы перебираем все выходные классы (`EI_CLASSIFIER_LABEL_COUNT`) и сохраняем индекс (`ix`) с максимальным значением классификации (`result.classification[ix].value`).

Число EI_CLASSIFIER_LABEL_COUNT – это C-определение (define), предоставляемое Edge Impulse и равное количеству выходных классов.

9. Если вероятность выходного класса (pb_max) выше фиксированного порога (например, 0.5) и класс не является неизвестным (unknown), проверьте, равен ли он одному из цветов. Если метка класса оказалась цветом, отличным от последнего распознанного, отключите current_color и включите new_color:

```
size_t new_color = ix_max;
if (new_color != current_color) {
  rgb[current_color] = OFF;
  rgb[new_color] = ON;
  current_color = new_color;
}
```

Если класс представляет собой число, мигните светодиодом current_color определенное количество раз:

```
const size_t num_blinks = ix_max0 - 2;
for(size_t i = 0; i < num_blinks; ++i) {
  rgb[current_color] = OFF;
  delay(1000);
  rgb[current_color] = ON;
  delay(1000);
}
```

Скомпилируйте и загрузите скетч в Arduino Nano. Теперь вы должны иметь возможность голосом изменять цвет светодиода или заставлять его мигать.

Схема для голосового управления светодиодами на Raspberry Pi Pico

Raspberry Pi Pico не имеет на борту ни микрофона, ни RGB-светодиодов для создания KWS-приложения. Следовательно, голосовое управление RGB-светодиодами на этой платформе требует создания электронной схемы.

Цель этого примера – подготовить схему с Raspberry Pi Pico, RGB-светодиодами, кнопкой и электретным микрофоном с усилителем MAX9814.

Подготовка

Приложение, которое мы рассмотрим для Raspberry Pi Pico, не основано на непрерывном выводе. Здесь мы собираемся использовать кнопку, чтобы начать аудиозапись продолжительностью 1 с, а затем запустить просчет модели для распознавания высказывания. Произнесенное слово, в свою очередь, будет использоваться для управления состоянием RGB-светодиодов.

В следующем подразделе мы узнаем больше об использовании электретного микрофона с усилителем MAX9814.

Представляем модуль электретного микрофона с усилителем MAX9814

Микрофон, задействованный в этом примере, представляет собой недорогой модуль электретного микрофона с усилителем MAX9814. Вы можете приобрести микрофон в магазине Pimoroni: https://shop.pimoroni.com/products/adafruit-electret-microphone-amplifier-max9814-w-auto-gain-control?variant=568716869[61].

Сигнал, поступающий с микрофона, очень мал и требует усиления для адекватного захвата и анализа.

По этой причине в нашем модуле микрофон соединен с усилителем **MAX9814** со встроенной автоматической регулировкой усиления (АРУ). АРУ позволяет записывать речь в условиях, где громкость звука и уровень фонового шума непредсказуемо меняются. MAX9814 автоматически регулирует коэффициент усиления, чтобы голос всегда был различим.

Усилитель требует напряжения питания от 2.7 до 5.5 В. При питании 3.3 В он выдает на выходе максимальное напряжение от пика к пику (2*V*pp), равное 2.45 В при постоянном смещении (DC) 1.25 В.

*V*pp – это полная высота формы сигнала.

Таким образом, устройство может быть подключено к АЦП Raspberry Pi Pico, принимающему входные сигналы в диапазоне от 0 до 3.3 В.

Как показано на следующем рисунке, микрофонный модуль имеет пять отверстий в нижней части для установки разъема:

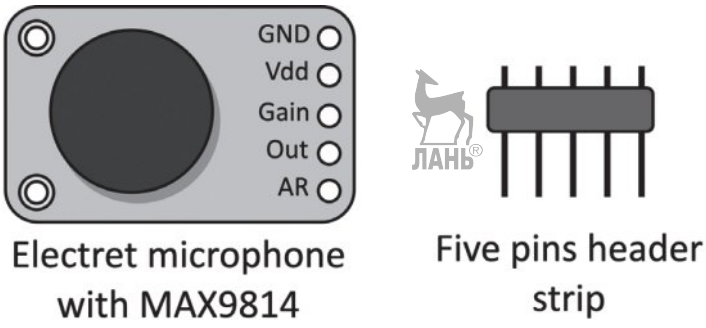


Рис. 4.36 ❖ Модуль электретного микрофона с MAX9814

[61] В России можно заказать оригинальный модуль (например, https://www.chipdip.ru/product1/8002217926) или приобрести китайский аналог в отечественных интернет-магазинах (например, https://voltiq.ru/shop/max-9814-electret-microphone/), а также на AliExpress.

Соединительный разъем необходим для монтажа устройства на макетной плате, и обычно его необходимо припаять.

Если вы не знакомы с пайкой, мы рекомендуем прочитать руководство по установке разъема:

https://learn.adafruit.com/adafruit-agc-electret-microphone-amplifier-max9814/assembly.

В следующем подразделе вы узнаете, как подключить это устройство к Raspberry Pi Pico.

Подключение микрофона к АЦП Raspberry Pi Pico

Колебания напряжения, создаваемые микрофоном, требуют преобразования в цифровой формат. Микроконтроллер RP2040 на Raspberry Pi Pico имеет четыре АЦП для выполнения такого преобразования, но только три из них могут использоваться для внешних входов, поскольку один напрямую подключен к внутреннему датчику температуры.

Номера выводов, зарезервированные для АЦП, показаны здесь:

Таблица 4.1. Выводы АЦП

Обозначение АЦП	**ADC0**	**ADC1**	**ADC2**
Вывод	GP26	GP27	GP28

Ожидаемый диапазон напряжений для АЦП на Raspberry Pi Pico должен находиться в пределах от 0 до 3.3 В, что идеально подходит для сигнала, поступающего от модуля электретного микрофона.

Как это делается...

Давайте начнем с размещения Raspberry Pi Pico на макетной плате. Необходимо установить плату вертикально, как мы это делали в главе 2 «*Прототипирование на микроконтроллерах*».

После установки устройства на макетную плату убедитесь, что USB-кабель не подключен к источнику питания, и выполните следующие действия по сборке схемы.

1. Разместите RGB-светодиоды на макетной плате (рис. 4.37).
   Подключите резистор к светодиодам последовательно, соединив один из двух выводов с катодом светодиода, другой – с GND. В табл. 4.2 указано, какой резистор использовать с каждым светодиодом.

Таблица 4.2. Резисторы, используемые с RGB-светодиодами[62]

Светодиод (LED)	Красный (red)	Зеленый (green)	Синий (blue)
Сопротивление (Ом)	220	220	100

Сопротивления были выбраны таким образом, чтобы гарантировать по крайней мере ~ 3 мА прямого тока через каждый светодиод.

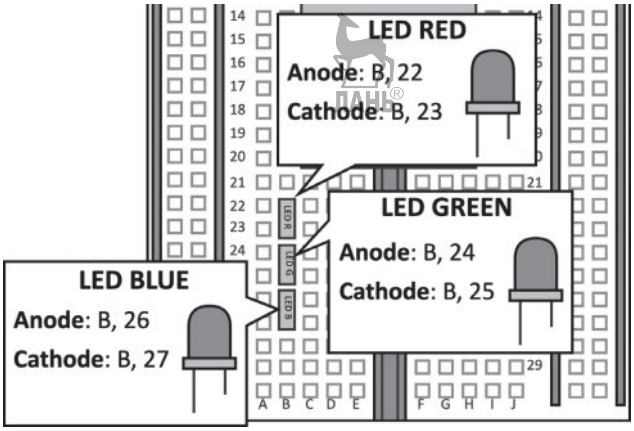


Рис. 4.37 ❖ RGB-светодиоды на макетной плате[63]

На рис. 4.38 показано последовательное подключение резисторов к светодиодам.

Следует подключить вывод GND микроконтроллера к отрицательной шине питания платы и вставить в нее вторые выводы резисторов.

[62] Указанные величины сопротивления резисторов ориентированы на сигнальные светодиоды обычной яркости (15–40 мкд). Отметим, что в Arduino Nano 33 BLE Sense установлен RGB-светодиод повышенной яркости, для которого сопротивления токоограничивающих резисторов существенно больше (см. рис. 4.34); см. также замечание в сноске на стр. 68.

[63] Строго говоря, на этой схеме и далее речь идет не о трехцветных RGB-светодиодах, а об обычных отдельных красном, зеленом и синем светодиодах. Разница между ними заключается в конструкции и подключении: RGB-светодиод – это три цветных в едином корпусе с отдельными выводами; именно RGB-светодиод с общим анодом установлен в Arduino Nano 33 BLE Sense (см. рис. 4.34).

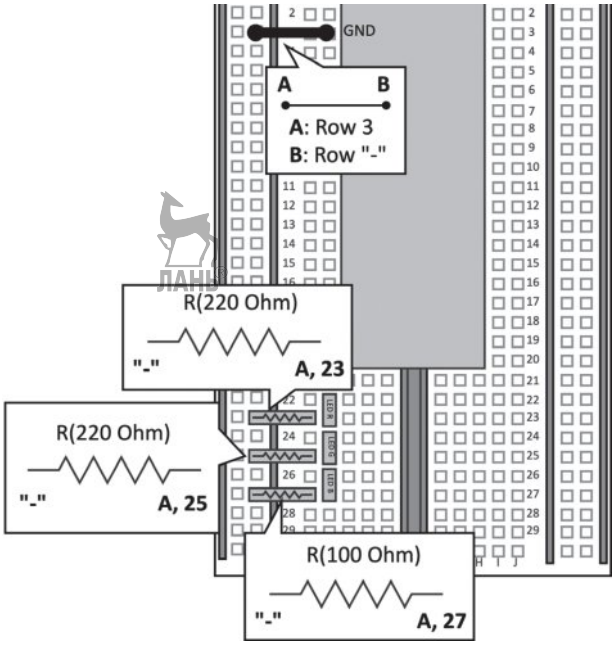


Рис. 4.38 ❖ Резисторы,
подключенные последовательно к светодиодам

2. Подсоедините аноды светодиодов RGB к выводам GPIO:

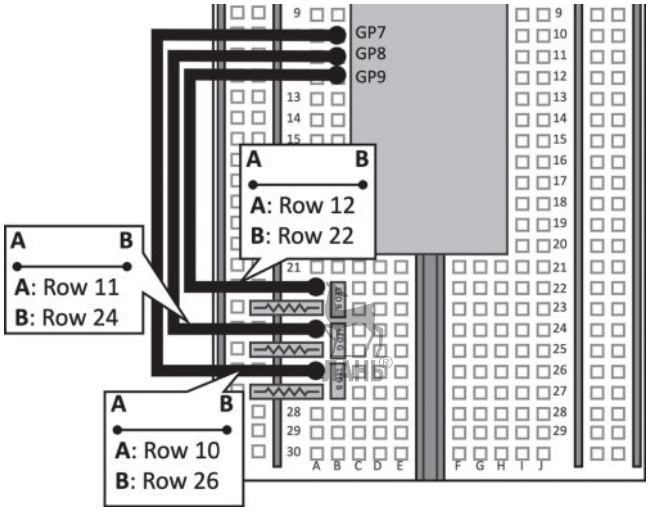


Рис. 4.39 ❖ Светодиоды, подключенные к выводам GPIO

Как показано на этой схеме, используемые для управления светодиодами выводы GPIO – это GP9 (красный), GP8 (зеленый) и GP7 (синий). Поскольку резисторы подключены к GND, светодиоды питаются от схемы источника тока. Следовательно, чтобы включить их, мы должны подать на выводы GP9, GP8 или GP7 высокий уровень (`HIGH`).

3. Поместите кнопку на макетную плату:

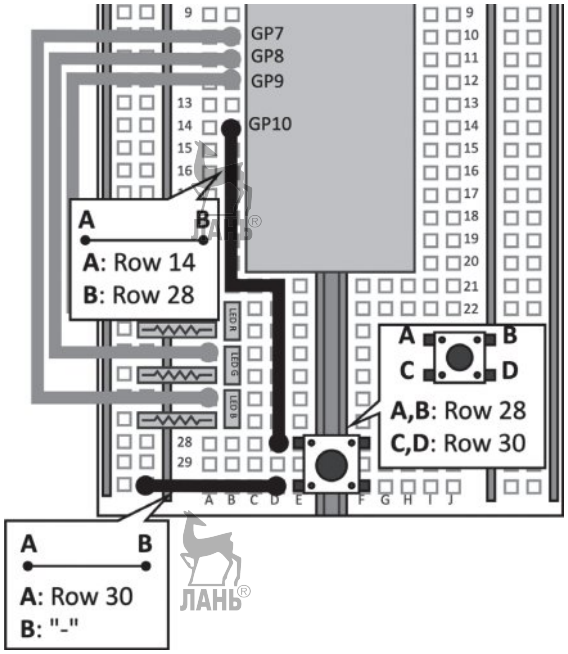


Рис. 4.40 ❖ Кнопка, подключенная к GP10 и GND

Для подключения кнопки используется вывод GP10.

Поскольку для подключения все равно потребуются соединительные провода-перемычки, мы размещаем кнопку в нижней части макетной платы, чтобы было достаточно места для нажатия.

4. Поместите на макетную плату модуль электретного микрофона (рис. 4.41).

   Вывод АЦП, используемый для подключения модуля, – GP26. Из пяти контактов на микрофонном модуле нам нужно подключить только три из них:

   - **Vdd** (3.3 В): напряжение питания усилителя. Vdd должно быть стабилизированным и равным напряжению питания АЦП. Эти условия необходимы для уменьшения шума в аналоговом сигнале, поступающем с микрофона. Поэтому к Vdd должен быть подключен вывод опорного напряжения АЦП ADC_VREF[64];

[64] Вывод ADC_VREF в контроллере RP2040 является выводом опорного напряжения и одновременно питания АЦП.

- **GND**: общий провод усилителя должен совпадать с общим проводом АЦП. Поскольку аналоговые сигналы более чувствительны к шуму, чем цифровые, Raspberry Pi Pico предлагает специальный вывод общего провода для АЦП: аналоговую землю (AGND). AGND здесь должен быть подключен к общей земле GND;
- **Out**: усиленный аналоговый сигнал, поступающий от микрофонного модуля. Его следует подключить к GP26 для преобразования с помощью периферийного устройства ADC0.

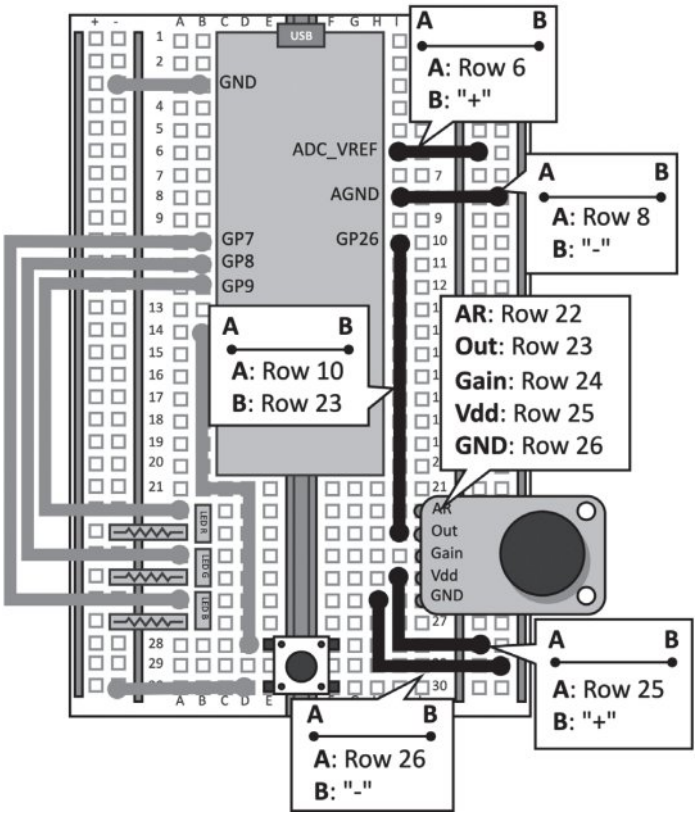


Рис. 4.41 ❖ Модуль микрофона, установленный на макетной плате

В следующей таблице приведены сведения о соединениях, которые необходимо выполнить между Raspberry Pi Pico и модулем микрофона с усилителем MAX9814:

Таблица 4.3. Подключения модуля электретного микрофона

Вывод микрофонного модуля	Vdd	GND	Out
Вывод Raspberry Pi Pico	ADC_VREF	AGND	GP26

Оставшиеся два вывода микрофонного модуля (**G** и **AR**) используются для настройки усиления (*gain*) и регулировки времени срабатывания/восстановления (*attack and release*) цепи автоматической регулировки усиления (АРУ). Эти настройки не требуются для данного примера, но вы можете узнать больше в спецификации MAX9814 (https://robotchip.ru/download/datasheet/MAX9814.pdf).

Схема готова к реализации KWS-приложения, и на этом этапе вы можете подключить Raspberry Pi Pico к компьютеру с помощью кабеля micro-USB.

Выборка звука на Raspberry Pi Pico с помощью АЦП и прерываний по таймеру

Теперь все компоненты смонтированы на макетной плате. Таким образом, нам остается только написать наше KWS-приложение.

Приложение состоит из записи односекундного аудио и запуска просчета модели ML при нажатии кнопки. Результат классификации будет показан с помощью RGB-светодиодов аналогично тому, что мы делали в непрерывном распознавании на Arduino Nano.

Описанный далее код содержится в скетче Arduino (*09_kws_raspberrypi_pico.ino*) и скрипте Python (*09_debugging.py*) по следующим адресам:

- https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter04/ArduinoSketches/09_kws_raspberrypi_pico.ino;
- https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter04/PythonScripts/09_debugging.py.

Подготовка

Приложение для Raspberry Pi Pico будет основано на примере *Edge Impulse nano_ble33_sense_microphone.cpp*. В нем пользователь говорит в четко определенное время, а приложение рассчитывает ML-модель, чтобы угадать произнесенное слово.

В отличие от того, что мы реализовали в непрерывном распознавании на Arduino Nano, задачи записи и обработки звука здесь могут выполняться последовательно, так как нажатие кнопки сообщит нам о начале высказывания.

В следующем подразделе будет представлен используемый в этом примере подход для выборки аудиосигнала с помощью АЦП и прерываний по таймеру.

Выборка звука на Raspberry PI Pico с помощью АЦП и прерываний по таймеру

Микроконтроллер RP2040 на Raspberry Pi Pico оснащен четырьмя АЦП с 12-битным разрешением и максимальной частотой дискретизации 500 кГц,

или 500 тыс. выборок в секунду. АЦП будет настроен в режиме одноразового срабатывания, когда он предоставляет данные строго по запросу.

Таймер будет инициализирован для запуска прерываний с той же частотой, что и частота дискретизации. Следовательно, процедура обработки прерывания таймера (ISR) будет отвечать за выборку сигнала, поступающего с микрофона, и сохранение данных в аудиобуфере.

Поскольку максимальная частота АЦП составляет 500 кГц, минимальное время между двумя последовательными преобразованиями составляет две микросекунды (мкс). Это ограничение намного перекрывает наши требования, поскольку аудиосигнал будет оцифровываться с частотой 16 кГц (каждые 62.5 мкс).

Как это делается...

Откройте в Arduino IDE пример *nano_ble33_sense_microphone* из **Examples | FROM LIBRARIES | <название вашего проекта>_INFERENCING** и внесите следующие изменения для реализации KWS-приложения на Raspberry Pi Pico.

1. Удалите все ссылки на библиотеку `PDM`, такие как файл заголовка (`#include <PDM.h>`) и вызовы методов класса PDM, поскольку они требуются только для встроенного микрофона Arduino Nano.
   Удалите код из функции `microphone_inference_record()`.
2. Объявите и инициализируйте глобальный массив объектов `mbed::DigitalOut` для управления RGB-светодиодами:

   ```
   mbed::DigitalOut rgb[] = {p9, p8, p7};
   ```

   Объявите и инициализируйте глобальный объект `mbed::DigitalOut` для управления встроенным светодиодом[65]:

   ```
   mbed::DigitalOut led_builtin(p25);
   #define ON 1
   #define OFF 0
   ```

   Поскольку все светодиоды в этой схеме питаются от источника тока, для их включения нам необходимо на вывод подавать напряжение высокого уровня (`HIGH`, `1`).
3. Определите целочисленную глобальную переменную (`current_color`) для отслеживания последнего распознанного цвета. Инициализируйте его равным 0 (*red*):

   ```
   size_t current_color = 0;
   ```

   Инициализируйте RGB-светодиоды в функции `setup()`, включив только `current_color`:

   ```
   rgb[0] = OFF; rgb[1] = OFF; rgb[2] = OFF; rgb[current_color] = ON; led_builtin = OFF;
   ```

[65] Имеется в виду светодиод, размещенный на плате Raspberry Pi Pico (подключен к выводу GP25).

Объявите и инициализируйте глобальный объект `mbed::DigitalIn` для считывания состояния кнопки:

```
mbed::DigitalIn button(p10);
#define PRESSED 0
```

Установите режим вывода кнопки на подтягивание к питанию (`PullUp`) в функции `setup()`:

```
button.mode(PullUp);
```

Поскольку кнопка напрямую подключена к GND и выводу GPIO, мы должны включить внутренний подтягивающий резистор (режим PullUp). Числовое значение, возвращаемое объектом `mbed::DigitalIn`, будет равно 0 при нажатии кнопки.

5. Добавьте заголовочный файл `hardware/adc.h` для использования периферийного устройства АЦП:

```
#include "hardware/adc.h"
```

Инициализируйте периферийное устройство ADC0 (вывод GP26) в функции `setup()` с помощью Raspberry Pi Pico API:

```
adc_init(); adc_gpio_init(26); adc_select_input(0);
```

Фирма Raspberry Pi предлагает специальный API для микроконтроллера RP2040 в Raspberry Pi Pico SDK (https://raspberrypi.github.io/pico-sdkdoxygen/index.html).

Поскольку Raspberry Pi Pico SDK интегрирован в Arduino IDE, нам не нужно импортировать какую-либо библиотеку. Нам просто нужно включить в скетч файл заголовка (`hardware/adc.h`), чтобы использовать API аналогово-цифрового преобразователя.

АЦП инициализируется с помощью вызова следующих функций в `setup()`:

A) `adc_init()` – для инициализации периферийного устройства АЦП;
B) `adc_gpio_init(26)` – для инициализации вывода GPIO, используемого АЦП. Для этой функции необходим номер GPIO, подключенный к периферийному устройству АЦП. Мы указываем 26, так как ADC0 подключен к GP26;
C) `adc_select_input(0)` – чтобы инициализировать вход АЦП. Вход АЦП – это номер АЦП, подключенного к выбранному GPIO. Так как мы используем ADC0, в функции указываем 0.

При помощи указанных функций мы инициализируем АЦП в режиме однократного преобразования.

6. Объявите глобальный объект `mbed::Ticker` для использования таймера:

```
mbed::Ticker timer;
```

Объект `timer` будет использоваться для запуска прерываний по таймеру с частотой дискретизации звука (16 кГц).

7. Напишите обработчик прерывания таймера (ISR) для выборки звука, поступающего с микрофона:

```
#define BIAS_MIC ((int16_t)(1.25f * 4095) / 3.3f)
volatile int ix_buffer = 0;
volatile bool is_buffer_ready = false;
void timer_ISR() {
  if(ix_buffer < EI_CLASSIFIER_RAW_SAMPLE_COUNT) {
    int16_t v = (int16_t)((adc_read() - BIAS_MIC));
    inference.buffer[ix_buffer] = v;
    ++ix_buffer;
  }
  else {
    is_buffer_ready = true;
  }
}
```


Обработчик прерывания (ISR) производит выборку сигнала микрофона с помощью функции `adc_read()`, возвращающей значение от 0 до 4096, так как разрешение АЦП равно 12 бит. Поскольку сигнал, генерируемый усилителем MAX9814, имеет смещение 1.25 В, мы должны вычесть соответствующее цифровое значение из измерения. Соотношение между входным напряжением, образцовым (опорным) напряжением АЦП и преобразованным цифровым значением устанавливается следующей формулой:

$$DS = \frac{(2^{resolution} - 1) \cdot VS}{VREF},$$

где:

- *DS* – цифровой результат преобразования;
- *resolution* – разрешение АЦП;
- *VS* – отсчет входного напряжения;
- *VREF* – опорное напряжение питания АЦП (в данном случае напряжение на выводе ADC_VREF).

Таким образом, 12-разрядный АЦП при напряжении VREF 3.3 В преобразует смещение 1.25 В в число 1552.

После вычитания смещения из измерения, результат можно сохранить в аудиобуфере (`inference.buffer[ix_buffer] = v`), а затем увеличить индекс буфера (`++ix_buffer`).

Аудиобуфер должен быть динамически выделен в `setup()` с помощью функции `microphone_inference_start()`, он должен хранить количество семплов, необходимое для записи за 1 с. Параметр `EI_CLASSIFIER_RAW_SAMPLE_COUNT` предоставляется Edge Impulse для определения количества семплов в односекундном аудио. Поскольку мы отбираем аудиопоток с частотой дискретизации 16 кГц, аудиобуфер будет содержать 16 000 выборок формата `int16_t`.

ISR-обработчик устанавливает `is_buffer_ready` в значение `true`, когда аудиобуфер заполнен (`ix_buffer` больше или равен `EI_CLASSIFIER_RAW_SAMPLE_COUNT`).

Переменные ix_buffer и is_buffer_ready являются глобальными, поскольку они используются основной программой для определения готовности записи. Так как ISR изменяет эти переменные, мы должны объявить их типа volatile, чтобы предотвратить оптимизацию при компиляции.

8. В функции microphone_inference_record() вписываем код для записи одной секунды аудио:

```
bool microphone_inference_record(void) {
  ix_buffer = 0;
  is_buffer_ready = false;
  led_builtin = ON;
  unsigned int sampling_period_us = 1000000 / 16000;
  timer.attach_us(&timer_ISR, sampling_period_us);
  while(!is_buffer_ready);
  timer.detach();
  led_builtin = OFF;
  return true;
}
```

В функции microphone_inference_record() мы устанавливаем переменную ix_buffer равной 0, а переменную is_buffer_ready равной false каждый раз, когда начинаем новую запись.

Пользователь узнает, когда начнется запись, благодаря встроенному светодиоду (led_builtin = ON).

На этом этапе мы инициализируем объект mbed::Ticker для запуска прерываний с частотой 16 кГц. Для этого мы вызываем метод attach_us(), который требует следующего:

- вызова ISR при срабатывании прерывания (&timer_ISR);
- интервала времени, в течение которого мы запускаем прерывание. Поскольку мы отбираем аудиосигнал на частоте 16 кГц, мы передаем 62 мкс (unsigned int sampling_period_us = 1000000 / 16000).

Оператор while(!is_buffer_ready) используется для проверки того, завершена ли аудиозапись. Когда запись заканчивается, мы можем прекратить генерировать прерывания по таймеру (timer.detach()) и выключить встроенный светодиод (led_builtin = OFF).

9. Проверка нажатия кнопки в функции loop():

```
if(button == PRESSED) {
```

Если это так, ожидаем почти секунду (например, 700 мс), чтобы избежать записи механического звука нажатой кнопки:

```
  delay(700);
```

Мы рекомендуем не отпускать кнопку до конца записи, чтобы также предотвратить запись механического звука при ее отпускании.

Затем запишите 1 с аудио с помощью функции microphone_inference_record() и запустите расчет модели, вызвав run_classifier():

```
  microphone_inference_record();

  signal_t signal;
  signal.total_length = EI_CLASSIFIER_RAW_SAMPLE_COUNT;
  signal.get_data = &microphone_audio_signal_get_data;
  ei_impulse_result_t result = { 0 };

  run_classifier(&signal, &result, debug_nn);
```

После функции `run_classifier()` вы можете для управления RGB-светодиодами использовать код, написанный в примере «*Непрерывное распознавание на Arduino Nano*».

Однако прежде чем завершить функцию `loop()`, дождитесь, пока кнопка будет отпущена:

```
  while(button == PRESSED);
}
```

Теперь скомпилируйте и загрузите скетч на Raspberry Pi Pico. Когда устройство будет готово, нажмите кнопку, дождитесь, пока загорится встроенный светодиод, и попробуйте управлять светодиодами RGB голосом, произнося ключевые слова громко и поблизости от микрофона.

Теперь вы имеете возможность управлять светодиодами RGB своим голосом!

Дополнительно

Что мы можем сделать, если приложение не работает? У этого могут быть разные причины, но одна из них может быть связана с записанным звуком. Как мы можем узнать, правильно ли записан звук?

Для отладки приложения мы внедрили Python-скрипт *09_debugging.py*: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter04/PythonScripts/09_debugging.py.

Скрипт генерирует аудиофайл (*.wav*) из аудио, записанного Raspberry Pi Pico. Скрипт работает локально на вашем компьютере и нуждается в модулях `PySerial`, `uuid`, `Struct` и `Wave` в вашей среде.

Для использования Python-скрипта при отладке приложения на Raspberry Pi Pico необходимы следующие действия.

1. Импортируйте скетч *09_kws_raspberrypi_pico.ino* в среде Arduino IDE: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter04/ArduinoSketches/09_kws_raspberrypi_pico.ino.
   Установите для переменной `debug_audio_raw` значение `true`. Этот флаг позволит Raspberry Pi Pico передачу звуковых семплов по последовательному каналу всякий раз, когда у нас появляется новая запись.
2. Скомпилируйте и загрузите скетч *09_kws_raspberrypi_pico.ino* на Raspberry Pi Pico.
3. Запустите Python-скрипт *09_debugging.py* со следующими входными аргументами:

- `--label`: метка, присвоенная записанному высказыванию. Метка будет префиксом для имени сгенерированного аудиофайла в формате *.wav*;
- `--port`: имя последовательного порта, используемого Raspberry Pi Pico.

Имя порта зависит от операционной системы – например, `/dev/ttyACM0` на GNU/Linux или `COM1` в Windows. Самый простой способ узнать название последовательного порта – из выпадающего меню в Arduino IDE:

Рис. 4.42 ❖ Выпадающее меню устройства в веб-редакторе Arduino

Как только скрипт Python будет выполнен, он будет анализировать звуковые семплы, передаваемые по последовательному каналу, для создания файла *.wav* всякий раз, когда вы нажимаете кнопку.

Вы можете прослушать аудиофайл с помощью любой программы, способной открывать файлы *.wav*. Если уровень звука в .wav-файле слишком низкий, попробуйте во время записи говорить громче и ближе к микрофону.

Однако предположим, что уровень звука приемлемый, а приложение по-прежнему не работает. В этом случае модель ML, вероятно, недостаточно универсальна, чтобы иметь дело с сигналом электретного микрофона. Чтобы устранить эту проблему, вы можете расширить обучающий набор данных в Edge Impulse с образцами звука, полученными с этого микрофона. Для этой области загрузите сгенерированные аудиофайлы *.wav* в разделе сбора данных Edge Impulse и снова обучите модель. После того как вы подготовили модель, вам просто нужно создать новую библиотеку Arduino и импортировать ее в Arduino IDE.

Если вам интересно, как работает этот скрипт, не торопитесь. В следующей главе вы узнаете об этом больше.

Глава 5

Распознавание интерьеров помещений с помощью TensorFlow Lite for Microcontrollers и Arduino Nano

Компьютерное зрение сделало сверточные нейронные сети чрезвычайно популярными. Без этого алгоритма глубокого обучения такие задачи, как распознавание объектов, интерьеров или оценка позы, были бы действительно сложными. В настоящее время многие современные приложения для камер основаны на машинном обучении (ML), и нам просто нужно взять смартфон, чтобы увидеть их в действии. Компьютерное зрение также находит место в микроконтроллерах, хотя и с ограничениями, учитывая уменьшенную встроенную память.

В этой главе мы увидим преимущества добавления зрения к нашим миниустройствам путем распознавания помещений с помощью модуля камеры OV7670 в сочетании с платой Arduino Nano 33 BLE Sense.

В первой части мы узнаем, как получать изображения с модуля камеры OV7670. Затем сосредоточимся на дизайне модели, применяя трансфертное обучение[66] с помощью Keras API для распознавания кухонь и ванных комнат. Наконец, мы развернем квантизованную модель **TensorFlow Lite** (TFLite) на Arduino Nano с помощью **TensorFlow Lite for Microcontrollers** (TFLu).

Цель этой главы – показать, как применять трансфертное обучение с помощью TensorFlow, и изучить лучшие практические примеры использования модуля камеры с микроконтроллером.

---

[66] Трансфертное обучение (*Transfer Learning*) – способ машинного обучения, основанный на применении параметров модели, полученных в одной задаче, к другой целевой задаче.

В этой главе мы собираемся выполнить следующие примеры.

- Съемка с помощью модуля камеры OV7670.
- Захват кадров камеры через последовательный порт с помощью Python.
- Преобразование изображений QQVGA[67] из YCbCr422 в RGB888.
- Создание набора данных для распознавания интерьеров помещений.
- Трансфертное обучение с помощью Keras API.
- Подготовка и тестирование квантизованной модели TFLite.
- Сокращение объема RAM за счет объединения функций обрезки, изменения размера, масштабирования и квантизации.

Технические требования

Чтобы выполнить все практические примеры этой главы, нам понадобится следующее:

- плата Arduino Nano 33 BLE Sense,
- кабель micro-USB,
- 1 беспаечная макетная плата половинного размера,
- 1 модуль камеры OV7670,
- 1 тактовая кнопка,
- 18 соединительных проводов-перемычек (штырь–гнездо),
- ноутбук/ПК с Ubuntu 18.04+ или Windows 10 на x86-64.

Исходный код и дополнительные материалы доступны по адресу https://github.com/PacktPublishing/TinyML-Cookbook/tree/main/Chapter05.

Съемка с помощью модуля камеры OV7670

Добавление способности видеть в Arduino Nano – наш первый шаг к открытию приложений компьютерного зрения.

В этом первом примере мы создадим электронную схему для съемки камерой OV7670 с помощью Arduino Nano. После сборки схемы мы будем использовать предварительно созданный Arduino-скетч *CameraCaptureRawBytes* для передачи значений пикселей через последовательный порт.

[67] QQVGA (Quarter-Quarter-VGA) – разрешение изображения 160×120, равное четверти от четвертой части VGA по площади, или по ¼ от VGA (640×480) по каждой стороне (см. подробные пояснения на стр. 183 и далее).

Скетч Arduino *01_camera_capture.ino*, содержащий код, описываемый в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter05/ArduinoSketches/01_camera_capture.ino.

Подготовка

Модуль камеры OV7670 является основным компонентом, необходимым для съемки с помощью Arduino Nano. Это одна из самых доступных камер для приложений TinyML – вы можете купить ее у различных дистрибьюторов менее чем за 10 долл. Однако стоимость не единственная причина, по которой мы выбрали этот модуль. Следующие факторы делают это устройство нашим предпочтительным вариантом:

- **разрешение кадра и поддержка различных цветовых форматов**. Поскольку микроконтроллеры имеют ограниченную память, нам следует рассмотреть камеры, способные передавать изображения с низким разрешением. Камера OV7670 является хорошим выбором, поскольку она может выводить изображения в формате QVGA (320×240) и QQVGA (160×120). Кроме того, устройство может кодировать изображения в различных цветовых форматах, таких как RGB565, RGB444 и YCbCr422[68];
- **поддержка программной библиотеки**. Камерами может быть сложно управлять без программного драйвера. Поэтому для упрощения программирования обычно рекомендуются видеодатчики с поддержкой программной библиотеки. OV7670 имеет библиотеку поддержки для платы Arduino Nano 33 BLE Sense (https://github.com/arduino-libraries/Arduino_OV767X), которая уже интегрирована в веб-редактор Arduino.

Эти факторы, наряду с напряжением питания, потребляемой мощностью, частотой кадров и интерфейсом, обычно учитываются при выборе видеомодуля для приложений TinyML.

Как это делается...

Давайте начнем с этого примера, взяв беспаечную макетную плату и установив Arduino Nano вертикально между левой и правой шинами питания, как показано на следующем рисунке:

[68] RGB565, RGB444 и YCbCr422 – различные экономичные способы представления цвета в компьютерных изображениях. Цифры в конце означают количество бит на каждую составляющую пикселя (например, RGB444 означает по 4 бита на каждый цвет R, G и B, всего 12 бит на пиксель, RGB565 – 5 бит R, 6 бит G и 5 бит B, всего 16 бит; о формате YCbCr422 см. далее стр. 184).

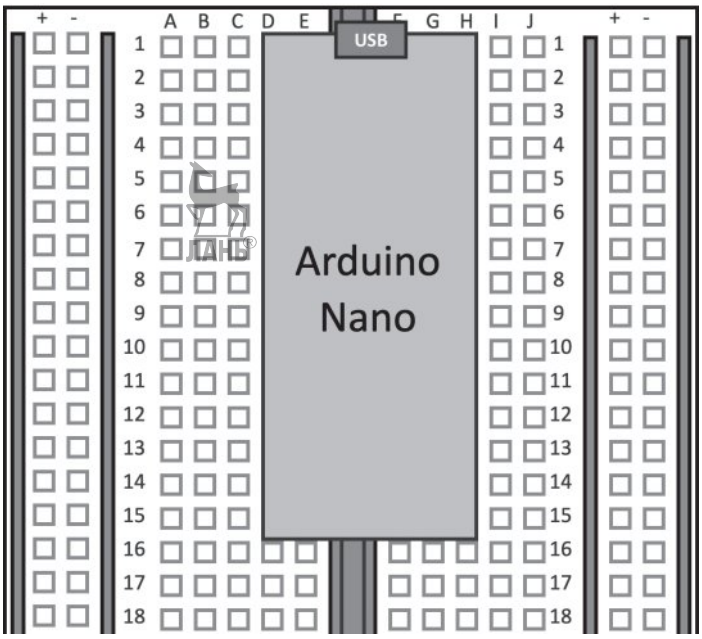


Рис. 5.1 ❖ Arduino Nano установлен вертикально между левой и правой шинами питания

Чтобы собрать схему с Arduino Nano, модулем OV7670 и кнопкой, необходимо сделать следующее.

1. Подключите модуль камеры OV7670 к Arduino Nano с помощью 16 соединительных проводов типа штырь–гнездо, как показано на следующей схеме:

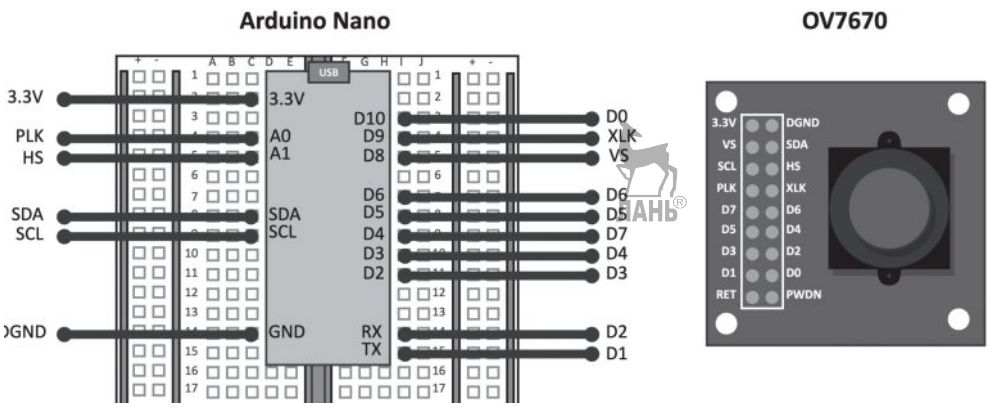


Рис. 5.2 ❖ Соединения между Arduino Nano и OV7670

Хотя OV7670 имеет 18 контактов, нам нужно подключить только 16 из них. Модуль камеры OV7670 подключен к Arduino Nano в соответствии

с расположением, указанным для программной библиотеки Arduino_OV767X.

Вы можете найти расположение выводов, требуемое библиотекой Arduino_OV767x, по следующей ссылке:

https://github.com/arduino-libraries/Arduino_OV767X/blob/master/src/OV767X.h.

2. Добавьте кнопку на макетной плате и подключите ее к P0.30 и GND:

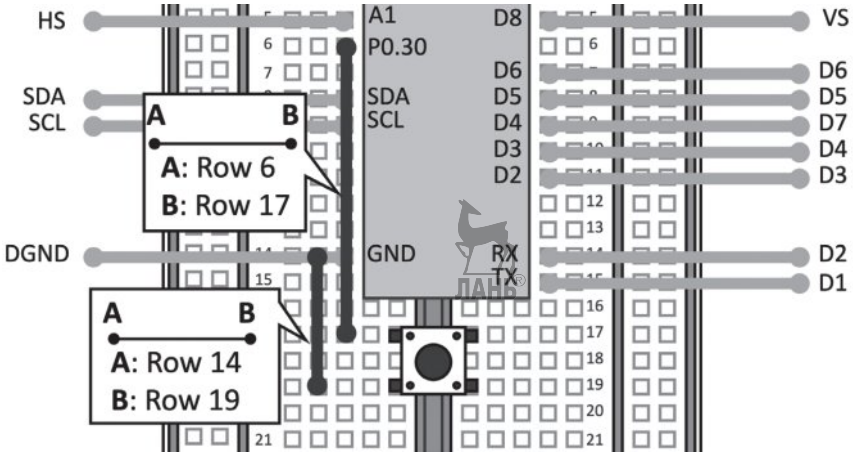


Рис. 5.3 ❖ Кнопка, подключенная между P0.30 и GND

Кнопка не нуждается в дополнительном резисторе, потому что мы будем использовать подтягивающий резистор микроконтроллера.

Чтобы запускать скетч для съемки всякий раз, когда мы нажимаем кнопку, откройте Arduino IDE и выполните следующие действия.

1. Откройте скетч *CameraCaptureRawBytes* из примеров к библиотеке **Examples -> FROM LIBRARIES -> ARDUINO_OV767X**:

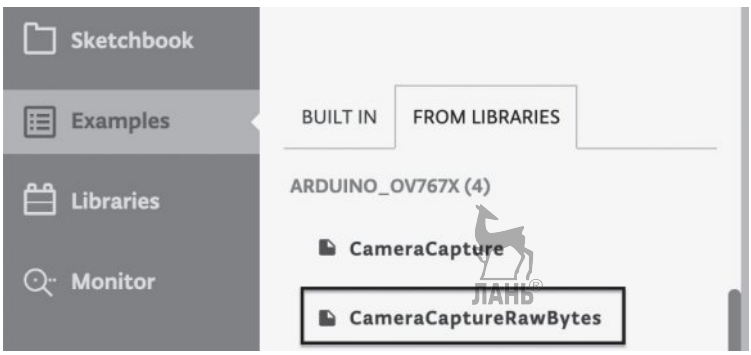


Рис. 5.4 ❖ Пример *CameraCaptureRawBytes*

Скопируйте содержимое CameraCaptureRawBytes в новый скетч.

2. Объявите и инициализируйте глобальный объект `mbed::DigitalIn` для считывания состояния кнопки:

```
mbed::DigitalIn button(p30);
#define PRESSED 0
```

Затем установите в функции `setup()` режим кнопки на подтягивание:

```
button.mode(PullUp);
```

3. Установите в функции `setup()` скорость передачи данных последовательного порта на 115600:

```
Serial.begin(115600);
```

4. Добавьте оператор `if` в функцию `loop()` для проверки, нажата ли кнопка. Если кнопка нажата, сделайте снимок с камеры OV7670 и отправьте значения пикселей через последовательный порт:

```
if(button == PRESSED) {
  Camera.readFrame(data);
  Serial.write(data, bytes_per_frame);
}
```

Vpp – это полная высота формы сигнала. Имена переменных в имеющемся скетче *CameraCaptureRawBytes* вводятся в Pascal-стиле, поэтому первая буква каждого названия пишется с заглавной буквы. Чтобы сохранить соответствие с соглашением об именовании в нижнем регистре, используемым в книге, мы переименовали `BytesPerFrame` в `bytes_per_frame`.

Скомпилируйте и загрузите эскиз на Arduino Nano. Теперь вы можете открыть монитор последовательного порта, нажав на **Monitor** в меню **Editor**. Там вы увидите все значения пикселей, передаваемые каждый раз, когда вы нажимаете кнопку.

Захват кадров камеры через последовательный порт с помощью Python

В предыдущем примере мы показали, как делать снимки с OV7670, но не представили способ их отображения.

В этом примере для анализа передаваемых последовательно значений пикселей и отображения захваченных изображений на экране будет использоваться Python.

Описанный далее код содержится в скетче Arduino (*02_camera_capture_qvga_rgb565.ino*) и скрипте Python (*02_parse_camera_frame.py*) по следующим адресам:

- https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter05/ArduinoSketches/02_camera_capture_qvga_rgb565.ino;

- https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter05/PythonScripts/02_parse_camera_frame.py.

Подготовка

В отличие от всех программ на Python, разработанных до сих пор, мы напишем скрипт на Python на нашем локальном компьютере для доступа к последовательному порту, используемому Arduino Nano.

Синтаксический анализ последовательных данных с помощью Python не требует особых усилий с помощью библиотеки `pySerial`, которую можно установить через Python package manager (`pip`):

```
$ pip install pyserial
```

Однако какого формата данных мы должны ожидать от микроконтроллера?

Передача изображений RGB888 через последовательный порт

Чтобы упростить синтаксический анализ пикселей, передаваемых по последовательному каналу, мы внесем некоторые изменения в скетч Arduino предыдущего примера для отправки изображений в формате RGB888.

Этот формат упаковывает пиксель в 3 байта, используя 8 бит для каждого цветового компонента. Использование RGB888 означает, что наш скрипт на Python может напрямую создавать изображение с помощью **PIL**[69] без дополнительных преобразований.

Однако хорошей практикой является передача изображения с **метаданными** (**metadata**), чтобы упростить синтаксический анализ и проверять ошибки связи.

В нашем случае метаданные будут содержать следующую информацию.

1. **Начало передачи изображения**. Мы отправляем строку `<image>` для обозначения начала сеанса связи.
2. **Разрешение изображения**. Мы отправляем разрешение изображения в виде строки цифр, чтобы указать, сколько пикселей RGB будет передано. Ширина и высота будут отправлены в двух разных строках.
3. **Завершение передачи изображения**. После отправки всех значений пикселей мы передаем строку `</image>`, чтобы уведомить об окончании сеанса связи.

Значения пикселей будут отправлены сразу после метаданных разрешения изображения и в порядке сканирования растра сверху вниз, слева направо:

[69] Python Imaging Library (сокращенно PIL) – библиотека языка Python, предназначенная для работы с растровой графикой. Автором используется ее современная версия под названием Pillow (см. далее).

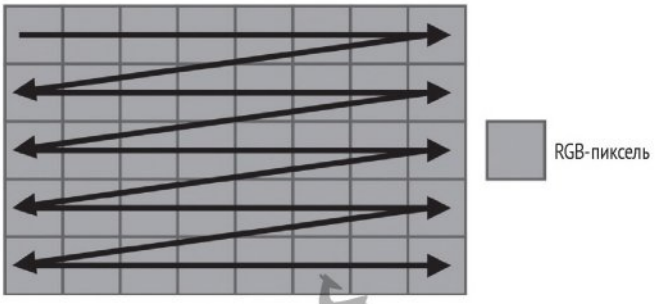


Рис. 5.5 ❖ Порядок сканирования растра

Цветовые компоненты будут отправлены в виде строк цифр, заканчивающихся символом новой строки (\n), и следуя порядку RGB. Таким образом, красный канал появляется первым, а синий последним, как показано на следующем рисунке:

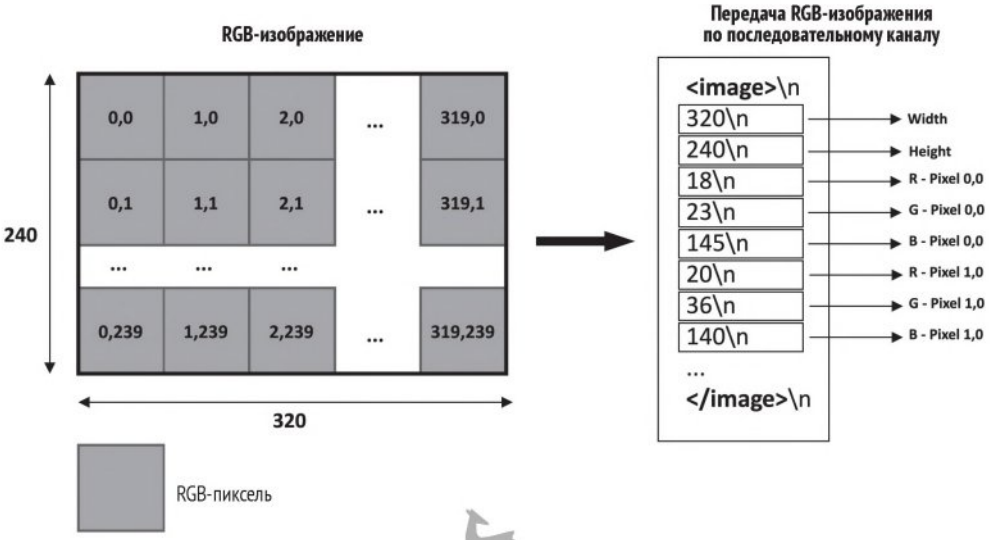


Рис. 5.6 ❖ Протокол последовательной передачи RGB-изображения

Как вы можете видеть из этой иллюстрации, значения пикселей передаются в соответствии с порядком сканирования растра. Каждый цветовой компонент отправляется в виде строки цифр, заканчивающейся символом новой строки (\n).

Однако камера OV7670 инициализирована для вывода изображений в цветовом формате RGB565. Поэтому нам нужно преобразовать пиксели камеры в формат RGB888 перед отправкой их по последовательному каналу.

Изучаем, как преобразовать RGB565 в RGB888

Вы, возможно, заметили, что RGB565 – это формат, используемый при инициализации камеры в скетче CameraCaptureRawBytes:

```
Camera.begin(QVGA, RGB565, 1)
```

RGB565 упаковывает пиксель в 2 байта (16 бит), в которых по 5 бит отводятся для красного и синего компонентов и 6 бит для зеленого:

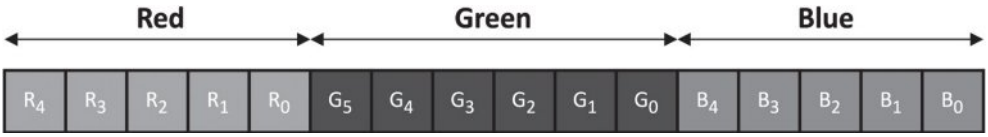


Рис. 5.7 ❖ Цветовой формат RGB565

Этот цветовой формат находит применение в основном во встраиваемых системах с ограниченным объемом памяти, поскольку он уменьшает размер изображения. Однако уменьшение объема памяти достигается за счет уменьшения динамического диапазона цветовых компонентов.

Как это делается...

Чтобы отправить по последовательному каналу пиксели в формате RGB888, нужно внести изменения в скетч Arduino предыдущего примера по следующим шагам. Как только скетч будет готов, мы напишем скрипт на Python для отображения передаваемого изображения на экране.

1. Напишите функцию для преобразования пикселя RGB565 в RGB888:

```
void rgb565_rgb888(uint8_t* in, uint8_t* out) {
  uint16_t p = (in[0] << 8) | in[1];
  out[0] = ((p >> 11) & 0x1f) << 3;
  out[1] = ((p >> 5) & 0x3f) << 2;
  out[2] = (p & 0x1f) << 3;
}
```

Функция берет 2 байта из входного буфера для формирования 16-битного пикселя в формате RGB565. Первый байт (`in [0]`) сдвигается влево на восемь позиций, чтобы поместить его в старшую половину переменной `p` типа `uint16_t`. Второй байт (`in[1]`) занимает младшую половину:

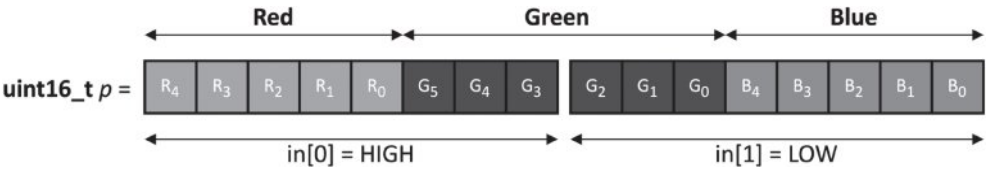


Рис. 5.8 ❖ Пиксель RGB565 формируется с помощью байт `in[0]` и `in[1]`

Как только у нас есть 16-битный пиксель, мы получаем 8-битные цветовые компоненты из переменной `p`, сдвигая каждый канал вправо, начиная с младшего байта:

- 8-битный красный канал (`out[0]`) получается путем сдвига переменной `p` на 11 позиций так, что бит R0 становится младшим битом переменной `p` (типа `uint16_t`). После этого мы очищаем все не красные биты, применяя битовую маску 0x1F (все биты очищены, кроме пяти младших), и сдвигаем влево на три позиции так, что бит R4 становится старшим битом переменной `out[0]`;
- 8-битный зеленый канал (`out[1]`) получается путем сдвига переменной `p` на 5 позиций так, чтобы G0 стал младшим битом переменной `out[1]`. После этого мы очищаем все не зеленые биты, применяя битовую маску с 0x3F (все биты очищены, кроме шести младших), и сдвигаем влево на две позиции так, что бит G5 становится старшим битом переменной `out[1]`;
- 8-битный синий канал (`out[2]`) получается без сдвига, поскольку B0 уже является младшим битом переменной `p`. Следовательно, нам просто нужно очистить не синие биты, применив битовую маску с 0x1F (все биты очищены, кроме младших пяти), и сдвинуть влево на три позиции так, что бит B4 становится старшим битом переменной `out[2]`.

2. Включите `testPattern` в функции `setup()`:

```
Camera.testPattern();
```

При включении режима `testPattern` модуль `Camera` будет возвращать фиксированное изображение цветных полос.

3. В функции `loop()` замените функцию `Serial.write(data, bytes_per_frame)` процедурой отправки пикселей RGB888 по последовательному каналу:

```
Camera.readFrame(data);
uint8_t rgb888[3];
Serial.println("<image>");
Serial.println(Camera.width());
Serial.println(Camera.height());
const int bytes_per_pixel = Camera.bytesPerPixel();
for(int i = 0; i < bytes_per_frame; i+=bytes_per_pixel) {
  rgb565_rgb888(&data[i], &rgb888[0]);
  Serial.println(rgb888[0]);
  Serial.println(rgb888[1]);
  Serial.println(rgb888[2]);
}
Serial.println("</image>");
```

Связь начинается с отправки по последовательному каналу строки `<image>` и заданного разрешения изображения (`Camera.width()`, `Camera.height()`). Далее мы перебираем все байты, хранящиеся в буфере камеры, и применяем преобразование RGB565 в RGB888 с помощью функции `rgb565_rgb888()`. Затем каждый цветовой компонент отправляется

в виде строки цифр + символ новой строки (\n). По завершении отправляем строку </image>, чтобы обозначить окончание передачи данных. Теперь можно скомпилировать и загрузить скетч на Arduino Nano.

4. На вашем компьютере создайте новый скрипт на Python и импортируйте следующие модули:

```
import numpy as np
import serial
from PIL import Image
```

Инициализируйте pySerial с номером порта и скоростью передачи данных, используемых микроконтроллером Arduino Nano:

```
port = '/dev/ttyACM0'
baudrate = 115600
ser = serial.Serial()
ser.port = port
ser.baudrate = baudrate
```


Самый простой способ узнать имя последовательного порта – из выпадающего меню устройства в Arduino IDE:

Рис. 5.9 ❖ Выпадающее меню устройства в веб-редакторе Arduino

На этом скриншоте имя последовательного порта – /dev/ttyACM0.
Затем откройте последовательный порт и удалите содержимое входного буфера:

```
ser.open()
ser.reset_input_buffer()
def serial_readline():
data = ser.readline
return data.decode("utf-8").strip()
```

Строка, передаваемая Arduino Nano по последовательному каналу, кодируется в UTF-8 и заканчивается символом новой строки. Поэтому мы декодируем байты, закодированные UTF-8 (.decode("utf-8")), и удаляем символ новой строки с помощью .strip().

7. Создайте 3D-массив с помощью NumPy[70] для хранения значений пикселей, передаваемых по последовательному каналу. Поскольку Arduino Nano отправляет разрешение кадра, вы можете инициализировать ширину и высоту с помощью, например, цифры 1, и изменить размер массива NumPy позже при разборе последовательного потока:

[70] NumPy – библиотека с открытым исходным кодом, ориентированная на операции с многомерными массивами и ряд других распространенных математических задач.

```
width = 1
height = 1
num_ch = 3
image = np.empty((height, width, num_ch), dtype=np.uint8)
```

8. Используйте цикл `while` для чтения последовательных данных построчно:

```
while True:
  data_str = serial_readline()
```

Проверьте, есть ли у нас метаданные `<image>`:

```
  if str(data_str) == "<image>":
```

Если это так, получите разрешение кадра (ширину и высоту) и соответствующим образом измените размер массива NumPy:

```
    w_str = serial_readline()
    h_str = serial_readline()
    w = int(w_str)
    h = int(h_str)
    if w != width or h != height:
      if w * h != width * height:
        image.resize((h, w, num_ch))
      else:
        image.reshape((h, w, num_ch))
      width = w
      height = h
```

9. После разрешения кадра получите значения пикселей, передаваемые по последовательному каналу, и сохраните их в массиве NumPy:

```
for y in range(0, height):
for x in range(0, width):
for c in range(0, num_ch):
data_str = serial_readline()
image[y][x][c] = int(data_str)
```

Чтобы получить более эффективное решение, можно рассмотреть следующий альтернативный код без вложенных циклов for:

```
for i in range(0, width * height * num_ch):
c = int(i % num_ch)
x = int((i / num_ch) % width)
y = int((i / num_ch) / width)
data_str = serial_readline()
image[y][x][c] = int(data_str)
```

10. Проверьте, содержит ли последняя строка метаданные `</image>`. Если да, выведите изображение на экран:

```
data_str = serial_readline()
if str(data_str) == "</image>":
```

```
image_pil = Image.fromarray(image)
image_pil.show()
```

Оставив Arduino Nano подключенным к вашему компьютеру, запустите полученный Python-скрипт. Теперь всякий раз, когда вы нажимаете кнопку, программа Python анализирует данные, передаваемые по последовательному каналу, и через несколько секунд показывает изображение с восемью цветовыми полосами, которое можно посмотреть по следующей ссылке: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter05/test_qvga_rgb565.png.

Если вы не получаете изображение с этим тестовым шаблоном, мы рекомендуем проверить соединения между камерой и Arduino Nano.

Преобразование изображений QQVGA из YCbCr422 в RGB888

При компиляции предыдущего скетча на Arduino вы, возможно, заметили нехватку доступной памяти – может возникнуть предупреждение о нестабильности в сообщениях Arduino IDE.

Arduino IDE выдает это предупреждение, потому что для изображения QVGA в цветовом формате RGB565 требуется буфер размером 153.6 Кбайт, что составляет примерно 60 % от объема SRAM, доступного в микроконтроллере.

В этом примере мы покажем, как получить изображение с более низким разрешением и использовать цветовой формат YCbCr422 для предотвращения ухудшения качества изображения.

Скетч Arduino *03_camera_capture_qqvga_ycbcr422.ino*, содержащий код этого примера, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter05/ArduinoSketches/03_camera_capture_qqvga_ycbcr422.ino.

Подготовка

Основные возможности для уменьшения размера изображения основаны на разрешении и цветовом формате.

Цифровые изображения требуют больших объемов памяти, что может быть проблемой при работе с микроконтроллерами. Снижение разрешения изображения является обычной практикой для уменьшения объема памяти изображений.

Изображения со стандартным разрешением, принятым на микроконтроллерах, как правило, меньше, чем QVGA (320×240), например QQVGA (160×120) или QQQVGA (80×60). Существуют изображения даже с более низким разрешением, но они не всегда подходят для приложений компьютерного зрения.

Кодирование цветов – еще один рычаг для уменьшения объема памяти изображений. Как мы видели в предыдущем примере, формат RGB565 экономит память за счет уменьшения динамического диапазона цветовых ком-

понентов. Однако модуль камеры OV7670 предлагает более эффективную альтернативную цветовую кодировку: YCbCr422.

Преобразование YCbCr422 в RGB888

YCbCr422 – это цифровая цветовая кодировка, которая выражает компоненты цвета пикселя не в величинах интенсивности красного, зеленого и синего, а в величинах яркости (Y), цветоразностного синего (Cb) и цветоразностного красного (Cr).

Модуль камеры OV7670 может выводить изображения в формате YCbCr422, в котором Cb и Cr являются общими для двух последовательных пикселей на одной и той же линии сканирования. Следовательно, 4 байта используются для кодирования двух пикселей:

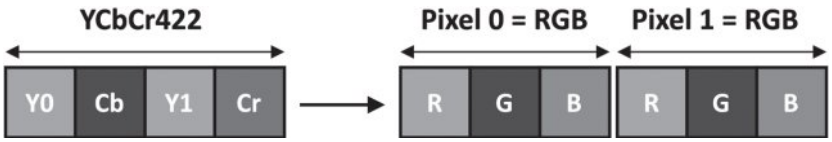


Рис. 5.10 ❖ Четыре байта в формате YCbCr422 содержат два пикселя RGB888

Хотя YCbCr422 по-прежнему требует 2 байта на пиксель, как RGB565, он обеспечивает лучшее качество изображения.

В следующей таблице приведены формулы для выполнения преобразования цвета из YCbCr422 в RGB888 с использованием только целочисленных арифметических операций:

Цвет	Формула
Red	$R_i = Y_i + Cr + (Cr \gg 2) + (Cr \gg 3) + (cr \gg 5) \in [0, 255]$
Green	$G_i = Y_i - (Cb \gg 2) - (Cb \gg 4) - (Cb \gg 5) - (Cr \gg 1) - (Cr \gg 3) - (Cr \gg 5) \in [0, 255]$
Blue	$B_i = Y_i + Cb + (Cb \gg 1) + (Cb \gg 2) + (Cb \gg 6) \in [0, 255]$

Рис. 5.11 ❖ Формулы для преобразования YCbCr422 в RGB888

Индекс i в R_i, G_i, B_i и Y_i представляет номер пикселя и равен либо 0 (первый пиксель), либо 1 (второй пиксель).

Как это делается...

Откройте скетч Arduino из предыдущего примера и внесите следующие изменения, чтобы получить изображения QQVGA YCbCr422 из модуля камеры OV7670.

1. Измените размер буфера данных камеры, чтобы разместить изображение QQVGA в формате YCbCr422:

```
byte data[160 * 120 * 2];
```

   Разрешение QQVGA требует буфер в четыре раза меньше, чем использовавшийся в предыдущем примере.
2. Напишите функцию для получения пикселя RGB888 из компонентов Y, Cb и Cr:

```
template <typename T>
inline T clamp_0_255(T x) {
  return std::max(std::min(x, (T)255)), (T)(0));
}

void ycbcr422_rgb888(int32_t Y, int32_t Cb, int32_t Cr, uint8_t* out) {
  Cr = Cr - 128;
  Cb = Cb - 128;
  out[0] = clamp_0_255((int)(Y + Cr + (Cr >> 2) + (Cr >> 3) + (Cr >> 5)));
  out[1] = clamp_0_255((int)(Y - ((Cb >> 2) + (Cb >> 4) + (Cb >> 5)) -
                             ((Cr >> 1) + (Cr >> 3) + (Cr >> 4)) + (Cr >> 5)));
  out[2] = clamp_0_255((int)(Y + Cb + (Cb >> 1) + (Cb >> 2) + (Cb >> 6)));
}
```

   Функция возвращает два пикселя, потому что компоненты Cb и Cr являются общими для двух пикселей. Преобразование выполняется с использованием формул, приведенных на рис. 5.11.

Пожалуйста, обратите внимание, что драйвер OV7670 возвращает компонент Cr раньше Cb.

3. В функции `setup()` инициализируйте камеру OV7670 для выдачи кадров QQVGA в цветовом формате YCbCr422 (YUV422):

```
if (!Camera.begin(QQVGA, YUV422, 1)) {
  Serial.println("Failed to initialize camera!");
  while (1);
}
```

   К сожалению, для драйвера OV7670 YCbCr422 следует заменять на YUV422, что приводит к некоторой путанице. Основное различие между форматами YUV и YCbCr заключается в том, что YUV изначально был предназначен для аналогового телевидения. Поэтому, хотя в функции `Camera.begin()` мы передаем YUV422, мы фактически инициализируем устройство для формата YCbCr422.
4. В функции `loop()` удалите инструкцию, которая выполняет итерацию по пикселям RGB565, сохраненным в предыдущем буфере камеры. Затем создайте подпрограмму для чтения 4 байт из буфера для YCbCr422, возвращающую два пикселя RGB888:

```
const int step_bytes = Camera.bytesPerPixel() * 2;
for(int i = 0; i < bytes_per_frame; i+=step_bytes) {
```

```
    const int32_t Y0 = data[i + 0];
    const int32_t Cr = data[i + 1];
    const int32_t Y1 = data[i + 2];
    const int32_t Cb = data[i + 3];
    ycbcr422_to_rgb888_i(Y0, Cb, Cr, &rgb888[0]);
    Serial.println(rgb888[0]);
    Serial.println(rgb888[1]);
    Serial.println(rgb888[2]);
    ycbcr422_to_rgb888_i(Y1, Cb, Cr, &rgb888[0]);
    Serial.println(rgb888[0]);
    Serial.println(rgb888[1]);
    Serial.println(rgb888[2]);
  }
```

Скомпилируйте и загрузите скетч на Arduino Nano. Запустите скрипт на Python и нажмите кнопку на макете. Через несколько секунд вы должны снова увидеть на экране изображение с восемью цветовыми полосами, показанное по следующей ссылке: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter05/test_qqvga_ycbcr422.png.

Изображение должно быть меньше, но с более яркими цветами, чем изображение, полученное в формате RGB565.

Создание набора данных для распознавания интерьеров помещений

Теперь, когда мы научились снимать кадры с камеры, пришло время создать набор данных для распознавания интерьеров помещений.

В этом примере мы создадим набор данных, получив изображения кухни и ванной комнаты с помощью камеры OV7670.

Код, разбираемый в этом примере, содержит Python-скрипт *04_build_dataset.py*, доступный по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter05/PythonScripts/04_build_dataset.py.

Подготовка

При обучении для классификации изображений глубокой нейронной сети с нуля обычно требуется набор данных не менее чем с 1000 изображениями на каждый класс. Как вы могли догадаться, это решение непрактично в нашем случае, так как сбор тысяч фотографий занимает много времени.

Поэтому мы рассмотрим альтернативный метод ML: трансфертное обучение (**transfer learning**). Трансфертное обучение – популярный метод, который использует предварительно обученную модель для обучения глубокой нейронной сети с небольшим набором данных. Этот метод ML будет использоваться в данном примере, и для получения базовой рабочей модели требуется набор данных, содержащий всего 20 образцов на класс.

Как это делается...

Перед внедрением скрипта Python удалите режим тестового шаблона (`Camera.testPattern()`) в скетче Arduino, чтобы вы могли получать реальные изображения. После этого скомпилируйте и загрузите скетч на платформу.

Скрипт на Python, реализованный в этом примере, будет повторно использовать часть кода, разработанного в разделе «*Захват кадров камеры через последовательный порт с помощью Python*». Следующие шаги покажут, какие изменения необходимо внести в скрипт Python, чтобы сохранить захваченные изображения в виде файлов *.png* и создать набор данных для распознавания кухонь и ванных комнат.

1. Импортируйте модуль Python UUID:

   ```
   import uuid
   ```

   UUID будет использоваться для создания уникальных имен файлов в формате *.png*.
2. Добавьте переменную в начале программы для метки изображения:

   ```
   label = "test"
   ```

   Метка будет префиксом для имени файла в формате *.png*.
3. После получения изображения через последовательный канал обрежьте его в квадратную форму и отобразите на экране:

   ```
   crop_area = (0, 0, height, height)
   image_pil = Image.fromarray(image)
   image_cropped = image_pil.crop(crop_area)
   image_cropped.show()
   ```

   Мы обрезаем полученное с последовательного порта изображение в квадратную форму, потому что предварительно обученная модель будет использовать входные данные с квадратным соотношением сторон. Мы обрезаем изображение справа, оставляя область с размерами, соответствующими высоте исходного изображения, как показано на следующем рисунке:

Рис. 5.12 ❖ Обрезанная область (*cropping area*)

Затем изображение отображается на экране.

4. Спросите пользователя, можно ли сохранить изображение, и прочитайте ответ с помощью функции Python `input()`. Если пользователь вводит с клавиатуры, запросите название метки и сохраните изображение в виде файла *.png*:

```
key = input("Save image? [y] for YES: ")
  if key == 'y':
    str_label = "Write label or leave it blank to use [{}]: ".format(label)
    label_new = input(str_label)
    if label_new != '':
      label = label_new
    unique_id = str(uuid.uuid4())
    filename = label + "_"+ unique_id + ".png"
    image_cropped.save(filename)
```

Если пользователь оставит метку пустой, программа будет использовать последнюю предоставленную метку. Результирующее имя файла: <*label*>_<*unique_id*>*.png*, где <*label*> – метка, выбранная пользователем, а <*unique_id*> – уникальный идентификатор, сгенерированный библиотекой UUID.

5. Выполните 20 изображений кухонь и ванных комнат с помощью камеры OV7670. Поскольку мы делаем всего несколько снимков на каждый класс, рекомендуется навести камеру на определенные элементы комнат. Не забудьте также сделать 20 снимков для неизвестного класса, представляющих случаи, когда это не кухня и не ванная.

Получив все изображения, поместите их в отдельные подкаталоги, соответствующие имени класса, например:

Рис. 5.13 ❖ Пример структуры каталогов

После этого сожмите получившиеся три папки в единый zip-файл (*dataset.zip*).

Трансфертное обучение с помощью Keras API

Трансфертное обучение – эффективный метод получения немедленных результатов с помощью глубокого обучения при работе с небольшими наборами данных.

В этом примере мы применим трансфертное обучение наряду с предварительно обученной моделью MobileNet v2 для распознавания внутренней среды.

Colab notebook под названием *prepare_model.ipynb* (раздел «*Transfer learning with Keras*») содержит код, разбираемый в этом примере: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter05/ColabNotebooks/prepare_model.ipynb.

Подготовка

Трансфертное обучение использует предварительно обученную модель для получения рабочей модели ML за короткое время.

При выполнении классификации изображений с трансфертным обучением предварительно обученная модель (**convolution based network**) соединяется с обучаемым классификатором (**head**), как показано на следующем рисунке:

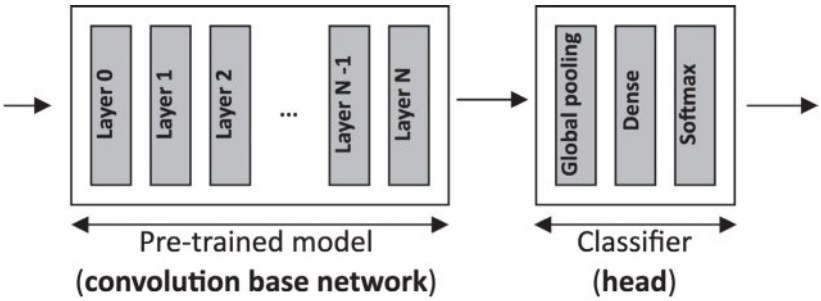


Рис. 5.14 ❖ Архитектура модели с трансфертным обучением

Как вы можете видеть из приведенной иллюстрации, предварительно обученная модель является основой для извлечения параметров и питает классификатор, обычно состоящий из глобального объединения (*global pooling*), уплотненных слоев (*dense*) и функции активации softmax.

В нашем сценарии мы будем обучать только классификатор. Следовательно, предварительно обученная модель будет зафиксирована и будет действовать как средство извлечения фиксированных параметров.

Keras предоставляет различные предварительно обученные модели, такие как VGG16, ResNet50, InceptionV3, MobileNet и т. д. Какую из них мы должны использовать?

При рассмотрении предварительно обученной модели для TinyML размер модели – это показатель, который следует учитывать, чтобы адаптировать архитектуру глубокого обучения к устройствам с ограниченным объемом памяти.

Из списка предварительно обученных моделей, предлагаемых Keras (https://keras.io/api/applications/), выберем MobileNet v2 – сеть с меньшим количеством параметров, предназначенную для развертывания на целевых устройствах с небольшой вычислительной мощностью.

Изучение вариантов дизайна сети MobileNet

MobileNet v2 – это второе поколение сетей MobileNet. По сравнению с предыдущим (MobileNet v1) в нем вдвое меньше операций и более высокая достоверность.

Эта модель – идеальное место, чтобы взять пример с архитектурных решений, которые сделали нейронные сети MobileNet небольшими, быстрыми и точными для фронтальных (*edge*) вычислений[71].

Одним из успешных дизайнерских решений, которые сделали первое поколение сетей MobileNet подходящим для фронтальных вычислений, было внедрение глубинной свертки (**depthwise convolution**).

Как мы знаем, традиционные слои свертки хорошо известны своей вычислительной дороговизной. Кроме того, при работе с ядром размером 3×3 или больше этому оператору обычно требуется дополнительная память, чтобы свести вычисления к процедурам умножения матриц.

Идея MobileNet v1 заключалась в замене стандартной свертки 2D (*convolution 2D*) на свертку, разделяемую по глубине, как показано на следующей иллюстрации:

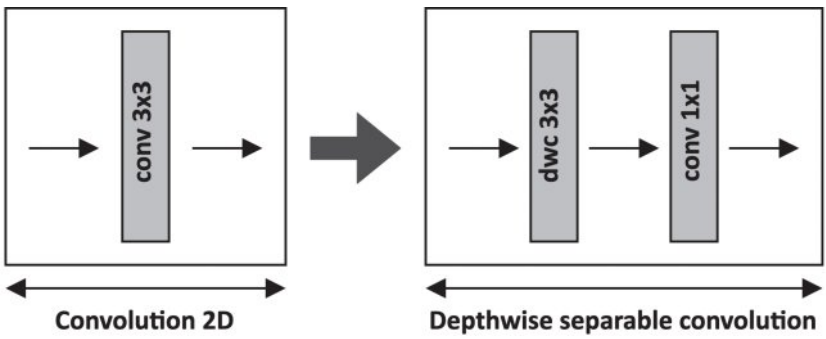


Рис. 5.15 ❖ Свертка, разделяемая по глубине

[71] Edge inferencing, edge computing (*фронтальные вычисления*) – подвид распределенных вычислений, в котором обработка информации происходит в непосредственной близости к месту, где данные были получены и будут потребляться. Подробнее см.: https://habr.com/ru/company/ibm/blog/508118/.

Как вы можете видеть из приведенной иллюстрации, разделяемая по глубине свертка состоит из свертки по глубине с размером ядра 3×3, за которой следует слой свертки с размером ядра 1×1 (также известный как точечная свертка, **pointwise convolution**). Это решение обеспечивает меньшее количество обучаемых параметров, меньшее использование памяти и меньшую вычислительную стоимость.

В главе 7 «*Запуск модели TinyML CIFAR-10 на виртуальной платформе ОС Zephyr*» имеется больше информации о преимуществах, предоставляемых сверткой, разделяемой по глубине.

Вычислительные затраты на MobileNet v2 были дополнительно снижены за счет выполнения свертки на тензорах с меньшим количеством каналов. С идеальной вычислительной точки зрения все слои должны работать на тензорах с небольшим количеством каналов (feature maps), чтобы улучшить задержку модели. Практически с точки зрения точности распознавания это означает, что наши компактные тензоры могут сохранять соответствующие параметры для задачи, которую мы хотим решить.

Разделяемая по глубине свертка сама по себе не может помочь, поскольку уменьшение количества feature maps приводит к снижению достоверности модели. Поэтому MobileNet v2 ввел остаточный сужающий блок (**bottleneck residual block**), чтобы уменьшить количество каналов, используемых в сети:

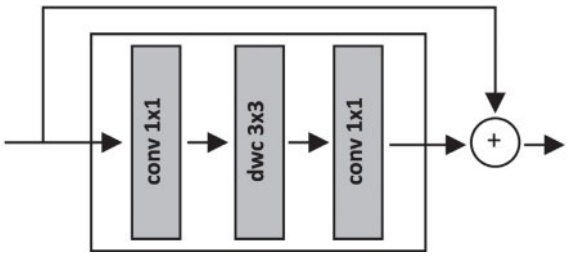


Рис. 5.16 ❖ Остаточный сужающий блок

Остаточный сужающий блок действует как функциональный компрессор. Как показано на рисунке, входные данные обрабатываются поточечной сверткой, которая расширяет (или увеличивает) количество отображений объектов. Затем выходные данные свертки передаются в слой свертки, разделяемой по глубине, чтобы сжать объекты в меньшем количестве выходных каналов.

Как это делается...

Создайте новый Colab notebook. Затем загрузите zip-файл, содержащий наш набор данных (*dataset.zip*), с помощью кнопки загрузки в верхней части менеджера файлов:

Рис. 5.17 ❖ Кнопка загрузки в верхней части менеджера файлов

Чтобы применить трансфертное обучение с помощью предварительно обученной модели MobileNet v2, выполните следующие действия.

1. Распакуйте архив набора данных:

```
import zipfile
with zipfile.ZipFile("dataset.zip", 'r') as zip_ref:
  zip_ref.extractall(".")
data_dir = "dataset"
```

2. Подготовьте обучающие и проверочные наборы данных:

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  interpolation="bilinear",
  image_size=(48, 48))

val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  interpolation="bilinear",
  image_size=(48, 48))
```

3. Показанный код изменяет размер входных изображений до 48×48 с помощью билинейной интерполяции и создает обучающие и проверочные наборы данных в соотношении 80/20.
4. Измените масштаб значений пикселей с [0, 255] на [–1, 1]:

```
rescale = tf.keras.layers.Rescaling(1./255, offset= -1)
train_ds = train_ds.map(lambda x, y: (rescale(x), y))
val_ds = val_ds.map(lambda x, y: (rescale(x), y))
```

Причина масштабирования значений пикселей с [0, 255] до [–1, 1] заключается в том, что предварительно обученная модель для входного тензора ожидает этот диапазон данных.

5. Импортируйте предварительно обученную модель MobileNet v2 с весами, полученными по набору данных ImageNet, и величиной `alpha` = 0.35. Кроме того, установите для входного изображения минимальное разрешение, допускаемое предварительно обученной моделью (48, 48, 3), и исключите верхние (полносвязные) слои:

```
base_model = MobileNetV2 (input_shape=(48, 48, 3),
                          include_top=False, weights='imagenet', alpha=0.35)
```

Keras предлагает более одного варианта MobileNet v2. Из списка моделей MobileNet v2 Keras (https://github.com/keras-team/keras-applications/blob/master/keras_applications/mobilenet_v2.py) мы выбираем *mobilenet_v2_0.35_96* с наименьшим размером входных данных (48, 48, 3) и наименьшим значением `alpha` (0.35).

6. Заморозьте значения весов, чтобы не обновлять их во время тренировки:

```
base_model.trainable = False
feat_extr = base_model
```

7. Дополните входные данные:

```
augmen = tf.keras.Sequential([
tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
tf.keras.layers.experimental.preprocessing.RandomRotation(0.2)])

train_ds = train_ds.map(lambda x, y: (augmen(x), y))
val_ds = val_ds.map(lambda x, y: (augmen(x), y))
```

Поскольку у нас нет большого набора данных, мы рекомендуем искусственно применить некоторые случайные преобразования к изображениям, чтобы предотвратить переобучение.

8. Подготовьте классификационный глобально объединенный слой, за которым следует плотный слой с активацией softmax:

```
global_avg_layer = tf.keras.layers.GlobalAveragePooling2D()

dense_layer = tf.keras.layers.Dense(3, activation='softmax')
```

9. Задайте архитектуру модели:

```
inputs = tf.keras.Input(shape=MODEL_INPUT_SIZE)
x = global_avg_layer(feat_extr.layers[-1].output)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = dense_layer(x)
model = tf.keras.Model(inputs=feat_extr.inputs, outputs=outputs)
```

Мы рекомендуем передать модулю извлечения параметров `training =False`, чтобы не обновлять внутренние переменные слоев с пакетной нормализацией (среднее значение и дисперсию) в MobileNet v2.

10. Скомпилируйте модель со скоростью обучения 0.0005:

```
lr = 0.0005
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
    loss=tf.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy'])
```

Скорость обучения по умолчанию, используемая TensorFlow, равна 0.001. Причина снижения скорости обучения до 0.0005 заключается в предотвращении переобучения.

11. Обучите модель в течение 10 эпох (*epochs*)[72]:

```
model.fit(train_ds, validation_data=val_ds, epochs=10)
```

Ожидаемая достоверность по проверочному набору данных должна составлять около 90 % или более.

12. Сохраните модель TensorFlow как `SavedModel`:

```
model.save("indoor_scene_recognition")
```

Теперь модель готова к квантизации с помощью преобразователя TFLite.

Подготовка и тестирование квантизованной модели TFLite

Как мы знаем из главы 3 «*Создание метеостанции с помощью библиотеки TensorFlow Lite for Microcontrollers*», для более эффективной работы на микроконтроллере модель требует квантизации до 8 бит. Однако как мы узнаем, может ли модель поместиться в Arduino Nano?

Кроме того, как мы узнаем, сохраняет ли квантизованная модель достоверность варианта с плавающей запятой?

Ответы на эти вопросы будут даны в этом примере, где мы покажем, как оценить использование памяти программ и достоверность квантизованной модели, сгенерированной TFLite-конвертером. После анализа использования памяти и проверки достоверности мы преобразуем модель TFLite в C-байтовый массив.

Colab notebook под названием *prepare_model.ipynb* (см. в нем раздел «*Preparing and testing the quantized TFLite model*») содержит код, разбираемый в этом примере: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter05/ColabNotebooks/prepare_model.ipynb.

[72] Epochs (эпоха) – одна из мер охвата данных при машинном обучении. Одна эпоха означает, что весь набор данных один раз прошел через нейронную сеть. Подробнее см.: https://neurohive.io/ru/osnovy-data-science/jepoha-razmer-batcha-iteracija/.

Подготовка

Чтобы избежать неприятных сюрпризов при развертывании модели на целевом устройстве, всегда следует оценивать требования модели к памяти и достоверности. Например, C-байтовый массив, сгенерированный из модели TFLite, обычно является постоянным объектом, хранящимся в программной памяти микроконтроллера. Однако объем памяти программы ограничен, и обычно он не превышает 1 Мбайт.

Однако потребность в памяти не единственная проблема, с которой мы можем столкнуться. Квантизация является эффективным методом уменьшения размера модели и значительного улучшения задержки. Однако применение арифметики с ограниченной точностью может изменить достоверность модели. По этой причине крайне важно оценить достоверность квантизованной модели, чтобы быть уверенным, что приложение работает так, как ожидалось. К сожалению, TFLite не предоставляет встроенной функции для оценки достоверности по тестовому набору данных. Следовательно, нам нужно будет пропустить квантизованную модель TFLite через интерпретатор Python TFLite с тестовым набором данных, чтобы проверить, насколько они правильно распознаются.

Как это делается...

Давайте начнем с сбора некоторых тестовых образцов с помощью модуля камеры OV7670. Вы можете выполнить те же действия, что описаны выше в разделе «*Создание набора данных для распознавания интерьеров помещений*». Вам просто нужно сделать несколько снимков (например, 10) для каждого выходного класса и создать zip-файл (*test_samples.zip*) с той же структурой папок, которая была у нас для обучающего набора данных (см. рис. 5.13).

Затем загрузите zip-файл в Colab и выполните следующие действия, чтобы оценить достоверность квантизованной модели и узнать ее размер.

1. Распакуйте архив *test_samples.zip*:

```
with zipfile.ZipFile("test_samples.zip", 'r') as zip_ref:
    zip_ref.extractall(".")
test_dir = "test_samples"
```

2. Измените размер тестовых изображений до 48×48 с помощью билинейной интерполяции:

```
test_ds = tf.keras.utils.image_dataset_from_directory(
          test_dir, interpolation="bilinear", image_size=(48, 48))
```

3. Измените масштаб значений пикселей с [0, 255] на [–1, 1]:

```
test_ds = test_ds.map(lambda x, y: (rescale(x), y))
```

4. Преобразуйте модель TensorFlow в формат TensorFlow Lite (FlatBuffers) с помощью инструмента TensorFlow Lite converter. Примените 8-битное квантизование ко всей модели, за исключением выходного слоя:

```
repr_ds = test_ds.unbatch()

def representative_data_gen():
  for i_value, o_value in repr_ds.batch(1).take(60):
    yield [i_value]

TF_MODEL = "indoor_scene_recognition"

converter = tf.lite.TFLiteConverter.from_saved_model(TF_MODEL)
converter.representative_dataset = tf.lite.
RepresentativeDataset(representative_data_gen)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_ops =
    [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8

tfl_model = converter.convert()
```

Преобразование выполняется таким же образом, как мы делали это в главе 3 «*Создание метеостанции с помощью библиотеки TensorFlow Lite for Microcontrollers*», за исключением типа выходных данных. В этом случае выходные данные сохраняются в формате с плавающей запятой, чтобы избежать деквантизации выходного результата.

5. Получите размер модели TFLite в байтах:

```
print(len(tfl_model), "bytes")
```

Сгенерированный объект TFLite (`tfl_model`) – это то, что мы развертываем на микроконтроллере, он содержит архитектуру модели и веса обучаемых слоев. Поскольку веса постоянны, модель TFLite может храниться в программной памяти микроконтроллера, а размер объекта `tfl_model` указывает использование памяти. Ожидаемый размер модели 627880 байт, что составляет примерно 63 % от общей памяти программ.

6. Инициализируйте интерпретатор TFLite:

```
interpreter = tf.lite.Interpreter(model_content=tfl_model)
interpreter.allocate_tensors()
```

К сожалению, TFLite не предлагает готовых функций для оценки достоверности модели, как его аналог TensorFlow. Поэтому нам требуется запустить квантизованную модель TensorFlow Lite на Python для оценки достоверности по тестовому набору данных. Интерпретатор Python TFLite отвечает за загрузку и выполнение модели TFLite.

7. Получите входные параметры квантизации:

```
i_details = interpreter.get_input_details()[0]
o_details = interpreter.get_output_details()[0]
i_quant = i_details["quantization_parameters"]
i_scale = i_quant['scales'][0]
i_zero_point = i_quant['zero_points'][0]
```

8. Оцените достоверность квантизованной модели TFLite:

```
test_ds0 = test_ds.unbatch()
num_correct_samples = 0
num_total_samples = len(list(test_ds0.batch(1)))

for i_value, o_value in test_ds0.batch(1):
  i_value = (i_value / i_scale) + i_zero_point
  i_value = tf.cast(i_value, dtype=tf.int8)
  interpreter.set_tensor(i_details["index"], i_value)
  interpreter.invoke()
  o_pred = interpreter.get_tensor(o_details["index"])[0]
  if np.argmax(o_pred) == o_value:
    num_correct_samples += 1
print("Accuracy:", num_correct_samples/num_total_samples)
```

9. Преобразуйте модель TFLite в C-байтовый массив с помощью команды `xxd`:

```
open("model.tflite", "wb").write(tflite_model)
!apt-get update && apt-get -qq install xxd
!xxd -c 60 -i model.tflite > indoor_scene_recognition.h
```

   Команда генерирует заголовочный файл C, содержащий модель TensorFlow Lite в виде массива символов без знака. Поскольку веб-редактор Arduino обрезает файлы C, превышающие 20 000 строк, мы рекомендуем передать в `xxd` параметр `-c 60`. Этот параметр увеличивает количество символов в строке с 16 (по умолчанию) до 60, чтобы в файле было примерно 10 500 строк.

Теперь вы можете загрузить файл *indoor_scene_recognition.h* с левой панели Colab.

Сокращение объема RAM за счет объединения функций обрезки, изменения размера, масштабирования и квантизации

В этом последнем примере мы развернем приложение на Arduino Nano. Однако для распознавания интерьеров с помощью нашего малого устройства требуется несколько дополнительных операторов.

В этом примере мы узнаем, как объединить операторы обрезки, изменения размера, масштабирования и квантизации, чтобы уменьшить использование оперативной памяти. Эти дополнительные операторы понадобятся для подготовки входных данных модели TFLite.

Скетч Arduino *07_indoor_scene_recognition.ino*, содержащий код, разбираемый в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter05/ArduinoSketches/07_indoor_scene_recognition.ino.

Подготовка

Чтобы подготовиться к этому примеру, нужно знать, какие части приложения влияют на использование оперативной памяти.

На использование оперативной памяти влияют переменные, выделяемые во время выполнения программы, такие как входные, выходные и промежуточные тензоры модели ML. Однако модель не несет исключительной ответственности за использование памяти. Чтобы обеспечить соответствующие входные данные для модели, изображение, полученное с камеры OV7670, необходимо обработать с помощью следующих операций.

1. Преобразовать цветовой формат из YCbCr422 в RGB888 (*color convertion*).
2. Обрезать кадр камеры (*camera frame*) в соответствии с соотношением сторон входной формы модели TFLite (*crop*).
3. Изменить размер рамки камеры в соответствии с ожидаемой входной формой для модели TFLite (*resize*).
4. Изменить масштаб значений пикселей с [0, 255] на [–1, 1] (*rescale*).
5. Квантизовать значения пикселей с плавающей запятой (*quantize*).

Каждая из перечисленных операций считывает значения из буфера и возвращает результат вычисления в новом виде, как показано на следующем рисунке:

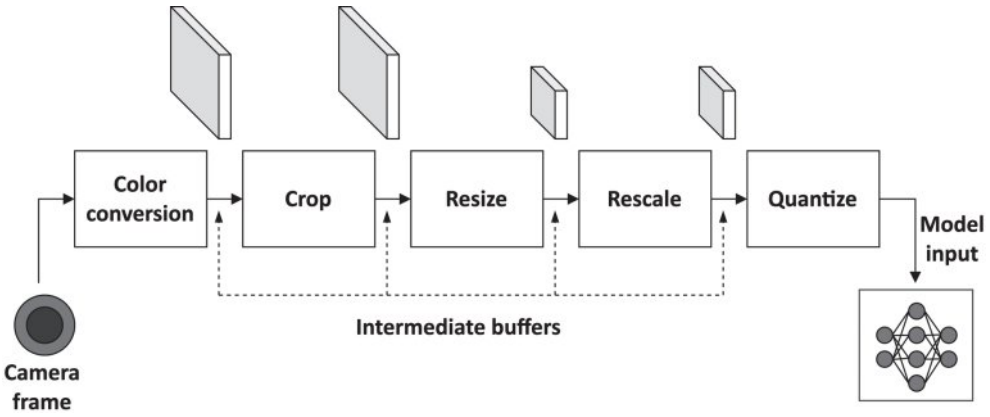


Рис. 5.18 ❖ Конвейер подготовки входного сигнала

Следовательно, на использование оперативной памяти также влияет формат кадра камеры и промежуточные буферы, передаваемые от одной операции к следующей.

Наша цель состоит в том, чтобы выполнить конвейер обработки, описанный ранее, используя как можно меньшие промежуточные буферы.

Для достижения этой цели данные, распространяемые по всему конвейеру, должны представлять часть всех входных данных, подлежащих обработке. При использовании этого метода, обычно называемого **operator fusion**

(*сращивания*), кадр камеры будет единственным значительным фрагментом памяти, который будет находиться в оперативной памяти в дополнение к входным, выходным и промежуточным тензорам модели TFLite.

Прежде чем показать, как выполнить окончательный пример, давайте посмотрим более подробно, как реализовать изменение размера.

Изменение размера с помощью билинейной интерполяции

Изменение размера – это функция обработки изображений, используемая для изменения разрешения изображения (ширины *width* и высоты *height*), как показано на следующем рисунке:

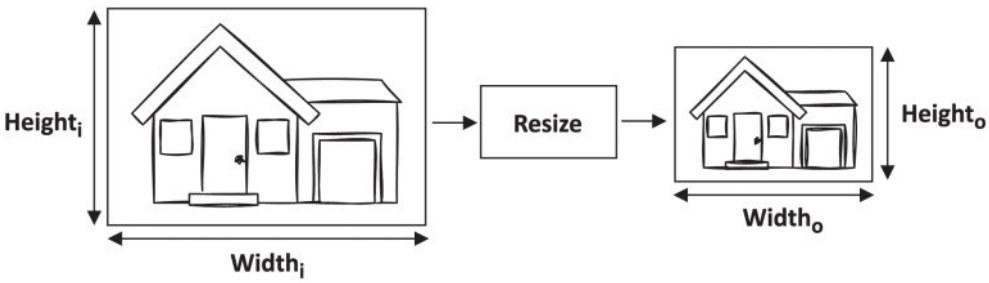


Рис. 5.19 ❖ Операция изменения размера

Результирующее изображение создается из пикселей входного изображения. Как правило, для сопоставления пространственных координат выходных пикселей с соответствующими входными применяются следующие формулы:

$$x_i = x_o \cdot ScaleX \quad ScaleX = \frac{width_i}{width_o};$$

$$y_i = y_o \cdot ScaleY \quad ScaleY = \frac{height_i}{height_o}.$$

В этих формулах:

- x_i, y_i – пространственные координаты входного пикселя;
- x_o, y_o – пространственные координаты выходного пикселя;
- $width_i$, $height_i$ – размеры входного изображения;
- $width_o$, $height_o$ – размеры выходного изображения.

Как мы знаем, цифровое изображение – это сетка пикселей. Однако, применяя эти две формулы, мы не всегда получаем целочисленную пространственную координату, т. е. фактический входной образец отличается от оригинала. Это одна из причин, по которой качество изображения ухудшается всякий раз, когда мы меняем разрешение изображения. Существуют некоторые методы интерполяции, облегчающие проблему, такие как интерполяция по ближайшему соседу (**nearest-neighbor**), билинейная (**bilinear**) или бикубическая (**bicubic**).

Билинейная интерполяция – метод, используемый в этом примере для улучшения качества изображения с измененным размером. Как показано на следующем рисунке, этот метод использует четыре ближайших пикселя к выбранной точке в сетке 2×2:

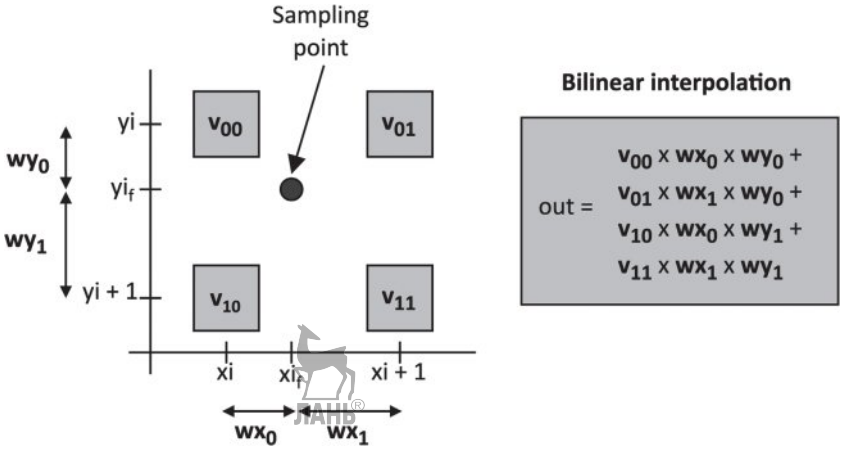


Рис. 5.20 ❖ Билинейная интерполяция

Функция интерполяции вычисляет выходной пиксель с использованием средневзвешенного значения четырех ближайших пикселей к выбранной точке образца в соответствии с формулой на рисунке.

Мы показали пример билинейной интерполяции, примененной к одноцветному компонентному изображению. Однако этот метод работает независимо от количества цветовых компонентов, поскольку мы можем интерполировать значения цветов независимо друг от друга.

Как это делается...

Отсоедините USB-кабель от Arduino Nano и снимите кнопку с макетной платы. После этого откройте Arduino IDE и скопируйте скетч, разработанный в примере «*Преобразование изображений QQVGA из YCbCr422 в RGB888*», в новый проект. Затем импортируйте заголовочный файл *indoor_scene_recognition.h* в Arduino IDE.

В примере удалите код из функции `loop()` и все ссылки на использование кнопки.

Ниже приведены необходимые шаги для распознавания интерьера с помощью Arduino Nano.

1. Включите файл заголовка *indoor_scene_recognition.h*:

```
#include "indoor_scene_recognition.h"
```

2. Включите заголовочные файлы для использования среды выполнения TFLu:

```
#include <TensorFlowLite.h>
#include <tensorflow/lite/micro/all_ops_resolver.h>
#include <tensorflow/lite/micro/micro_error_reporter.h>
#include <tensorflow/lite/micro/micro_interpreter.h>
#include <tensorflow/lite/schema/schema_generated.h>
#include <tensorflow/lite/version.h>
```

Заголовочные файлы – те же самые, что описаны в главе 3 «*Создание метеостанции с помощью библиотеки TensorFlow Lite for Microcontrollers*».

3. Объявите переменные, связанные с инициализацией TFLu, как глобальные:

```
const tflite::Model* tflu_model = nullptr;
tflite::MicroInterpreter* tflu_interpreter = nullptr;
TfLiteTensor* tflu_i_tensor = nullptr;
TfLiteTensor* tflu_o_tensor = nullptr;
tflite::MicroErrorReporter tflu_error;
constexpr int tensor_arena_size = 144000;
uint8_t *tensor_arena = nullptr;
float tflu_scale = 0.0f;
int32_t tflu_zeropoint = 0;
```

4. Переменные, указанные в приведенном коде, являются теми же самыми, что использовались в главе 3 «*Создание метеостанции с помощью библиотеки TensorFlow Lite for Microcontrollers*», единственным исключением являются параметры выходной квантизации, поскольку в данном случае они не требуются. Размер `tensor_arena` (памяти, требуемой интерпретатором TFLu) установлен равным 144 000 для размещения входных, выходных и промежуточных тензоров модели TFLite.
5. Объявите и инициализируйте разрешения обрезанного кадра камеры и входной формы в качестве глобальных переменных:

```
int height_i = 120; int width_i = hi;
int height_o = 48; int width_o = 48;
```

Поскольку мы обрезаем кадр перед изменением его размера, мы можем упростить обрезку, взяв квадратную область, соответствующую высоте кадра камеры с левой стороны.

6. Объявите и инициализируйте коэффициенты масштабирования разрешения для изменения размера кадра камеры в качестве глобальных переменных:

```
float scale_x = (float)width_i / (float)width_o;
float scale_y = scale_x;
```

7. Напишите функцию вычисления билинейной интерполяции для пикселя с одним цветовым компонентом:

```
uint8_t bilinear_inter(uint8_t v00, uint8_t v01,
                       uint8_t v10, uint8_t v11,
                       float xi_f, float yi_f,
                       int xi, int yi) {
```

```
    const float wx1 = (xi_f - xi);
    const float wx0 = (1.f - wx1);
    const float wy1 = (yi_f - yi);
    const float wy0 = (1.f - wy1);
    return clamp_0_255((v00 * wx0 * wy0) + (v01 * wx1 * wy0) +
                       (v10 * wx0 * wy1) + (v11 * wx1 * wy1));
}
```

Приведенная функция вычисляет веса на основе расстояния и применяет формулу билинейной интерполяции, описанную в разделе «*Подготовка*» этого примера.

8. Напишите функцию для масштабирования значений пикселей с [0, 255] на [−1, 1]:

```
float rescaling(float x, float scale, float offset) {
  return (x * scale) - offset;
}
```

Затем напишите функцию для квантизации входного изображения:

```
int8_t quantize(float x, float scale, float zero_point) {
  return (x / scale) + zero_point;
}
```

Поскольку масштабирование и квантизация выполняются одно за другим, вы можете подумать над объединением их в одной функции, чтобы сделать реализацию более эффективной с точки зрения выполняемых арифметических инструкций.

9. В функции `setup()` динамически выделите память для `tensor_arena`:

```
tensor_arena = (uint8_t *)malloc(tensor_arena_size);
```

Мы выделяем `tensor_arena` с помощью функции `malloc()`, чтобы поместить память в кучу (**heap**). Как мы знаем, куча – это область оперативной памяти, связанная с динамической памятью, и может быть освобождена пользователем только явно с помощью функции `free()`. Куча противоположна стековой памяти, где время жизни данных ограничено областью действия.
Размеры стека и кучи памяти определяются в коде установок, выполняемых микроконтроллером при сбросе системы. Поскольку стек обычно намного меньше кучи, предпочтительно выделить рабочее пространство TFLu в куче, потому что `tensor_arena` занимает значительную часть оперативной памяти (144 Кбайт).

10. Загрузите модель *indoor_scene_recognition*, инициализируйте интерпретатор TFLu и разместите тензоры:

```
  tflu_model = tflite::GetModel(indoor_scene_recognition);
  tflite::AllOpsResolver tflu_ops_resolver;

tflu_interpreter = new tflite::MicroInterpreter(tflu_model,
flu_ops_resolver, tensor_arena, tensor_arena_size, &tflu_error);

tflu_interpreter->AllocateTensors();
```

Затем получите указатели на входные и выходные тензоры:

```
  tflu_i_tensor = tflu_interpreter->input(0);
  tflu_o_tensor = tflu_interpreter->output(0);
```

Наконец, получите входные параметры квантизации:

```
  const auto* i_quantization =
    reinterpret_cast<TfLiteAffineQuantization*>(
    tflu_i_tensor->quantization.params);
  tflu_scale = i_quantization->scale->data[0];
  tflu_zeropoint = i_quantization->zero_point->data[0];
}
```

11. Выполните итерацию по пространственным координатам входной формы MobileNet v2 в функции `loop()`. Затем вычислите соответствующее положение точки выборки для каждой выходной координаты. Затем округлите до ближайшего целого значения координату точки выборки:

```
for (int yo = 0; yo < height_o; yo++) {
  float yi_f = (yo * scale_y);
  int yi = (int)std::floor(yi_f);
  for(int xo = 0; xo < width_o; xo++) {
    float xi_f = (xo * scale_x);
    int xi = (int)std::floor(xi_f);
```

Как вы можете видеть из кода, мы повторяем пространственные координаты входной формы MobileNet v2 (48×48). Для каждого `x0` и `y0` мы вычисляем положение выборки (`xi_f` и `yi_f`) в кадре камеры, необходимое для операции изменения размера. Поскольку мы применяем билинейную интерполяцию для изменения размера изображения, мы округляем `xi_f` и `yi_f` до ближайшего целого числа, чтобы получить пространственные координаты верхнего левого пикселя в сетке выборок 2×2.
Как только у вас будут входные координаты, вычислите смещения буфера камеры, чтобы прочитать четыре пикселя YCbCr422, необходимые для билинейной интерполяции:

```
    int x0 = xi;
    int y0 = yi;
    int x1 = std::min(xi + 1, width_i - 1);
    int y1 = std::min(yi + 1, height_i - 1);
    int stride_in_y = Camera.width() * bytes_per_pixel;
    int ix_y00 = x0 * sizeof(int16_t) + y0 * stride_in_y;

    int ix_y01 = x1 * sizeof(int16_t) + y0 * stride_in_y;
    int ix_y10 = x0 * sizeof(int16_t) + y1 * stride_in_y;
    int ix_y11 = x1 * sizeof(int16_t) + y1 * stride_in_y;
```

12. Считайте компонент `Y` для каждого из четырех пикселей:

```
    int Y00 = data[ix_y00];
    int Y01 = data[ix_y01];
```

```
int Y10 = data[ix_y10];
int Y11 = data[ix_y11];
```

Далее прочтите красный цветоделеный компонент (Cr):

```
int offset_cr00 = xi % 2 == 0? 1 : -1;
int offset_cr01 = (xi + 1) % 2 == 0? 1 : -1;
int Cr00 = data[ix_y00 + offset_cr00];
int Cr01 = data[ix_y01 + offset_cr01];
int Cr10 = data[ix_y10 + offset_cr00];
int Cr11 = data[ix_y11 + offset_cr01];
```

Далее прочтите синий цветоделеный компонент (Cb):

```
int offset_cb00 = offset_cr00 + 2;
int offset_cb01 = offset_cr01 + 2;
int Cb00 = data[ix_y00 + offset_cb00];
int Cb01 = data[ix_y01 + offset_cb01];
int Cb10 = data[ix_y10 + offset_cb00];
int Cb11 = data[ix_y11 + offset_cb01];
```

13. Преобразуйте пиксели YCbCr422 в RGB888:

```
uint8_t rgb00[3], rgb01[3], rgb10[3], rgb11[3];
ycbcr422_rgb888(Y00, Cb00, Cr00, rgb00);
ycbcr422_rgb888(Y01, Cb01, Cr01, rgb01);
ycbcr422_rgb888(Y10, Cb10, Cr10, rgb10);
ycbcr422_rgb888(Y11, Cb11, Cr11, rgb11);
```

14. Выполните итерацию RGB-пикселей по каналам:

```
uint8_t c_i; float c_f; int8_t c_q;
for(int i = 0; i < 3; i++) {
```

Для каждого цветового компонента примените билинейную интерполяцию:

```
  c_i = bilinear(rgb00[i], rgb01[i], rgb10[i],
                 rgb11[i], xi_f, yi_f, xi, yi);
```

Затем измените масштаб и квантизуйте цветовой компонент:

```
  c_f = rescale((float)c, 1.f/255.f, -1.f);
  c_q = quantize(c_f, tflu_scale, tflu_zeropoint);
```

В конце сохраните квантизованный цветовой компонент во входном тензоре модели TFLite и закройте цикл `for`, выполняющий итерацию по пространственным координатам входной формы MobileNet v2:

```
    tflu_i_tensor->data.int8[idx++] = c_q;
  }
 }
}
```

15. Запустите просчет модели с выводом результата классификации по последовательному каналу:

```
TfLiteStatus invoke_status = tflu_interpreter->Invoke();
  size_t ix_max = 0;
  float pb_max = 0;
  for (size_t ix = 0; ix < 3; ix++) {
    if(tflu_o_tensor->data.f[ix] > pb_max) {
      ix_max = ix;
      pb_max = tflu_o_tensor->data.f[ix];
    }
  }
  const char *label[] = {"bathroom", "kitchen", "unknown"};
  Serial.println(label[ix_max]);
```

Скомпилируйте и загрузите скетч на Arduino Nano. Теперь приложение должно распознать ваши комнаты и сообщить результат через монитор последовательного порта!

Глава 6

Создание интерфейса на основе жестов для управления воспроизведением на YouTube

Распознавание жестов – это технология, которая интерпретирует человеческие жесты, позволяя людям взаимодействовать со своими устройствами, не прикасаясь к кнопкам или дисплеям. Эта технология в настоящее время используется в различной бытовой электронике (например, смартфонах и игровых консолях) и включает в себя два основных компонента: датчик и программный алгоритм.

В этой главе мы покажем вам, как использовать измерения акселерометра в сочетании с машинным обучением (ML) для распознавания трех жестов рук с помощью Raspberry Pi Pico. Эти распознанные жесты затем будут использоваться для включения воспроизведения и паузы, отключения или включения звука и переключения видео с YouTube на нашем компьютере.

Мы начнем со сбора данных акселерометра для создания набора данных распознавания жестов. В этой части мы узнаем, как работать с **протоколом I2C** и использовать инструмент **Edge Impulse data forwarder**. Далее мы сосредоточимся на создании вычислительного блока *Impulse* (см. главу 4), где построим полносвязную нейронную сеть для распознавания жестов, основанную на спектральных характеристиках.

Наконец, мы развернем модель на Raspberry Pi Pico и внедрим программу на Python с **PyAutoGUI** для создания бесконтактного интерфейса воспроизведения видео с YouTube.

Цель этой главы – помочь вам разработать комплексное приложение для распознавания жестов при помощи Edge Impulse и Raspberry Pi Pico, а также научить вас использовать периферийные устройства с I2C-интерфейсом, познакомить с инерциальными датчиками, показать, как написать много-

поточную программу в **Arm Mbed OS** и узнать, как отфильтровывать избыточные результаты классификации во время расчета модели.

В этой главе мы рассмотрим следующие примеры.

- Подключение к MPU-6050 IMU[73] через интерфейс I2C.
- Получение данных акселерометра.
- Построение набора данных с помощью инструмента пересылки данных Edge Impulse data forwarder.
- Разработка и обучение модели ML.
- Распознавание в реальном времени с помощью инструмента пересылки данных Edge Impulse data forwarder.
- Распознавание жестов на Raspberry Pi Pico в ОС Arm Mbed.
- Создание бесконтактного интерфейса с помощью PyAutoGUI.

Технические требования

Чтобы выполнить на практике все примеры, приведенные в этой главе, вам понадобится следующее:

- Raspberry Pi Pico,
- кабель micro-USB,
- беспаечная макетная плата половинного размера,
- 1 датчик MPU-6050 IMU,
- 4 соединительных провода-перемычки,
- ноутбук/ПК с Ubuntu 18.04+ или Windows 10 на x86-64.

Исходный код примеров из этой главы и дополнительные материалы доступны в папке *Chapter06* по адресу https://github.com/PacktPublishing/TinyML-Cookbook/tree/main/Chapter06.

Подключение к MPU-6050 IMU через интерфейс I2C

Формирование набора данных является основной частью любого проекта ML, поскольку он влияет на эффективность модели. Однако запись данных датчиков часто является в TinyML сложной задачей, поскольку для этого требуется низкоуровневое взаимодействие с аппаратным обеспечением.

В этом примере мы будем использовать **инерциальный измерительный блок** (**IMU**) **MPU-6050** для обучения основам, лежащим в основе обще-

[73] Inertial Measurement Unit, инерциальный измерительный блок, – устройство, объединяющее гироскоп (датчик угловой скорости) и акселерометр (датчик ускорения свободного падения). Часто к этим двум датчикам добавляют магнитометр (электронный компас), а также барометр, позволяющий оценить высоту над уровнем моря. Полный вариант IMU позволяет определять характеристики движения и абсолютное положение датчика в пространстве.

го протокола связи для многих датчиков: **Inter-Integrated Circuit**, **I2C**. К концу этого примера у нас будет скетч Arduino для считывания адреса MPU-6050.

Скетч Arduino *01_i2c_imu_addr.ino*, содержащий код, который будет разобран в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter06/ArduinoSketches/01_i2c_imu_addr.ino.

Подготовка

Для этого примера нам нужно знать, что такое датчик IMU и как получить его измерения с помощью протокола связи I2C.

Датчик IMU – это электронное устройство, которое способно измерять ускорения, угловые скорости и в некоторых случаях ориентацию своего корпуса с помощью комбинации встроенных датчиков. Это устройство лежит в основе многих технологий для определения местоположения и ориентации в различных отраслях промышленности, включая автомобильную, аэрокосмическую и бытовую электронику. Например, именно IMU позволяет экрану смартфона автоматически поворачиваться и включает варианты использования дополненной/виртуальной реальности (**augmented reality/virtual reality**, **AR/VR**).

В следующем подразделе представлена более подробная информация о MPU-6050 IMU.

Представляем MPU-6050 IMU

MPU-6050 (https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/)[74] – это IMU, который сочетает в себе 3-осевой акселерометр и 3-осевой гироскоп для измерения ускорений и угловой скорости тела. Это устройство существует на рынке уже несколько лет, и благодаря своей низкой стоимости и высокой эффективности оно по-прежнему является популярным выбором для электронных DIY-проектов, основанных на датчиках движения.

MPU-6050 IMU можно найти у различных дистрибьюторов, таких как Adafruit, Amazon, Pimoroni и PiHut, и он доступен в различных форм-факторах. В этом примере мы рассмотрим компактную плату, предлагаемую компанией Adafruit (https://learn.adafruit.com/mpu6050-6-dof-accelerometer-and-gyro/overview), которая может питаться от 3.3 В и не требует дополнительных электронных компонентов[75].

[74] Подробное описание датчика по-русски см., например: https://alexgyver.ru/arduino-mpu6050/.

[75] Так как датчик подключается только через I2C, то для использования в примере годится любая плата на основе MPU-6050 из доступных на отечественном рынке, например недорогая GY-521 (различные фирменные варианты, в том числе и рассматриваемый автором MPU-6050 6-DoF от Adafruit, могут быть существенно дороже, но их преимущества неочевидны).

К сожалению, модуль IMU поставляется с неприпаянными разъемами. Поэтому, если вы не знакомы с пайкой, рекомендуем прочитать следующее руководство:

https://learn.adafruit.com/adafruit-agc-electretmicrophone-amplifier-max9814/assembly.

MPU-6050 IMU взаимодействует с микроконтроллером по протоколу последовательной связи I2C. В следующем подразделе описываются некоторые из основных функций I2C, заслуживающих упоминания.

Связь с помощью I2C

I2C – это протокол связи, основанный на двух проводных линиях, обычно называемых SCL (тактовый сигнал) и SDA (сигнал передачи данных).

Протокол был построен таким образом, чтобы обеспечить связь между ведущим (*primary*) устройством (например, микроконтроллером) и многочисленными ведомыми (*secondary*) устройствами (например, датчиками). Каждое ведомое устройство идентифицируется постоянным 7-битным адресом.

Протокол I2C использует термины *master* и *slave*. В этой книге мы решили переименовать эти термины в *primary* и *secondary*, чтобы сделать язык более всесторонним и убрать ненужные ассоциации с рабством[76].

На следующей схеме показано, как подключаются ведущее и ведомое устройства:

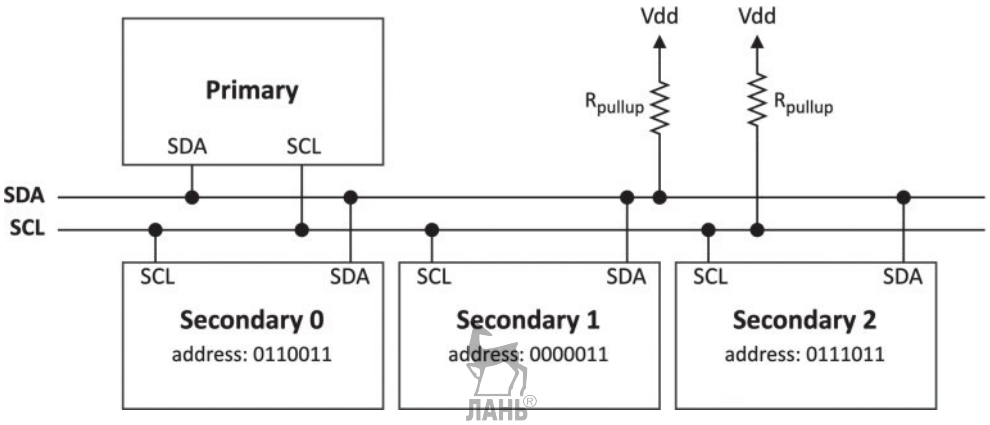


Рис. 6.1 ❖ Подключение по I2C

Как мы можем видеть, независимо от количества ведомых устройств существует только два сигнала (SCL и SDA). SCL генерируется только ведущим устройством и используется всеми устройствами I2C для выборки битов, ко-

[76] *Master* и *slave* в традиционном толковании означают «хозяин» и «раб». В русском переводе используемые автором термины *primary* (первичное, основное) и *secondary* (вторичное) заменены на общепринятые в нашей литературе «ведущее» и «ведомое» устройства.

торые передаются по линии данных. Как ведущее, так и ведомое устройства могут передавать данные по шине SDA.

Подтягивающие резисторы (R_{pullup}) требуются обязательно, поскольку устройство I2C может передавать сигнал только низкого уровня (логического уровня 0). В нашем случае подтягивающие резисторы не нужны, поскольку они встроены в плату MPU-6050. Согласно протоколу *сеанс связи всегда начинает ведущее устройство,* передавая в следующей последовательности:

1) 1 бит низкого уровня (логический уровень 0) на линии SDA (состояние **Start**),
2) 7-разрядный адрес целевого ведомого устройства,
3) 1 бит чтения или записи (флаг **R/W**). Логический уровень 0 указывает, что ведущее устройство будет отправлять данные через SDA (режим **записи W**). В противном случае логический уровень 1 означает, что ведущее устройство будет считывать данные, передаваемые ведомым устройством через SDA (режим **чтения R**).

На следующей схеме показан пример последовательности битовых команд в сценарии, когда ведущее устройство на рис. 6.1 начинает взаимодействовать с ведомым:

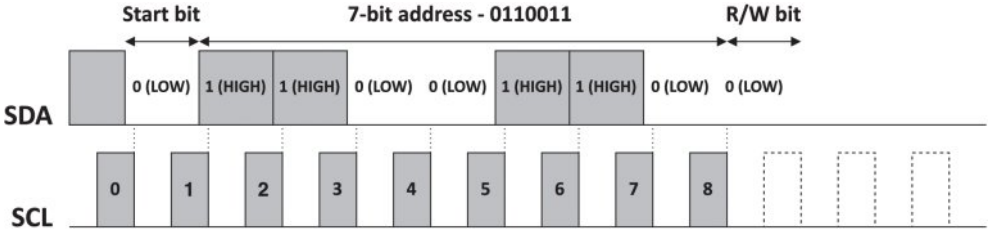


Рис. 6.2 ❖ Битовая последовательность команд, передаваемая ведущим устройством

Ведомое устройство, соответствующее 7-разрядному адресу, должно ответить одним битом с логическим уровнем 0 (**ACK**) по шине SDA.

Если ведомое устройство отвечает ACK, ведущее устройство может либо передавать, либо считывать данные порциями по 8 бит в соответствии с установленным флагом R/W.

В нашем контексте микроконтроллер является ведущим устройством, и он использует флаг R/W для выполнения следующего:

- **считывания данных с датчика**: микроконтроллер запрашивает то, что он хочет прочитать (режим записи), до того, как MPU-6050 IMU передаст данные (режим чтения);
- **программирования внутренней функции** IMU: микроконтроллер использует режим записи только для установки режима работы MPU-6050 (например, частоты дискретизации датчиков).

На этом этапе у вас может возникнуть вопрос: *что мы читаем и записываем с помощью ведущего устройства*?

Ведущее устройство считывает и записывает определенные регистры на ведомом устройстве. Таким образом, ведомое устройство работает как разновидность памяти, где каждый регистр имеет уникальный 8-битный адрес.

Карта регистров для MPU-6050 доступна по следующей ссылке:
https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf.

Как это делается...

Давайте начнем этот пример с того, что возьмем макетную плату на 30 рядов и 10 столбцов и установим Raspberry Pi Pico вертикально между левой и правой шинами питания. Мы должны разместить платформу микроконтроллера таким же образом, как мы это делали в главе 2 «*Прототипирование на микроконтроллерах*».

Затем поместите модуль датчика акселерометра в нижнюю часть макетной платы. Убедитесь, что вырез макетной платы находится посередине между двух разъемов, как показано на следующем рисунке:

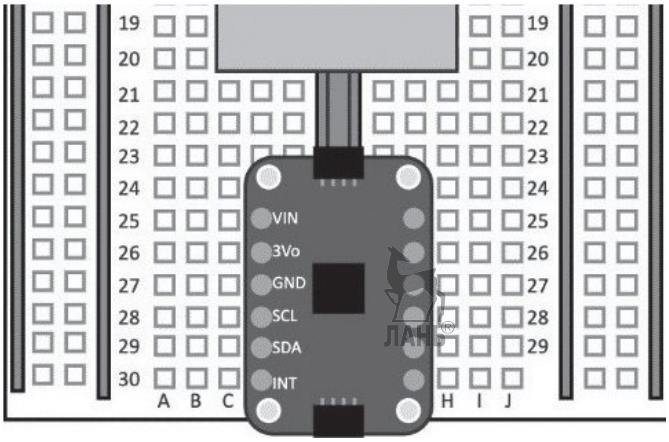


Рис. 6.3 ❖ MPU-6050, установленный в нижней части макетной платы

Как вы можете видеть, контакты I2C расположены на левом разъеме модуля MPU-6050.

Для подключения модуля акселерометра к Raspberry Pi Pico и написания базового скетча считывания идентификатора (адреса) устройства MPU-6050 необходимо выполнить следующие шаги.

1. Возьмите четыре провода-перемычки и подключите MPU-6050 IMU к Raspberry Pi Pico, как указано в следующей таблице:

Таблица 6.1. Соединения между MPU-6050 IMU и Raspberry Pi Pico

MPU-6050	VIN	GND	SCL	SDA
Raspberry Pi Pico	3V3	GND	GP7 (SCL1)	GP6 (SDA1)

Следующая схема должна помочь вам наглядно представить, как выполнить соединения:

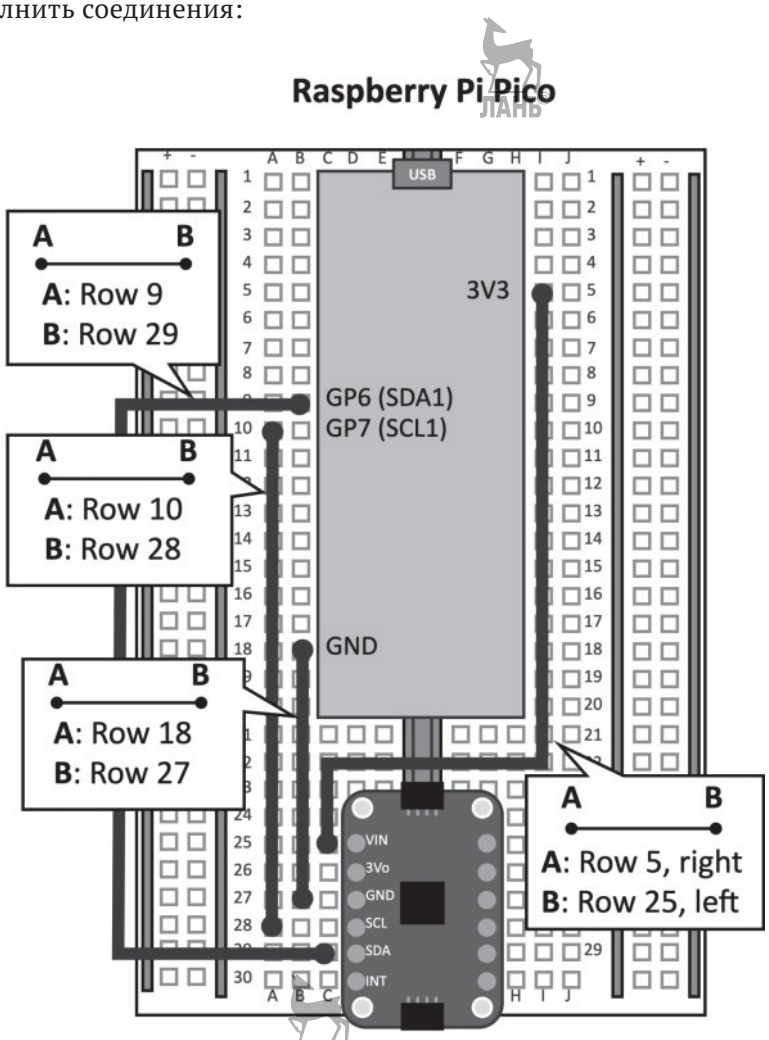


Рис. 6.4 ❖ Соединения между MPU-6050 IMU и Raspberry Pi Pico

Как мы упоминали в разделе «*Подготовка*» этого примера, нам не нужны подтягивающие резисторы на SDA и SCL, потому что они уже интегрированы в плату IMU.

2. Создайте новый скетч в Arduino IDE. Объявите и инициализируйте объект `mbed::I2C` с указанием контактов SDA и SCL:

```
#define I2C_SDA p6
#define I2C_SCL p7
I2C i2c(I2C_SDA, I2C_SCL);
```

Для инициализации периферийного устройства I2C требуется только указание контактов, предназначенных для шин SDA (`p6`) и SCL (`p7`).

3. Используйте C-определение, чтобы сохранить 7-разрядный адрес MPU-6050 IMU (0x68[77]):

```
#define MPU6050_ADDR_7BIT 0x68
```

4. Создайте вспомогательную функцию для считывания данных из регистра MPU-6050:

```
void read_reg(int addr_i2c, int addr_reg, char *buf, int length)
{
  char data = addr_reg;
  i2c.write(addr_i2c, &data, 1);
  i2c.read(addr_i2c, buf, length);
  return;
}
```

Согласно протоколу I2C нам нужно передать адрес устройства MPU-6050 IMU, а затем отправить адрес регистра для чтения. Итак, мы должны следующим образом использовать метод `write()` класса `mbed::I2C` с тремя входными аргументами:

- 8-разрядным адресом ведомого устройства (`addr_i2c`);
- массивом символов, содержащим адрес регистра (`char data = addr_reg`);
- количеством передаваемых байтов (1, поскольку мы отправляем только адрес регистра).

После отправки запроса на считывание данных из регистра мы с помощью метода `read()` класса `mbed::I2C` можем получить данные, переданные MPU-6050, для чего требуются следующие входные аргументы:

- 8-разрядный адрес ведомого устройства (`addr_i2c`);
- массив символов для хранения полученных данных (`buf`);
- размер этого массива (`length`).

Функция вернет результат, как только чтение будет завершено.

5. В функции `setup()` инициализируйте I2C на максимальной частоте, поддерживаемой MPU-6050 (400 кГц):

```
void setup() {
  i2c.frequency(400000);
```

6. В функции `setup()` используйте `read_reg()` для чтения регистра `WHO_AM_I` (адрес 0x75) из MPU-6050 IMU. Передайте сообщение «*MPU-6050 found*»

[77] I2C-адрес устройства обычно присваивается производителем и приводится в документации (см. ссылки на описания датчика MPU-6050 выше). Заметим, что для некоторых устройств имеются возможности по модификации адреса, обычно в пределах нескольких бит запаиванием или удалением перемычек.

по последовательному каналу, если регистр WHO_AM_I содержит 7-разрядный адрес устройства (0x68):

```
  #define MPU6050_WHO_AM_I 0x75
  Serial.begin(115600);
  while(!Serial);
  char id;
  read_reg(MPU6050_ADDR_8BIT, MPU6050_WHO_AM_I, &id, 1);
  if(id == MPU6050_ADDR_7BIT) {
    Serial.println("MPU-6050 found");
  } else {
    Serial.println("MPU-6050 not found");
    while(1);
  }
}
```

Скомпилируйте и загрузите скетч на Raspberry Pi Pico. Теперь вы можете открыть монитор последовательного порта из меню редактора. Если Raspberry Pi Pico может взаимодействовать с устройством MPU-6050, он передаст по последовательному каналу строку «*MPU-6050 found*».

Получение данных акселерометра

Акселерометр – один из наиболее распространенных датчиков, встроенных в IMU-устройства.

В этом примере мы разработаем приложение для считывания измерений акселерометра с MPU-6050 IMU с частотой 50 Гц. Затем измерения будут переданы по последовательному каналу, чтобы их можно было получить с помощью инструмента Edge Impulse data forwarder, описанного в следующем примере.

Скетч Arduino *02_i2c_imu_read_acc.ino*, содержащий код, который будет разобран в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter06/ArduinoSketches/02_i2c_imu_read_acc.ino.

Подготовка

Акселерометром называется датчик, измеряющий ускорения по одной, двум или трем пространственным осям, обозначаемым как X, Y и Z.

В этом и следующих примерах мы будем использовать 3-осевой акселерометр, встроенный в MPU-6050 IMU, для измерения ускорений в трех ортогональных направлениях.

Однако как работает акселерометр и как мы можем снимать измерения с датчика?

Давайте начнем с объяснения основного принципа работы этого датчика. Рассмотрим следующую систему, в которой масса прикреплена к пружине:

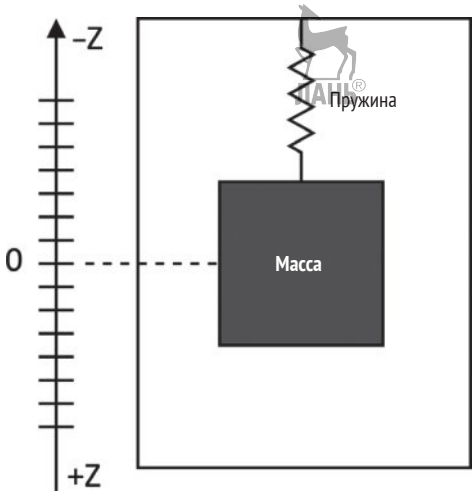


Рис. 6.5 ❖ Система масса–пружина

Рисунок иллюстрирует физический принцип работы акселерометра в одном пространственном измерении (т. е. 1-осевого акселерометра).

Что произойдет, если мы поставим акселерометр на стол?

В этом случае мы увидим, что масса идет вниз из-за постоянно действующей силы гравитации. Следовательно, указатель по оси Z сместится от положения покоя, как показано на следующем рисунке:

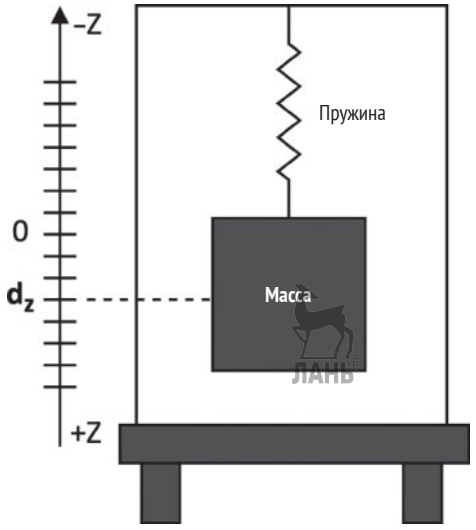


Рис. 6.6 ❖ Система масса–пружина под действием силы тяжести

Из уроков физики мы знаем, что закон Гука устанавливает для приложенной силы (силы реакции пружины):

$F = k \cdot d_z$.

Здесь F – сила, k – коэффициент упругости, а d_z – смещение.

Из второго закона Ньютона мы также знаем, что сила, приложенная к массе, равна:

$F = m \cdot a$.

Здесь F – сила, m – масса, а a – ускорение.

Приравнивая выражения $F = m \cdot a = k \cdot d_z$, мы можем сделать вывод, что смещение пружины d_z пропорционально ускорению a.

Следовательно, когда 1-осевой акселерометр помещается на стол, он возвращает значение ~ 9.81 м/с^2, что является ускорением объекта, когда он падает под действием силы тяжести. Ускорение 9.81 м/с^2 обычно обозначается символом g (9.81 м/с^2 = 1 g).

Как мы можем себе представить, пружина поднимается и опускается всякий раз, когда мы перемещаем акселерометр (даже слегка). Следовательно, смещение пружины – это физическая величина, получаемая датчиком для измерения ускорения.

Акселерометр, работающий в двух или трех пространственных измерениях, также может быть смоделирован с помощью системы масса–пружина. Например, 3-осевой акселерометр может быть смоделирован с использованием трех таких систем, так что каждая из них измеряет ускорение по своей оси.

Конечно, мы сделали некоторые упрощения при объяснении функциональности устройства. Основной механизм, основанный на системе масса–пружина, разработан из кремния с использованием технологии микроэлектромеханических систем (MEMS).

Большинство акселерометров имеет программируемый диапазон измерений (шкалу), который может варьироваться от ±1 g (±9.81 м/с^2) до ±250 g (±2452.5 м/с^2). Этот диапазон пропорционален чувствительности, которая обычно выражается как наименее значимый бит в отношении к g (LSB/g) и определяется как минимальное ускорение, вызывающее изменение числового представления. Чем выше чувствительность, тем меньше минимальное обнаруживаемое ускорение.

В IMU MPU-6050 мы можем запрограммировать диапазон измерений через регистр `ACCEL_CONFIG` (адрес 0x1C). В следующей таблице представлена соответствующая чувствительность для каждого значения:

Таблица 6.2. Диапазон измерений и чувствительность MPU-6050

Шкала (g)	**±2g**	**±4g**	**±4g**	**±16g**
Чувствительность (LSB/g)	16 384	8192	4096	2048
Значение регистра ACCEL_CONFIG	0x00	0x01	0x02	0x03

Как мы можем видеть, чем меньше диапазон измерений, тем выше чувствительность. Диапазон ±2g обычно достаточен для получения ускорений, обусловленных движениями рук.

Измерения, выполняемые IMU MPU-6050, представлены в виде 16-разрядного целого числа и хранятся в двух 8-разрядных регистрах. Названия этих двух регистров помечены суффиксами _H и _L для идентификации старших и младших байтов 16-разрядной переменной. На следующем рисунке показаны названия и адреса каждого регистра:

Address	Register
3B	ACCEL_XOUT_H
3C	ACCEL_XOUT_L
3D	ACCEL_YOUT_H
3E	ACCEL_YOUT_L
3F	ACCEL_ZOUT_H
40	ACCEL_ZOUT_L

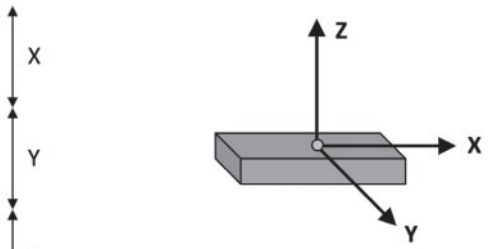


Рис. 6.7 ❖ Регистры для измерений акселерометра в MPU-6050 IMU

Как вы можете видеть, регистры размещаются по последовательным адресам памяти, начиная с `ACCEL_XOUT_H` по адресу 0x3B. Чтобы прочитать все измерения акселерометра без отправки адреса каждого регистра, мы можем просто получить доступ к `ACCEL_XOUT_H` и прочитать 6 байт подряд.

Как это делается...

Давайте продолжим работать над скетчем из предыдущего примера. Следующие шаги покажут вам, как расширить программу для считывания данных акселерометра с MPU-6050 IMU и передачи результатов измерений по последовательному каналу.

1. Создайте вспомогательную функцию для записи одного байта в регистр MPU-6050:

```
void write_reg(int addr_i2c, int addr_reg, char v) {
  char data[2] = {addr_reg, v};
  i2c.write(addr_i2c, data, 2);
  return;
}
```

   Как показано в предыдущем коде, мы используем метод `write()` класса `mbed::I2C` для передачи следующих сведений.
   I. Адрес MPU-6050.
   II. Адрес регистра для доступа.
   III. Байт количества читаемых регистров.
   Функция `write_reg()` потребуется для инициализации устройства MPU-6050.

2. Реализуйте вспомогательную функцию для считывания данных акселерометра с MPU-6050.
   Для этого создайте функцию с именем `read_accelerometer()` с тремя входными массивами с плавающей запятой:

```
void read_accelerometer(float *x, float *y, float *z) {
```

   Массивы x, y и z будут содержать выборку измеренных ускорений для трех ортогональных пространственных направлений.
3. В функции `read_accelerometer()` считайте измерения акселерометра с MPU-6050 IMU:

```
  char data[6];
  #define MPU6050_ACCEL_XOUT_H 0x3B
  read_reg(MPU6050_ADDR_8BIT, MPU6050_ACCEL_XOUT_H, data, 6);
```

   Затем объедините младший и старший байты каждого измерения, чтобы получить представление в 16-битном формате данных:

```
  int16_t ax_i16 = (int16_t)(data[0] << 8 | data[1]);
  int16_t ay_i16 = (int16_t)(data[2] << 8 | data[3]);
  int16_t az_i16 = (int16_t)(data[4] << 8 | data[5]);
```

   Получив эти 16-битные значения, разделите полученные цифры на чувствительность, присвоенную выбранному диапазону измерений, и умножьте ее на величину g (9.81 м/с^2). Затем сохраните ускорения в массивах x, y и z:

```
  const float sensitivity = 16384.f;
  const float k = (1.f / sensitivity) * 9.81f;
  *x = (float)ax_i16 * k;
  *y = (float)ay_i16 * k;
  *z = (float)az_i16 * k;
  return;
}
```

   Предыдущий код преобразует необработанные данные в числовое значение (м/с^2). Чувствительность составляет 16 384, поскольку диапазон MPU-6050 IMU установлен в ±2 g.
4. В функции `setup()` убедитесь, что MPU-6050 IMU не находится в спящем режиме:

```
#define MPU6050_PWR_MGMT_1 0x6B
#define MPU6050_ACCEL_CONFIG 0x1C
if (id == MPU6050_ADDR_7BIT) {
  Serial.println("MPU6050 found");
  write_reg(MPU6050_ADDR_8BIT, MPU6050_PWR_MGMT_1, 0);
```

   Когда IMU находится в спящем режиме, датчик не возвращает никаких измерений. Чтобы убедиться, что MPU-6050 IMU не находится в этом режиме, нам необходимо очистить шестой бит (бит 6) регистра `PWR_MGMT_1`. Это можно легко сделать, очистив этот бит напрямую.

5. В функции `setup()` установите диапазон измерения акселерометра MPU-6050 IMU на ±2 g:

```
  write_reg(MPU6050_ADDR_8BIT, MPU6050_ACCEL_CONFIG, 0);
}
```

6. В функции `loop()` выполните выборку измерений акселерометра с частотой 50 Гц (50 выборок 3-осевого акселерометра в секунду) и передайте их по последовательному каналу. Отправляйте данные по одной строке на каждое показание акселерометра и измерения по трем осям (`ax`, `ay` и `az`) через запятую:

```
#define FREQUENCY_HZ 50
#define INTERVAL_MS (1000 / (FREQUENCY_HZ + 1))
#define INTERVAL_US INTERVAL_MS * 1000
void loop() {
  mbed::Timer timer;
  timer.start();
  float ax, ay, az;
  read_accelerometer(&ax, &ay, &az);
  Serial.print(ax);
  Serial.print(",");
  Serial.print(ay);
  Serial.print(",");
  Serial.println(az);
  timer.stop();
  using std::chrono::duration_cast;
  using std::chrono::microseconds;
  auto t0 = timer.elapsed_time();
  auto t_diff = duration_cast<microseconds>(t0);
  uint64_t t_wait_us = INTERVAL_US - t_diff.count();
  int32_t t_wait_ms = (t_wait_us / 1000);
  int32_t t_wait_leftover_us = (t_wait_us % 1000);
  delay(t_wait_ms);
  delayMicroseconds(t_wait_leftover_us);
}
```

В приведенном коде мы сделали следующее.

I. Запустили `mbed::Timer` перед считыванием измерений акселерометра, чтобы установить время между получением отсчетов.
II. Считали ускорения с помощью функции `read_accelerometer()`.
III. Остановили `mbed::Timer` и восстановили прошедшее время в микросекундах (мкс).
IV. Подсчитали, сколько времени программе нужно подождать до следующего чтения показаний акселерометра. Этот шаг гарантирует частоту дискретизации 50 Гц.
V. Поставили программу на паузу.

Программа приостанавливается с помощью функции `delay()`, за которой следует `delayMicroseconds()`, по следующим причинам:

- `delay()` сама по себе была бы неточной, поскольку для этого таймера требуется входной аргумент в миллисекундах;

- `delayMicroseconds()` работает до 16 383 мкс, что недостаточно для частоты дискретизации 50 Гц (20 000 мкс).

Мы узнаем, сколько времени нужно ждать в миллисекундах, разделив `t_wait_us` на 1000. Затем мы вычисляем оставшееся время ожидания в микросекундах, вычисляя остаток от деления `t_wait_us % 1000`.

Формат, используемый для отправки данных акселерометра по последовательному каналу (по одной строке на считывание с разделением измерений по трем осям запятыми), будет необходим для выполнения задачи, представленной в следующем примере.

Скомпилируйте и загрузите код в Raspberry Pi Pico. Затем откройте монитор последовательного порта и проверьте, передает ли микроконтроллер измерения акселерометра. Если это так, положите макетную плату плашмя на стол. Ожидаемое ускорение для оси Z (третий отсчет каждого ряда) должно быть примерно равно ускорению под действием силы тяжести (9.81 м/с^2), в то время как ускорения для других осей должны быть примерно близки к нулю, как показано на следующем рисунке:

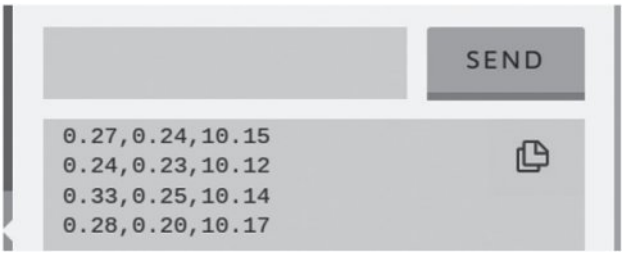


Рис. 6.8 ❖ Ускорения, отображаемые на мониторе последовательного порта Arduino

Как вы можете видеть, на ускорения могут влиять смещение и шум. Однако нам не нужно беспокоиться о точности измерений, потому что модель глубокого обучения все равно будет способна распознавать наши жесты.

Построение набора данных с помощью инструмента пересылки данных Edge Impulse data forwarder

Любому алгоритму ML нужен набор данных, для нас это означает получение отсчетов с акселерометра.

Запись данных акселерометра не так сложна, как может показаться на первый взгляд. Эта задача может быть легко выполнена с помощью Edge Impulse.

В этом примере мы будем использовать инструмент Edge Impulse data forwarder для фиксации показаний акселерометра при выполнении следующих трех движений макетной платой:

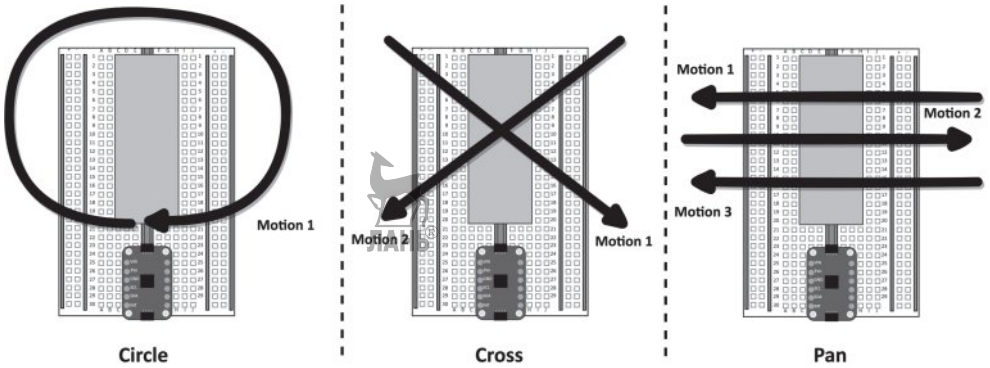


Рис. 6.9 ❖ Распознаваемые жесты – круг, крест и покачивание

Как показано на рисунке, мы должны убедиться, что макет расположен вертикально, Raspberry Pi Pico смотрит на нас, и тогда совершать движения, показанные стрелками.

Подготовка

Для получения адекватного набора данных для распознавания жестов требуется не менее 50 выборок для каждого выходного класса. Три жеста, которые мы рассмотрели для этого проекта, следующие:

- круг (*circle*): круговое перемещения платы по часовой стрелке,
- крест (*cross*): перемещение платы от верхнего левого угла в правый нижний, а затем из правого верхнего в нижний левый,
- покачивание (*pan*)[78]: перемещение платы горизонтально влево, затем вправо, а затем снова влево.

Каждый жест следует выполнять, располагая макет вертикально и с Raspberry Pi Pico, обращенным в вашу сторону. Поскольку мы будем рассматривать тренировочные выборки продолжительностью 2.5 с, мы рекомендуем выполнять каждое движение примерно за 2 с.

Хотя у нас есть три выходных класса для распознавания, требуется дополнительный, чтобы справиться с неизвестными движениями и для случая, когда нет жестов (например, макет, просто лежащий плашмя на столе).

В этом примере для создания набора данных мы будем использовать Edge Impulse data forwarder. Этот инструмент позволяет нам быстро получать значения ускорений с любого устройства, способного передавать данные по последовательному каналу для импорта выборок непосредственно в Edge Impulse.

---

[78] Название жеста *pan* имеет разное толкование для разных случаев. Здесь автор имеет в виду перемещение всего девайса справа налево и обратно (что можно назвать покачиванием), однако, например, для сенсорных экранов *pan* означает прокрутку (однократное движение пальца по вертикали сверху вниз / снизу вверх) или перелистывание (движение по горизонтали справа налево / слева направо). Есть и другие толкования.

Data forwarder будет запущен на вашем компьютере, поэтому вам нужно будет иметь установленный Edge Impulse CLI. Если вы еще не установили Edge Impulse CLI, мы рекомендуем следовать инструкциям в официальной документации: https://docs.edgeimpulse.com/docs/cli-installation.

Как это делается...

Скомпилируйте и загрузите на Raspberry Pi Pico скетч, который мы разработали в предыдущем примере. Убедитесь, что монитор последовательного порта Arduino закрыт; последовательный порт на вашем компьютере одновременно может взаимодействовать только с одним приложением.

Затем откройте Edge Impulse и создайте новый проект. Edge Impulse попросит вас написать название проекта. В нашем случае мы назвали проект *gesture_recognition*.

Чтобы создать набор данных с помощью инструмента Data forwarder, выполните следующие действия.

1. Запустите программу *edge-impulse-data-forwarder* на вашем компьютере с частотой 50 Гц и скоростью передачи 115 600 бод:

```
$ edge-impulse-data-forwarder -- frequency 50 --baud-rate115600
```

Программа Data forwarder попросит вас пройти аутентификацию в Edge Impulse, выбрать проект, над которым вы работаете, и дать вашему Raspberry Pi Pico имя (например, вы можете назвать его `pico`). Как только вы настроите инструмент, программа начнет анализировать данные, передаваемые по последовательному каналу. Протокол пересылки данных ожидает одну строку показаний для каждого измерения 3-осевого датчика, разделенных запятой или табуляцией, как показано на следующем рисунке:

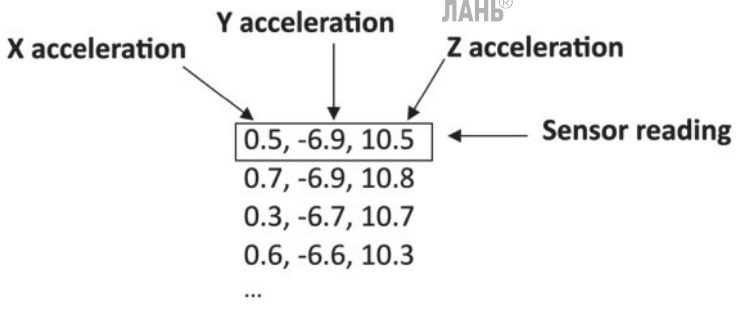


Рис. 6.10 ❖ Протокол пересылки данных

Поскольку скетч Arduino соответствует этому протоколу, программа пересылки данных обнаружит измерения по трем осям, которые передаются по последовательному каналу, и попросит вас присвоить им имя. Вы можете назвать их `ax`, `ay` и `az`.

2. Откройте Edge Impulse и перейдите на вкладку **Record new data** (*Записать новые данные*) в меню слева.
Используйте область **Record new data** для записи 50 образцов для каждого жеста (*circle*, *cross* и *pan*):

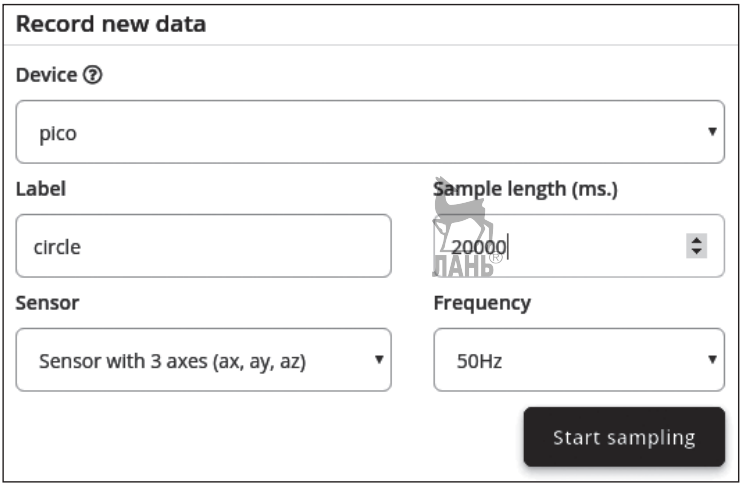


Рис. 6.11 ❖ Окно **Record new data** в Edge Impulse

В полях **Device** (*Устройство*) и **Frequency** (*Частота*) уже должно быть указано название устройства, подключенного к устройству пересылки данных (`pico`), а также частота дискретизации (50 Гц).
Для каждого жеста введите название метки в поле **Label** (например, `circle` для жеста *circle*) и продолжительность записи **Sample length** (в мс).
Так как продолжительность каждого отсчета составляет 2.5 с, вы получите 20 с данных, в которых вы повторяете одни и те же жесты несколько раз, как показано на следующем графике:

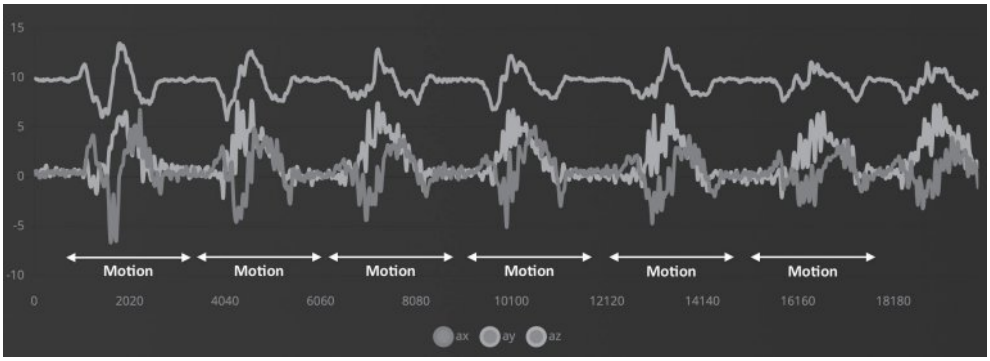


Рис. 6.12 ❖ Одна запись с несколькими однотипными движениями

Однако мы рекомендуем подождать одну или две секунды между движениями, чтобы помочь Edge Impulse распознать движения на следующем шаге.

3. Разделите запись на семплы продолжительностью 2.5 с, нажав на ⋮ рядом с именем файла, а затем нажав **Split sample**, как показано на следующем скриншоте:

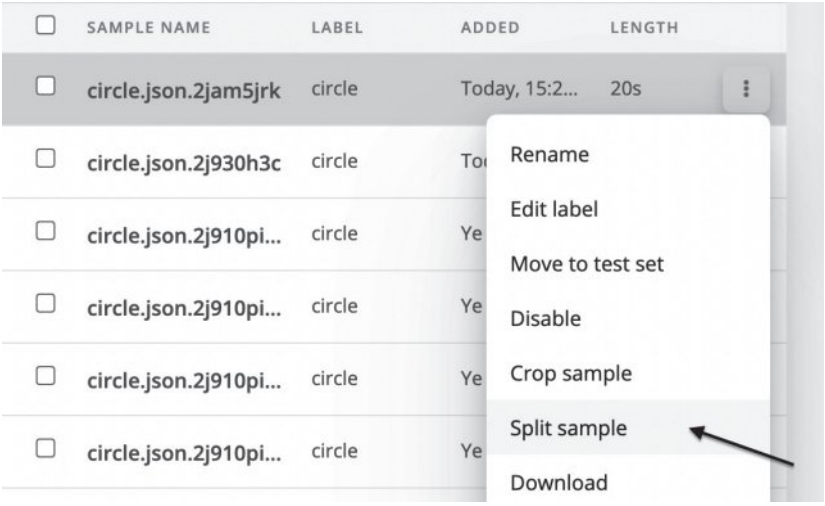


Рис. 6.13 ❖ Опция разделения выборки **Split sample** в Edge Impulse

Установите длину сегмента **segment length** (мс) равной 2500 (2.5 с) в новом окне и нажмите **Apply** (*Применить*). Edge Impulse обнаружит движения и установит окно продолжительностью 2.5 с для каждого из них:

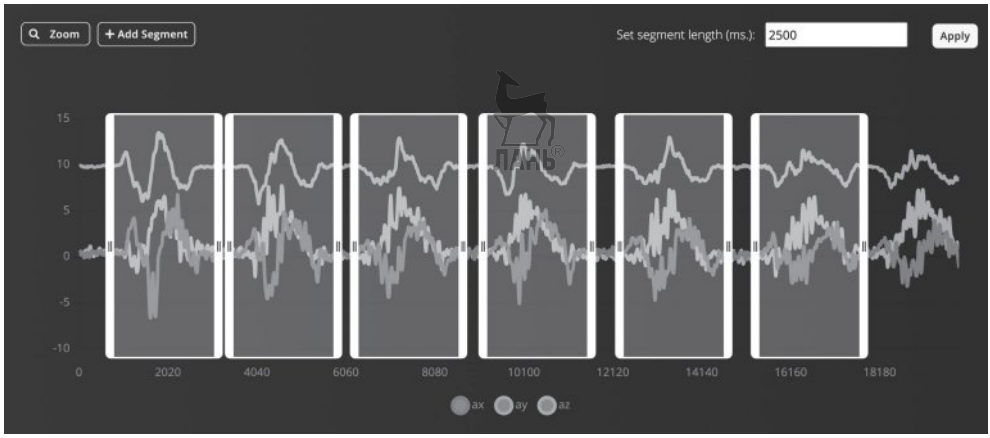


Рис. 6.14 ❖ Фрагменты выборки в окнах продолжительностью 2.5 с

Если Edge Impulse не распознает движение в записи, вы всегда можете добавить окно вручную, нажав кнопку **Add Segment** и щелкнув по области, которую вы хотите вырезать.

Как только все сегменты будут выбраны, нажмите **Split**, чтобы получить отдельные образцы.

4. Используйте ту же область **Record new data**, чтобы записать 50 случайных движений для формирования неизвестного класса. Для этого получите данные акселерометра за 40 с, когда вы произвольно перемещаете макет и кладете его плашмя на стол.
5. Разделите запись неизвестных жестов на семплы продолжительностью 2.5 с, нажав на : рядом с именем файла, а затем на **Split sample**. В новом окне добавьте 50 окон для вырезания и нажмите **Split**, когда закончите.
6. Разделите выборки между обучающим и тестовым наборами данных, нажав на кнопку **Perform train/test split** в области **Danger zone** на панели инструментов. Edge Impulse дважды спросит вас, уверены ли вы, что хотите выполнить это действие. Это связано с тем, что операция перетасовки данных является необратимой.

Теперь набор данных готов, 80 % выборок отнесены к набору для обучения/проверки и 20 % – к набору для тестирования.

Разработка и обучение модели ML

Имея в руках набор данных, мы можем приступить к проектированию модели.

В этом примере при помощи Edge Impulse мы разработаем следующую архитектуру:

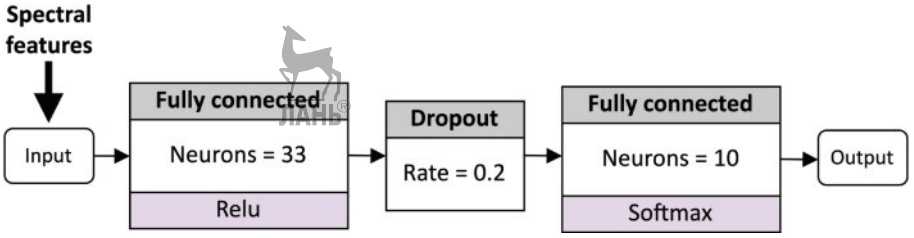


Рис. 6.15 ❖ Полносвязная нейронная сеть для обучения

Как вы можете видеть, для модели, состоящей всего из двух полносвязных слоев, входными данными являются спектральные характеристики.

Подготовка

В этом примере мы хотим объяснить, почему сжатая сеть, показанная на рис. 6.15, распознает жесты по данным акселерометра.

При разработке архитектур глубоких нейронных сетей мы обычно загружаем в модель необработанные данные, чтобы научить сеть извлекать характеристики автоматически.

Этот подход оказался эффективным в различных приложениях, таких как классификация изображений. Однако существуют некоторые приложения, в которых указанные вручную инженерные характеристики обеспечивают результаты, аналогичные результатам глубокого обучения, и помогают снизить сложность архитектуры. Это относится к распознаванию жестов, где мы можем использовать характеристики из частотной области.

Если вы не знакомы с анализом в частотной области, мы рекомендуем перечитать главу 4 «*Голосовое управление светодиодами с помощью Edge Impulse*».

Преимущества спектральных характеристик будут более подробно описаны в следующем подразделе.

Использование спектрального анализа для распознавания жестов

Спектральный анализ позволяет нам обнаружить характеристики сигнала, которые не видны во временной области. Например, рассмотрим следующие два сигнала:

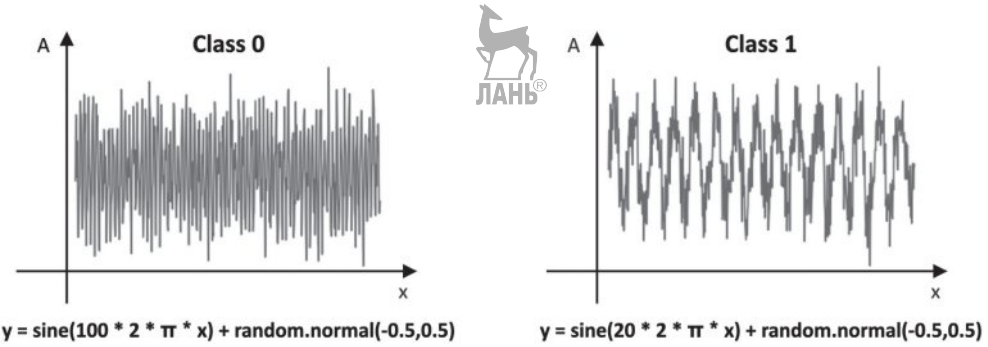


Рис. 6.16 ❖ Два сигнала во временной области

Эти два сигнала отнесены к двум разным классам: классу 0 и классу 1.

Какие параметры вы бы использовали во временной области, чтобы отличить класс 0 от класса 1?

Какой бы набор параметров вы ни рассматривали, они должны быть инвариантными к сдвигу и устойчивыми к шуму, чтобы быть эффективными. Хотя может существовать набор признаков, отличающих класс 0 от класса 1, решение было бы проще, если бы мы рассмотрели проблему в частотной области, как показано их спектрами на рис. 6.17.

Как мы можем видеть, два сигнала имеют разные доминирующие частоты, определяемые как компоненты с наибольшей амплитудой. Другими словами, доминирующие частоты – это компоненты, которые несут больше энергии.

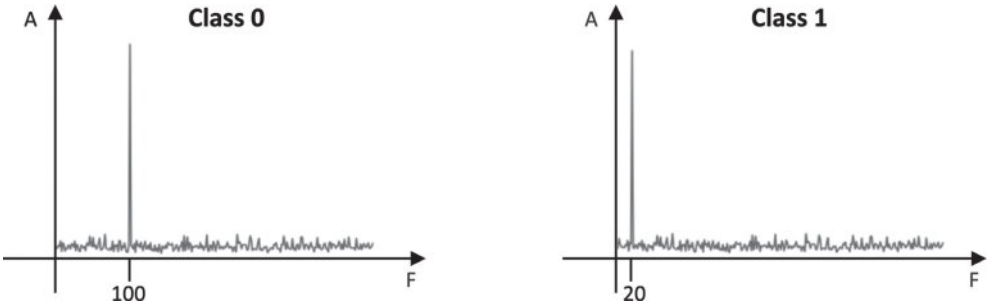


Рис. 6.17 ❖ Частотные представления сигналов класса 0 и класса 1

Хотя сигналы от акселерометра не совпадают с сигналами класса 0 и класса 1, они по-прежнему имеют повторяющиеся паттерны, которые делают частотные компоненты подходящими для задачи классификации.

Однако частотное представление также дает еще одно преимущество, связанное с возможностью получения сжатого представления исходного сигнала.

Например, давайте рассмотрим образцы нашего набора данных, которые представляют собой ускорения по трем осям, полученные нами с частотой 50 Гц в течение 2.5 с. Каждый экземпляр содержит 375 точек данных (125 точек данных на ось). Теперь давайте применим быстрое преобразование Фурье (БПФ) с 128 выходными частотами (длина БПФ) на каждом образце. Это преобразование создает 384 точки данных (128 точек данных на ось). Следовательно, БПФ, по-видимому, не меняет объем данных. Однако, как мы видели в предыдущем примере с классом 0 и классом 1, не все частоты приносят значимую информацию. Следовательно, мы могли бы просто извлечь частоты, которые получают наибольшее количество энергии (доминирующие частоты), чтобы уменьшить объем данных, а затем облегчить распознавание паттернов сигналов.

Для распознавания жестов мы обычно создаем спектральные характеристики, выполняя следующее.

1. Применение фильтра нижних частот к частотной области для отфильтровывания самых высоких частот. Этот шаг обычно делает извлечение объектов более устойчивым к шуму.
2. Извлечение частотных составляющих с наибольшей величиной. Обычно мы берем три частоты с самым высоким пиком.
3. Извлечение характеристик мощности в спектре мощности. Как правило, этими характеристиками являются среднеквадратичное значение (RMS) и спектральная плотность мощности (PSD), описывающие мощность, присутствующую в заданном интервале частот.

В нашем случае мы извлекаем следующие характеристики для каждой оси акселерометра:

- одно значение для среднеквадратичного значения;
- шесть значений для выделения частот с наибольшим пиком (три значения для частоты и три – для амплитуды);
- четыре значения для PSD.

Таким образом, мы получили бы только 33 функции, что означает сокращение объема данных более чем в 11 раз, по сравнению с исходным сигналом, чего достаточно для обучения сжатой полносвязной нейронной сети.

Как это делается...

Нажмите на вкладку **Create Impulse** в меню слева. В разделе **Create Impulse** установите размер окна **Window size** на 2500 мс и расширение окна **Window increase** до 200 мс.

Как мы видели в главе 4 «*Голосовое управление светодиодами с помощью Edge Impulse*», параметр **Window increase** необходим для выполнения расчета ML через регулярные промежутки времени. Этот параметр играет решающую роль в непрерывном потоке данных, поскольку мы не знаем, когда может начаться событие.

Следовательно, идея состоит в том, чтобы разделить поток входных данных на фиксированные окна (или сегменты) и запустить просчет модели ML для каждого из них. Размер окна – это продолжительность окна во времени, в то время как расширение окна – это временно́е расстояние между двумя последовательными отсчетами.

Следующие шаги покажут, как спроектировать нейронную сеть, показанную на рис. 6.17.

1. Нажмите кнопку **Add a processing block** (*Добавить блок обработки*) и найдите пункт **Spectral Analysis** (*Спектральный анализ*):

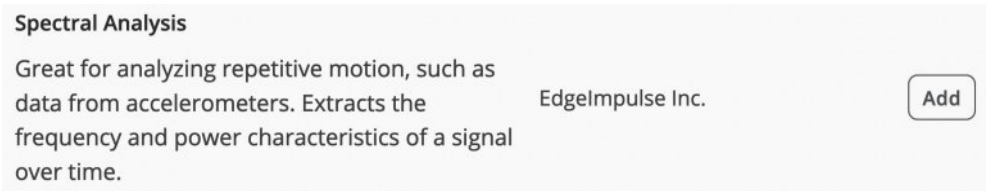


Рис. 6.18 ❖ Блок обработки с помощью спектрального анализа

   Нажмите кнопку **Add**, чтобы интегрировать блок обработки в *Impulse*.
2. Нажмите кнопку **Add a learning block** (Добавить обучающий блок) и добавьте **Classification (Keras)**.
   Блок **Output features** (*Выходные параметры*) должен сообщать о четырех выходных классах, которые мы должны распознать (*circle, cross, pan* и неизвестный *unknown*), как показано на рис. 6.19.
   Сохраните *Impulse*, нажав на кнопку **Save Impulse**.
3. Нажмите на кнопку **Spectral features** (*Спектральные характеристики*) из категории **Impulse design** (рис. 6.20).
   В новом окне мы можем поиграть с параметрами, влияющими на извлечение объектов, такими как следующие:

- **тип фильтра для применения к входному сигналу**: мы можем выбрать фильтр нижних или верхних частот, а затем установить частоту среза, т. е. частоту, на которой происходит затухание. Поскольку мы хотим отфильтровать вклад шума, мы должны использовать фильтр нижних частот;
- **параметры, влияющие на извлекаемые характеристики спектральной мощности**: включают в себя длину FFT, количество частотных составляющих с наибольшим пиком для извлечения и границы мощности, которые требуются для PSD.

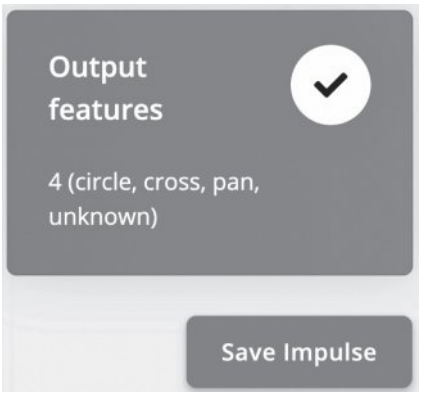


Рис. 6.19 ❖ Выходные классы

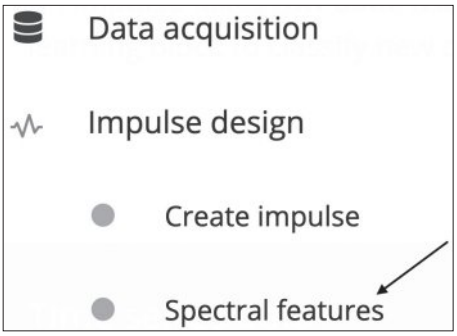


Рис. 6.20 ❖ Кнопка **Spectral features**

Мы можем сохранить все параметры в их значениях по умолчанию и нажать на кнопку **Generate features** (*Генерировать параметры*), чтобы извлечь спектральные параметры из каждой обучающей выборки. Когда процесс извлечения завершится, Edge Impulse вернет сообщение о завершении задания «*Job completed*» в выходных логах.

4. Нажмите на кнопку **Neural Network (Keras)** в разделе **Impulse design** и добавьте слой **Dropout layer** с соотношением 0.2 между полносвязными слоями. Убедитесь, что первый полносвязный слой содержит 33 нейрона, в то время как другой имеет 10 нейронов, как показано на рис. 6.21.
   Установите количество периодов обучения равным 100 и начните обучение, нажав кнопку **Start training**.
   Консоль вывода будет сообщать о достоверности и потерях по наборам данных обучения и проверки во время обучения после каждого периода.

Теперь давайте оценим производительность модели на тестовом наборе данных. Для этого нажмите кнопку тестирования модели **Model testing** на левой панели, а затем нажмите **Classify all**.

Edge Impulse отобразит процесс в выходных данных тестирования **Model testing output** и сгенерирует матрицу ошибок после его завершения:

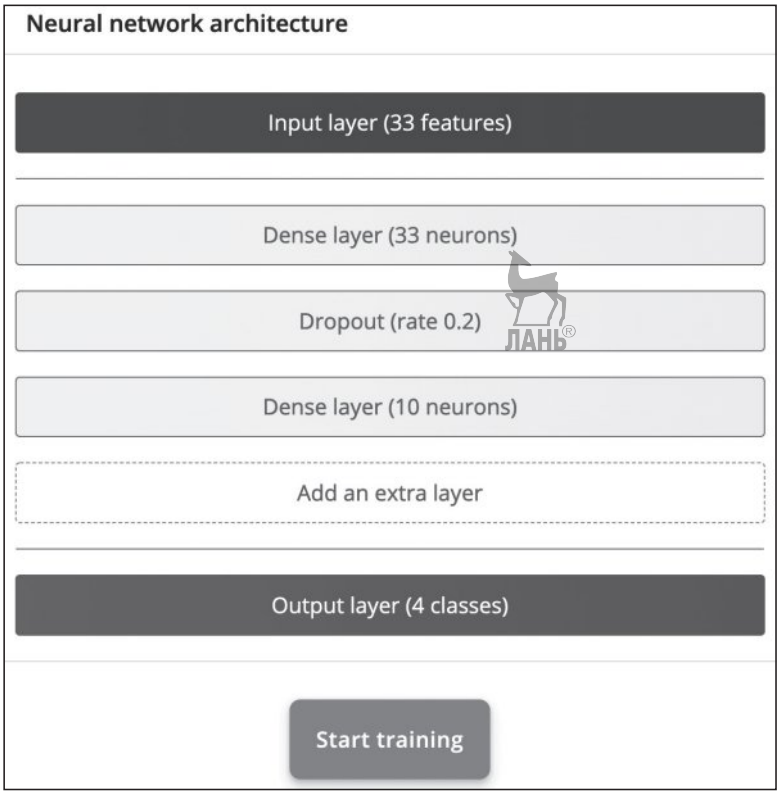
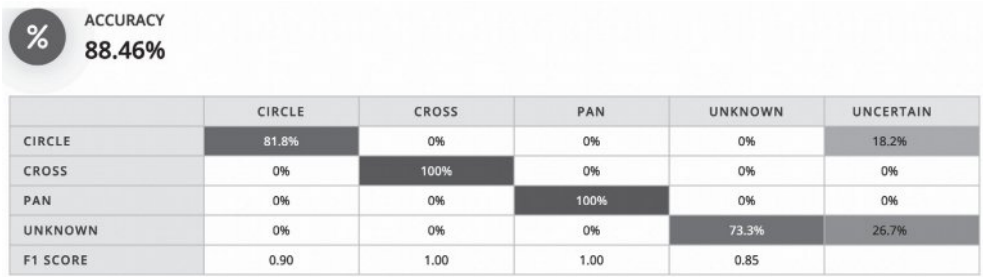


Рис. 6.21 ❖ Архитектура нейронной сети



	CIRCLE	CROSS	PAN	UNKNOWN	UNCERTAIN
CIRCLE	81.8%	0%	0%	0%	18.2%
CROSS	0%	100%	0%	0%	0%
PAN	0%	0%	100%	0%	0%
UNKNOWN	0%	0%	0%	73.3%	26.7%
F1 SCORE	0.90	1.00	1.00	0.85	

Рис. 6.22 ❖ Результаты тестирования модели

Как вы можете видеть, наша уменьшенная модель, состоящая всего из двух полносвязных слоев, достигла достоверности 88 %!

Классификации в реальном времени с помощью инструмента пересылки данных Edge Impulse data forwarder

Тестирование модели – это шаг, который мы всегда должны предпринять перед экспортом конечного приложения на целевую платформу. Модель после развертывания на микроконтроллерах может давать неверные результаты, поскольку код может содержать ошибки, подключения могут быть выполнены некорректно или модель может ненадежно работать в полевых условиях.

Следовательно, тестирование модели необходимо для исключения, по крайней мере, самой ML из источника сбоев. В этом примере мы узнаем, как выполнять классификацию в реальном времени с помощью Edge Impulse на Raspberry Pi Pico.

Подготовка

Наилучший способ оценить поведение модели ML – протестировать эффективность модели на целевой платформе.

В нашем случае у нас уже есть преимущество, потому что набор данных был создан с помощью Raspberry Pi Pico. Следовательно, достоверность по тестовому набору данных уже должна давать нам четкое представление о том, как ведет себя модель. Однако бывают случаи, когда набор данных не может быть создан из данных датчика, поступающих с целевого устройства. Когда это происходит, модель, развернутая на микроконтроллере, может вести себя не так, как мы ожидаем. Обычно причина такого снижения эффективности кроется в технических характеристиках датчика. По сути, датчики могут быть одного типа, но иметь разные характеристики, такие как смещение, точность, дальность действия, чувствительность и т. д.

Благодаря инструменту Edge Impulse data forwarder легко проверить работу модели на нашей целевой платформе.

Как это делается...

Убедитесь, что на вашем Raspberry Pi Pico по-прежнему запущена программа, разработанная нами в примере «*Получение данных акселерометра*», и что на вашем компьютере запущена программа `edge-impulse-data-forwarder`. Затем перейдите на вкладку **Live classification** (*Классификация в реальном времени*) и проверьте, отображается ли устройство (например, *pico*) в раскрывающемся списке **Device**, как показано на следующем скриншоте:

Рис. 6.23 ❖ Выпадающее меню устройства в Edge Impulse

Если устройства нет в списке, следуйте инструкциям, приведенным в разделе «*Как это делается…*» примера «*Получение данных акселерометра*», чтобы заново подключить ваш Raspberry Pi Pico к Edge Impulse.

Теперь выполните следующие действия, чтобы оценить эффективность модели с помощью инструмента распознавания в реальном времени.

1. В окне **Live classification** выберите 3-осевой датчик из выпадающего списка **Sensor** и установите длительность выборки (мс) равной 20 000. Сохраняйте значение частоты по умолчанию (50 Гц).
2. Поставив перед собой Raspberry Pi Pico, нажмите **Start sampling** (*Начать выборку*) и дождитесь сообщения «*Sampling…*», которое появится на кнопке.
   Когда начнется запись, выполните любое из трех движений, которые может распознать модель (*circle, cross* или *pan*). Семпл будет загружен в Edge Impulse по окончании записи.

Затем Edge Impulse разделит запись на выборки продолжительностью 2.5 с и протестирует обученную модель на каждом. Результаты распознавания будут представлены на той же странице аналогично тому, что мы видели в главе 4 «*Голосовое управление светодиодами с помощью Edge Impulse*».

Распознавание жестов на Raspberry Pi Pico в ОС Arm Mbed

Теперь, когда модель готова, мы можем развернуть ее на Raspberry Pi Pico.

В этом примере мы создадим приложение для непрерывного распознавания жестов с помощью Edge Impulse, Arm Mbed OS и алгоритма для фильтрации избыточных или ложных результатов классификации.

Скетч Arduino *06_gesture_recognition.ino*, содержащий код, который будет разбираться в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter06/ArduinoSketches/06_gesture_recognition.ino.

Подготовка

В этом рецепте мы сделаем наш Raspberry Pi Pico способным распознавать жесты с помощью библиотеки, созданной Edge Impulse для Arduino IDE. В главе 4 «*Голосовое управление светодиодами с помощью Edge Impulse*» мы использовали для этого готовый пример.

Однако здесь мы будем реализовывать всю программу с нуля.

Наша цель – разработать приложение для непрерывного распознавания жестов, что означает, что выборка данных акселерометра и расчеты модели должны выполняться одновременно. Такой подход гарантирует, что мы не пропускаем ни одного события, захватывая и обрабатывая все фрагменты входного потока данных.

Основными компонентами, которые понадобятся для выполнения этой задачи, являются следующие:

- Arm Mbed OS для написания многопоточной программы;
- алгоритм для фильтрации избыточных результатов классификации.

Давайте начнем с изучения того, как легко выполнять параллельные задачи с помощью API операционной системы реального времени (RTOS) в Arm Mbed OS.

Создание рабочих потоков с помощью RTOS API в Arm Mbed OS

Все скетчи Arduino, разработанные для платы Arduino Nano 33 BLE Sense и Raspberry Pi Pico, построены поверх Arm Mbed OS, представляющей собой RTOS с открытым исходным кодом для микроконтроллеров Arm Cortex-M. До сих пор мы использовали только Mbed API для взаимодействия с периферийными устройствами, такими как GPIO и I2C. Однако Arm Mbed OS также предлагает функциональные возможности, типичные для канонической ОС, такие как управление потоками (**managing threads**) для одновременного выполнения различных задач.

Как только поток создан, нам просто нужно привязать поток к функции, которую мы хотим запустить, и выполнить ее, когда мы будем готовы.

Если вы заинтересованы в том, чтобы узнать больше о функциональных возможностях Arm Mbed ОС, мы рекомендуем ознакомиться с официальной документацией, которую можно найти по следующей ссылке:

https://os.mbed.com/docs/mbed-os/v6.15/bare-metal/index.html.

Поток (*thread*) в микроконтроллере – это часть программы, которая выполняется независимо на одном ядре. Поскольку все потоки выполняются на одном ядре, планировщик отвечает за принятие решения о том, что именно выполняется и как долго. Mbed OS использует упреждающий планировщик и циклический алгоритм планирования на основе приоритетов (https://en.wikipedia.org/wiki/Round-robin_scheduling). Таким образом, каждому потоку присваивается приоритет, который предоставляется нами при создании объекта `thread` через RTOS API Mbed ОС (https://os.mbed.com/docs/mbed-os/v6.15/apis/thread.html). Сведения о поддерживаемых значениях приоритета можно найти по адресу https://os.mbed.com/docs/mbed-os/v6.15/apis/thread.html.

Для этого примера нам понадобятся два потока:

- **поток выборки данных** (**Sampling thread**): поток, который отвечает за получение ускорений от MPU-6050 IMU с частотой 50 Гц,
- **поток выполнения** (**Inference thread**): поток, который отвечает за запуск расчетов модели с выводом через каждые 200 мс.

Однако, как мы упоминали в начале этого раздела, многопоточная программа не единственный компонент, необходимый для создания нашего приложения для распознавания жестов. Для отбрасывания избыточных и ложных прогнозов будет также необходим алгоритм фильтрации.

Фильтрация избыточных и ложных прогнозов

Наше приложение для распознавания жестов использует подход на основе скользящего окна по непрерывному потоку данных, чтобы определить, есть ли у нас движение, представляющее интерес. Идея, лежащая в основе этого подхода, состоит в том, чтобы разделить поток данных на меньшие окна фиксированного размера и просчитать модель для каждого из них. Как мы уже знаем, ML – это мощный инструмент для сбора надежных результатов классификации, особенно если мы используем временные сдвиги во входных данных. Следовательно, соседние окна с высокой вероятностью будут иметь схожие оценки, что приведет к множественным избыточным обнаружениям.

В этом примере мы применим алгоритм проверочной следящей фильтрации, чтобы сделать приложение устойчивым к ложным обнаружениям. Концептуально этот алгоритм фильтрации рассматривает результат работы ML как действительный только в том случае, если последние N предсказаний (например, последние четыре) содержат следующее:

- тот же результат, но он отличается от неизвестного (*unknown*) класса;
- оценка вероятности превышает фиксированный порог (например, больше 0.7).

Чтобы наглядно понять, как работает этот алгоритм, посмотрите на следующую схему:

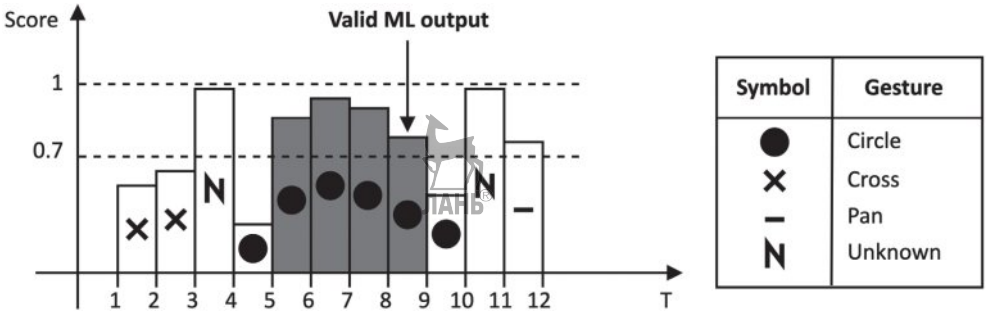


Рис. 6.24 ❖ Пример действительного прогноза ML

На этой схеме каждый прямоугольный столбик является прогнозируемым классом в данный момент времени, где происходит следующее:

- символ представляет прогнозируемый класс вывода;
- высота столбика – это оценка вероятности, связанная с прогнозируемым классом.

Следовательно, учитывая, что N равно четырем, а порог вероятности равен 0.7, мы можем считать результат работы модели ML допустимым только

в момент T = 8. Четыре результата классификации подряд возвращают круг (*circle*) при оценках вероятности, превышающих 0.7.

Как это делается...

Нажмите на **Deployment** (*Развертывание*) в меню с левой стороны и выберите **Arduino Library** из опции **Create library**, как показано на следующем рисунке:

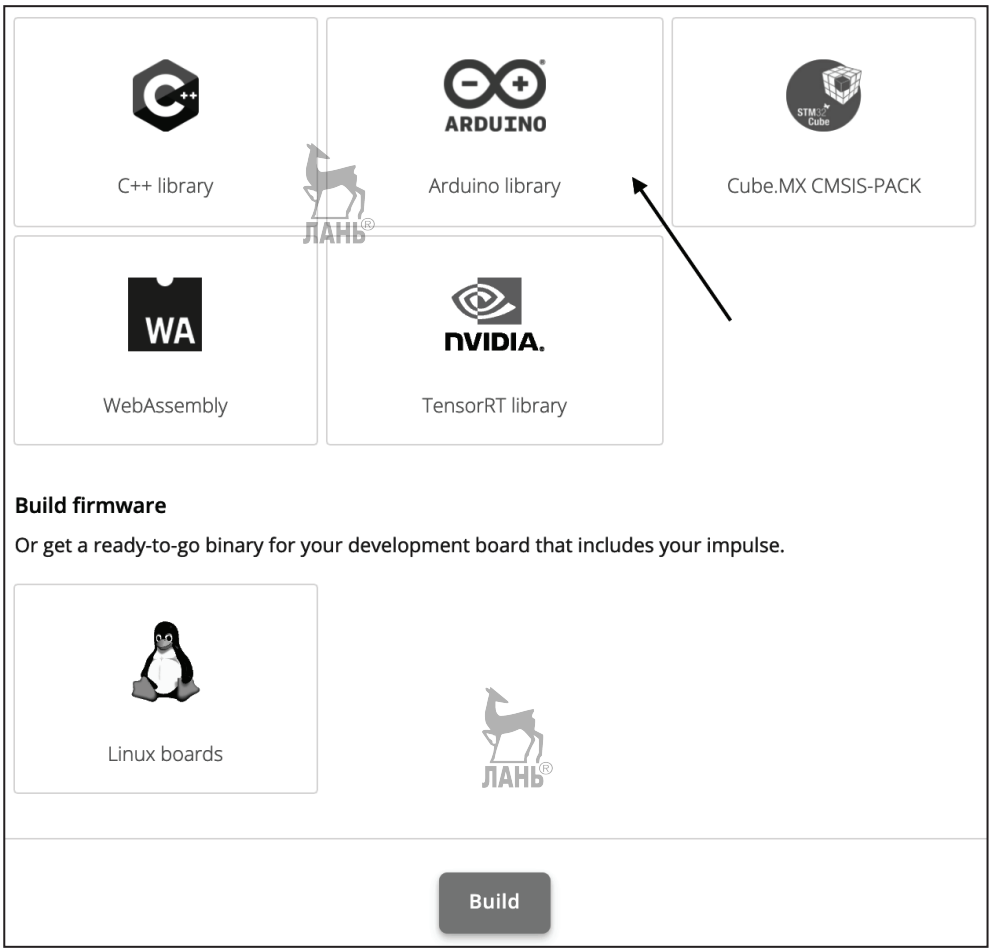


Рис. 6.25 ❖ Секция развертывания Edge Impulse

Затем нажмите на кнопку **Build** в нижней части страницы. Сохраните ZIP-файл на своем компьютере.

Затем импортируйте библиотеку в Arduino IDE. После этого скопируйте скетч, который мы разработали в примере «*Получение данных акселеромет-*

ра», в новый скетч. Чтобы сделать Raspberry Pi Pico способным распознавать наши три жеста, выполните следующие действия.

1. Включите заголовочный файл <*edge_impulse_project_name*>_*inferencing.h* в эскиз. Например, если название проекта Edge Impulse – распознавание жестов, вы должны включить следующую информацию:

   ```
   #include <gesture_recognition_inferencing.h>
   ```

   Этот заголовочный файл является единственным требованием для использования констант, функций и C-макросов, созданых Edge Impulse специально для нашего проекта.
2. Объявите два массива с плавающей запятой (`buf_sampling` и `buf_inference`), каждый из которых содержит по 375 элементов:

   ```
   #define INPUT_SIZE EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE
   float buf_sampling[INPUT_SIZE] = { 0 };
   float buf_inference[INPUT_SIZE];
   ```

   В коде мы использовали макроопределение Edge Impulse `EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE` для получения количества входных выборок, необходимых для 2.5 с данных акселерометра (375).
   Массив `buf_sampling` будет использоваться потоком выборки для хранения данных акселерометра, в то время как массив `buf_inference` будет использоваться потоком выполнения для подачи входных данных в модель.
3. Объявите поток RTOS с низким приоритетом для запуска модели ML:

   ```
   rtos::Thread inference_thread(osPriorityLow);
   ```

   Поток выполнения должен иметь более низкий приоритет (`osPriorityLow`), чем поток выборки, поскольку он имеет более длительное время выполнения из-за проведения расчетов модели ML. Таким образом, расписание с низким приоритетом для потока вычислений гарантирует, что мы не пропустим ни одной выборки данных акселерометра.
4. Создайте класс C++ для реализации алгоритма фильтрации тестов и трассировок. Задайте параметры фильтрации (N и порог вероятности) и переменные, необходимые для отслеживания прогнозов ML (счетчик и последний выходной допустимый индекс класса), типа `private`:

   ```
   class TestAndTraceFilter {
   private:
     int32_t _n {0};
     float _thr {0.0f};
     int32_t _counter {0};
     int32_t _last_idx_class {-1};
     const int32_t _num_classes {3};
   ```

   Алгоритму в основном нужны следующие две переменные для отслеживания результатов классификации:

- `_counter`: используется для отслеживания того, сколько раз у нас была одна и та же классификация с оценкой вероятности выше фиксированного порога (`_thr`);
- `_last_idx_class`: используется для поиска индекса выходного класса последнего результата расчета.

Переменной `_last_idx_class` мы присвоили –1, когда последний результат возвращает либо неизвестное значение, либо имеет оценку вероятности ниже фиксированного порога (`_thr`).

5. Объявите недопустимый результат выходного индекса (–1) константой типа `public`:

```
public:
  static constexpr int32_t invalid_idx_class = -1;
```

6. Создайте конструктор `TestAndTraceFilter` для инициализации параметров фильтрации:

```
public:
  TestAndTraceFilter(int32_t n, float thr) {
    _thr = thr;
    _n = n;
  }
```

7. В классе `TestAndTraceFilter` реализуйте частный метод для сброса внутренних переменных (`_counter` и `_last_idx_class`), которые будут использоваться для отслеживания прогноза ML:

```
  void reset() {
    _counter = 0;
    _last_idx_class = invalid_idx_class;
  }
```

8. В классе `TestAndTraceFilter` реализуйте `public` метод для обновления алгоритма фильтрации с помощью последнего результата классификации:

```
  void update(size_t idx_class, float prob) {
    if(idx_class >= _num_classes || prob < _thr) {
      reset();
    }
    else {
      if(prob > _thr) {
        if(idx_class != _last_idx_class) {
          _last_idx_class = idx_class;
          _counter = 0;
        }
        _counter += 1;
      }
      else {
        reset();
      }
    }
  }
```

Объект `TestAndTraceFilter` работает в двух состояниях – инкремента и сброса, – как показано на следующей блок-схеме:

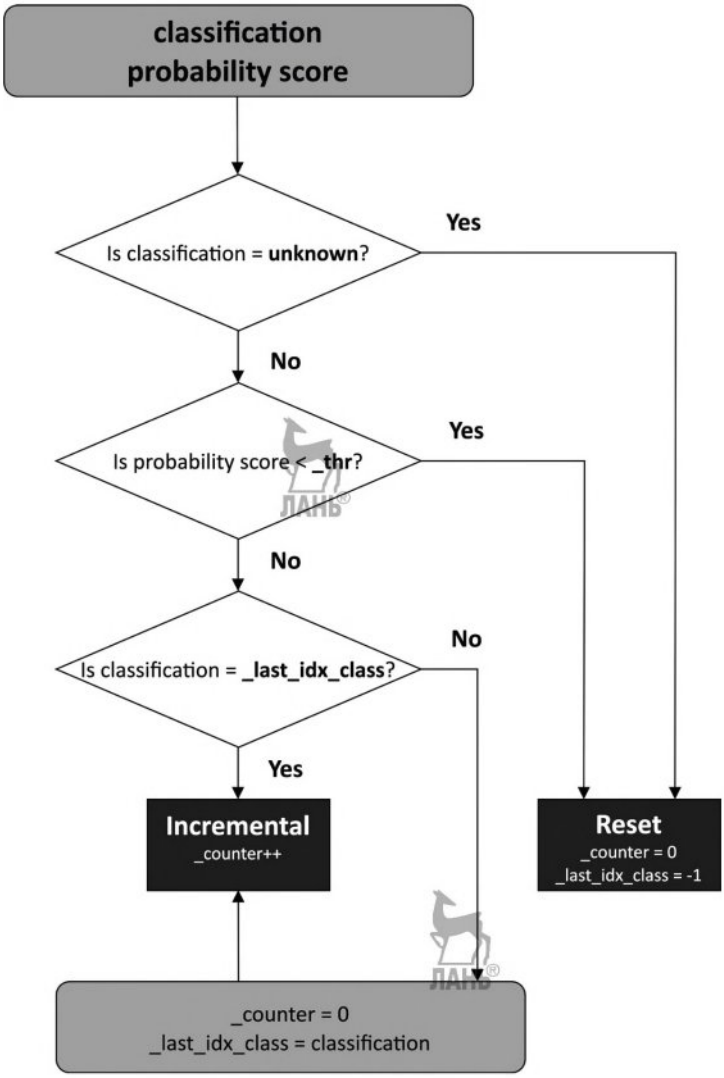


Рис. 6.26 ❖ Блок-схема `TestAndTraceFilter`

Как вы можете видеть, инкрементное состояние возникает, когда самая последняя классификация является допустимым выходным классом и вероятность превышает минимальное значение вероятности. Во всех остальных случаях мы переходим в состояние сброса, где устанавливаем `_counter` равным 0 и `_last_idx_class` равным –1. В инкрементном состоянии _counter увеличивается на единицу, а `_last_idx_class` сохраняет индекс предсказанного выходного класса.

9. В классе TestAndTraceFilter реализуйте public метод для вывода выходных данных фильтра:

```
int32_t output() {
  if(_counter >= _n) {
    int32_t out = _last_idx_class;
    reset();
    return out;
  }
  else {
    return invalid_idx_class;
  }
}
```

Как вы можете видеть, если _counter больше или равен _n, мы возвращаем _last_idx_class и переводим функцию TestAndTraceFilter в состояние сброса.

Если _counter меньше _n, мы возвращаем invalid_idx_class.

10. Напишите функцию для расчета ML (inference_func) в бесконечном цикле (while(1)). Эта функция будет выполнена потоком RTOS inference_thread. Прежде чем вы начнете этот расчет, дождитесь заполнения буфера выборки:

```
void inference_func() {
  delay((EI_CLASSIFIER_INTERVAL_MS * EI_CLASSIFIER_RAW_SAMPLE_COUNT) + 100);
```

Затем инициализируйте объект фильтра. Установите N и порог вероятности равными 4 и 0.7f соответственно:

```
  TestAndTraceFilter filter(4, 0.7f);
```

После инициализации запустите расчет ML в бесконечном цикле:

```
  while (1) {
    memcpy(buf_inference, buf_sampling, INPUT_SIZE * sizeof(float));
    signal_t signal;
    numpy::signal_from_buffer(buf_inference, INPUT_SIZE, &signal);
    ei_impulse_result_t result = { 0 };
    run_classifier(&signal, &result, false);
```

Прежде чем мы запустим расчет, нам нужно скопировать данные из buf_sampling в buf_inference и инициализировать объект Edge Impulse signal_t с буфером buf_inference.

11. Получите выходной класс с наибольшей вероятностью и обновите объект TestAndTraceFilter последним результатом классификации:

```
    size_t ix_max = 0; float pb_max = 0;
#define NUM_OUTPUT_CLASSES EI_CLASSIFIER_LABEL_COUNT
    for (size_t ix = 0; ix < NUM_OUTPUT_CLASSES; ix++) {
      if(result.classification[ix].value > pb_max) {
        ix_max = ix;
        pb_max = result.classification[ix].value;
```

```
    }
  }
  filter.update(ix_max, pb_max);
```

12. Прочитайте выходные данные объекта TestAndTraceFilter. Если результат не равен –1 (неизвестный результат), отправьте название предсказанного жеста по последовательному каналу:

```
int32_t out = filter.output();
if(out != filter.invalid_idx_class) {
  Serial.println(result.classification[out].label);
}
```

Затем подождите 200 мс (величина **window increase**, установленная в проекте Edge Impulse), прежде чем запускать следующий расчет:

```
delay(200);
```

Обратите внимание, что `delay()` переводит текущий поток в состояние ожидания. Как правило, мы всегда должны переводить поток в состояние ожидания, когда он не выполняет вычисления в течение длительного времени. Такой подход гарантирует, что мы не тратим впустую вычислительные ресурсы и что в это время могут выполняться другие потоки.

13. Запустите поток выполнения RTOS (**inference_thread**) в функции `setup()`:

```
inference_thread.start(mbed::callback(&inference_func));
```

14. В функции `loop()` замените вывод через последовательный порт кодом, который требуется для сохранения измерений акселерометра в `buf_sampling`:

```
float ax, ay, az;
read_accelerometer(&ax, &ay, &az);
numpy::roll(buf_sampling, INPUT_ SIZE, -3);
buf_sampling[INPUT_SIZE - 3] = ax;
buf_sampling[INPUT_SIZE - 2] = ay;
buf_sampling[INPUT_SIZE - 1] = az;
```

Поскольку функция Arduino `loop()` является потоком RTOS с высоким приоритетом, нам не нужно создавать дополнительный поток для выборки измерений акселерометра. Следовательно, мы можем заменить функции `Serial.print` кодом, который требуется для заполнения буфера `buf_sampling` данными акселерометра.

Буфер `buf_sampling` заполняется следующим образом:

- сначала мы сдвигаем данные в массиве `buf_sampling` на три позиции, используя функцию `numpy::roll()`. Эта функция предоставляется библиотекой Edge Impulse и работает так же, как ее аналог из NumPy (https://numpy.org/doc/stable/reference/generated/numpy.roll.html);
- затем мы сохраняем измерения 3-осевого акселерометра (ax, ay и az) в последних трех позициях `buf_sampling`.

Такой подход гарантирует, что последние измерения акселерометра всегда будут в последних трех положениях `buf_sampling`. В результате поток вывода может скопировать содержимое этого буфера в буфер `buf_inference` и передать модели ML напрямую, без необходимости выполнять перетасовку данных.

Скомпилируйте и загрузите скетч на Raspberry Pi Pico. Теперь, если вы сделаете любое из трех движений, которые может распознать модель ML (круг, крест или покачивание), вы увидите распознанные жесты в последовательном терминале Arduino.

Создание бесконтактного интерфейса с помощью PyAutoGUI

Теперь, когда мы умеем распознавать жесты рук с помощью Raspberry Pi Pico, можно создать бесконтактный интерфейс для воспроизведения видео на YouTube.

В этом примере мы реализуем скрипт на Python для считывания распознанного движения, передаваемого по последовательному каналу, и используем библиотеку PyAutoGUI для создания интерфейса на основе жестов для воспроизведения, паузы, отключения/включения звука и переключения видео YouTube.

Скрипт на Python *07_gesture_based_ui.py*, содержащий код, разбираемый в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter06/PythonScripts/07_gesture_based_ui.py.

Подготовка

Скрипт на Python, который мы разработаем в этом примере, не будет реализован в Google Colaboratory, так как для этого требуется доступ к локальному последовательному порту, клавиатуре и монитору.

Поэтому мы напишем программу в локальной среде разработки на Python.

Нам нужны только две библиотеки для создания нашего интерфейса на основе жестов: *pySerial* и *PyAutoGUI*. *PySerial* будет использоваться для захвата распознанного жеста, который будет передаваться по последовательному каналу аналогично тому, что мы видели в главе 5 «*Распознавание интерьеров помещений с помощью TensorFlow Lite for Microcontrollers и Arduino Nano*».

Распознанный жест, в свою очередь, выполнит одно из следующих трех действий по воспроизведению видео на YouTube:

Таблица 6.3. Таблица назначения жестов

Жест	**Circle (круг)**	**Cross (крест)**	**Pan (покачивание)**
Функция	Выключение/включение звука	Воспроизведение/пауза	Переход к следующему видео

Поскольку YouTube предлагает сочетания клавиш для перечисленных действий (https://support.google.com/youtube/answer/7631406), мы будем использовать PyAutoGUI для имитации нажатия клавиш клавиатуры, как показано в следующей таблице:

Таблица 6.4. Сочетания клавиш для действий при воспроизведении на YouTube

Жест	**Действие**	**Горячая клавиша**
Circle	Выключение/включение звука	**m**
Cross	Воспроизведение/пауза	**k**
Pan	Переход к следующему видео	**Shift**+**N**

Например, если микроконтроллер через последовательной порт вернет жест `circle`, нам нужно будет имитировать нажатие клавиши *m*.

Как это делается...

Убедитесь, что вы установили PyAutoGUI в вашей локальной среде разработки Python (например, `pip install pyautogui`). После этого создайте новый скрипт на Python и импортируйте следующие библиотеки:

```
import serial
import pyautogui
```

Чтобы создать бесконтактный интерфейс с помощью PyAutoGUI, выполните следующие действия.

1. Инициализируйте `pySerial` с указанием порта и скорости передачи данных, используемых Raspberry PI Pico:

```
port = '/dev/ttyACM0'
baudrate = 115600
ser = serial.Serial()
ser.port = port
ser.baudrate = baudrate
```


После инициализации откройте последовательный порт и очистите содержимое его входного буфера:

```
ser.open()
ser.reset_input_buffer()
```

2. Создайте служебную функцию для посылки строки из последовательного порта в нужном формате:

```
def serial_readline():
  data = ser.readline
  return data.decode("utf-8").strip()
```

3. Используйте цикл `while` для чтения присылаемых данных построчно:

```
while True:
  data_str = serial_readline()
```

Для каждой строки проверьте, есть ли у нас жесты *circle, cross* или *pan*. Если у нас обнаружено круговое движение, нажмите клавишу *m*, чтобы отключить или включить звук:

```
if str(data_str) == "circle":
  pyautogui.press('m')
```

Если у нас обнаружено движение крестом, нажмите клавишу *k* для воспроизведения/паузы:

```
if str(data_str) == "cross":
  pyautogui.press('k')
```

Если у нас обнаружено покачивание, нажмите горячую клавишу **Shift**+**N**, чтобы перейти к следующему видео:

```
if str(data_str) == "pan":
  pyautogui.hotkey('shift', 'n')
```

4. Запустите скрипт на Python, убедившись, что на вашем Raspberry Pi Pico запущен скетч, который мы разработали в предыдущем примере.

Затем откройте YouTube в своем веб-браузере, воспроизведите видео и положите Raspberry Pi Pico перед собой. Теперь, если вы сделаете любое из трех оговоренных движений, вы сможете управлять воспроизведением видео на YouTube с помощью жестов!

Глава 7

Запуск модели TinyML CIFAR-10 на виртуальной платформе ОС Zephyr

Создание прототипа приложения TinyML непосредственно на физическом устройстве действительно увлекательно, потому что мы можем мгновенно увидеть наши идеи в действии на чем-то, что выглядит и ощущается как настоящая вещь. Однако, прежде чем какое-либо приложение заработает, мы должны убедиться, что модели работают должным образом и, возможно, на разных устройствах. Тестирование и отладка приложений непосредственно на платах микроконтроллеров часто требует много времени. Основной причиной этого является необходимость загружать программу в устройство при каждом изменении кода. Виртуальные платформы могут пригодиться для того, чтобы сделать тестирование более простым и быстрым.

В этой главе мы создадим приложение для распознавания изображений с помощью **TensorFlow Lite for microcontrollers** (**TFLu**) для эмулируемого микроконтроллера Arm Cortex-M3. Мы начнем с установки **Zephyr OS**, основной платформы, используемой в этой главе для выполнения нашей задачи. Далее мы разработаем малую квантизованную модель **CIFAR-10** с помощью **TensorFlow** (TF). Эта модель будет способна работать на микроконтроллере с объемом всего 256 Кбайт памяти программ и 64 Кбайт оперативной памяти. В конце концов мы развернем приложение распознавания изображений на эмулируемом микроконтроллере Arm Cortex-M3 с помощью **Quick Emulator** (**QEMU**).

Цель этой главы – научиться создавать и запускать приложение на основе TFLu с ОС Zephyr на виртуальной платформе и предоставить практические рекомендации по разработке модели распознавания изображений для микроконтроллеров с ограниченным объемом памяти.

В этой главе мы собираемся реализовать следующие примеры.

- Начало работы с ОС Zephyr.
- Разработка и обучение малой модели CIFAR-10.
- Оценка точности модели TFLite.
- Преобразование цифрового изображения в C-байтовый массив.
- Подготовка основы проекта TFLu.
- Создание и запуск приложения TFLu на QEMU.

Технические требования

Чтобы практически выполнить все примеры этой главы, нам понадобится следующее:

- ноутбук/ПК с Ubuntu 18.04+ или более поздней версией на x86_64.

Исходный код и дополнительные материалы доступны в папке *Chapter07* по адресу https://github.com/PacktPublishing/TinyML-Cookbook/tree/main/Chapter07.

Начало работы с ОС Zephyr

В этом примере мы установим проект Zephyr – фреймворк, используемый в этой главе для сборки и запуска приложения TFLu на эмулируемом микроконтроллере Arm Cortex-M3. В конце этого примера мы проверим, все ли работает так, как ожидалось, запустив образец приложения на виртуальной платформе, рассматриваемой в нашем проекте.

Подготовка

Чтобы приступить к работе над этим первым примером, нам нужно знать, о чем идет речь в проекте Zephyr.

Zephyr (https://zephyrproject.org/) – это проект Apache 2.0 с открытым исходным кодом, который предоставляет компактную операционную систему реального времени (RTOS) для различных аппаратных платформ, основанных на нескольких архитектурах, включая Arm Cortex-M, Intel x86, ARC, Nios II и RISC-V. RTOS была разработана для устройств с ограниченным объемом памяти с учетом соображений безопасности.

Однако Zephyr предоставляет не только RTOS. Он также предлагает **Software Development Kit** (SDK) с коллекцией готовых к использованию примеров и инструментов для создания приложений на базе Zephyr для различных поддерживаемых устройств, включая виртуальные платформы через Quick Emulator (QEMU).

QEMU (https://www.qemu.org) – это эмулятор аппаратного обеспечения с открытым исходным кодом, который позволяет нам тестировать программы без использования реального оборудования. Zephyr SDK поддерживает два QEMU для микроконтроллеров на базе Arm Cortex-M:

- **micro:bit** от BBC (https://microbit.org/) на основе Arm Cortex-M0,
- Texas Instruments **LM3S6965** (https://www.ti.com/product/LM3S6965) на основе Arm Cortex-M3.

Из двух представленных платформ QEMU мы будем использовать LM3S6965. Наш выбор пал на плату Texas Instruments, потому что у нее больший объем оперативной памяти, чем у BBC micro:bit. Хотя устройства имеют одинаковый объем программной памяти (256 Кбайт), LM3S6965 имеет 64 Кбайт оперативной памяти. К сожалению, BBC micro:bit имеет всего 16 Кбайт оперативной памяти, что недостаточно для запуска модели CIFAR-10.

Как это делается...

Установка Zephyr состоит из следующих этапов.

1. Установка предварительных условий Zephyr.
2. Получение исходного кода Zephyr и связанных с ним зависимостей Python.
3. Установка Zephyr SDK.

Руководство по установке, представленное в этом разделе, относится к Zephyr 2.7.0 и Zephyr SDK 0.13.1.

Прежде чем приступить к работе, мы рекомендуем вам установить инструмент Python **Virtual Environment** (**virtualenv**) для создания изолированной среды Python. Если вы еще не установили его, откройте свой терминал и используйте следующую команду `pip`:

```
$ pip install virtualenv
```

Чтобы запустить виртуальную среду Python, создайте новый каталог (например, *zephyr*):

```
$ mkdir zephyr && cd zephyr
```

Затем создайте виртуальную среду внутри только что созданного каталога:

```
$ python -m venv env
```

Предыдущая команда создает каталог *env* со всеми исполняемыми файлами и пакетами Python, необходимыми для виртуальной среды.

Чтобы использовать виртуальную среду, вам просто нужно активировать ее с помощью следующей команды:


```
$ source env/bin/activate
```

Если виртуальная среда активирована, оболочка будет иметь префикс `(env)`:

```
(env)$
```

Вы можете деактивировать виртуальную среду Python в любое время, набрав `deactivate` в командной оболочке.

Следующие шаги помогут вам подготовить среду Zephyr и запустить простое приложение на виртуальном микроконтроллере на базе Arm Cortex-M3.

1. Следуйте инструкциям, приведенным в руководстве по началу работы с Zephyr (https://docs.zephyrproject.org/2.7.0/getting_started/index.html) до тех пор, пока не появится раздел *Install a Toolchain*. Все модули Zephyr будут доступны в каталоге *~/zephyrproject*.
2. Перейдите в каталог исходного кода Zephyr и введите `samples/synchronization`:

   ```
   $ cd ~/zephyrproject/zephyr/samples/synchronization
   ```

Zephyr в папке *samples/* предоставляет готовые к использованию приложения для демонстрации использования функций RTOS. Поскольку наша цель – запустить приложение на виртуальной платформе, рассмотрим пример *synchronization*, так как он не требует взаимодействия с внешними компонентами (например, светодиодами).

3. Создайте подготовленный образец синхронизации для `qemu_cortex_m3`:

```
$ west build -b qemu_cortex_m3
```

Образец теста составлен с помощью команды `west` (https://docs.zephyrproject.org/latest/guides/west/index.html). *West* – это инструмент, разработанный компанией Zephyr для удобного управления несколькими репозиториями с помощью нескольких командных строк. Однако *West* – это нечто большее, чем менеджер хранилища. Он также может с помощью расширений подключать дополнительные функции. Zephyr использует этот механизм для предоставления команд компиляции, прошивки и отладки приложений (https://docs.zephyrproject.org/latest/guides/west/build-flash-debug.html).

Команда `west`, используемая для компиляции приложения, имеет следующий синтаксис:

```
$ west build -b <BOARD> <EXAMPLE-TO-BUILD>
```

Разберем приведенную команду:

- `<BOARD>`: название целевой платформы. В нашем случае это QEMU Arm Cortex-M3 (`qemu_cortex_m3`),
- `<EXAMPLE-TO-BUILD>`: это путь к образцу теста для компиляции.

Как только мы создадим приложение, мы сможем запустить его на целевом устройстве.

4. Запустите пример синхронизации на виртуальной платформе LM3S6965:

```
$ west build -t run
```

Чтобы запустить приложение, нам нужно использовать команду `west build`, за которой следует целевая (`target`) платформа в качестве аргумента командной строки (`-t`). Поскольку мы указали целевую платформу при создании приложения, мы можем просто передать опцию запуска, чтобы загрузить и запустить программу на устройстве.

Если Zephyr установлен правильно, пример синхронизации будет запущен на виртуальной платформе Arm Cortex-M3 и выдаст следующее:

```
threadA: Hello World from arm!
threadB: Hello World from arm!
threadA: Hello World from arm!
threadB: Hello World from arm!
```

Теперь вы можете закрыть QEMU, нажав **Ctrl**+**A**.

Разработка и обучение малой модели CIFAR-10

Жесткие ограничения памяти на LM3S6965 вынуждают нас разрабатывать модель с чрезвычайно ограниченным использованием памяти. Фактически целевой микроконтроллер имеет в четыре раза меньший объем памяти, чем Arduino Nano.

Из-за такого серьезного ограничения в этом примере мы будем использовать уменьшенную модель для классификации изображений CIFAR-10, способную работать на LM3S6965:

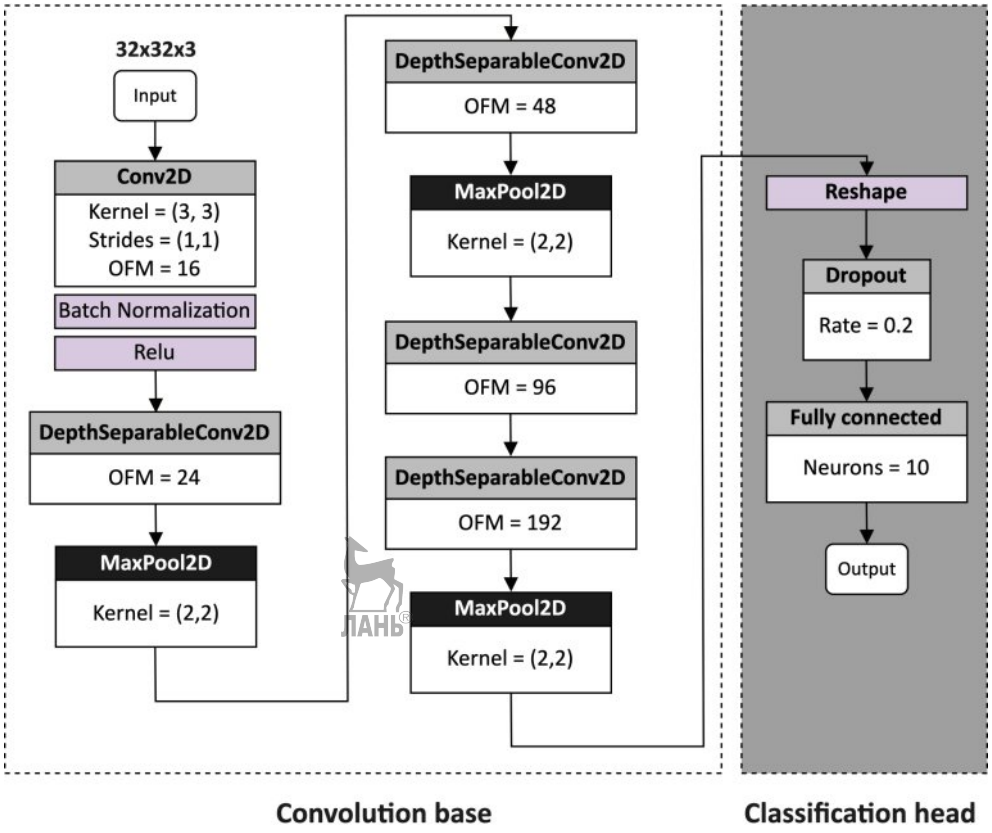


Рис. 7.1 ❖ Модель, адаптированная для классификации изображений набора данных CIFAR-10

Представленная сеть будет спроектирована с использованием TF и Keras API.

Файл Colab *prepare_model.ipynb*, содержащий (в разделе «*Designing and training a tiny CIFAR-10 model*») код, разбираемый в этом примере, доступен

по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter07/ColabNotebooks/prepare_model.ipynb.

Подготовка

Сеть, созданная в этом примере, основана на успешной архитектуре сверточной сети MobileNet V1 для классификации набора данных **ImageNet**. Наша модель направлена на классификацию 10 классов набора данных CIFAR-10, это: самолет (*airplane*), автомобиль (*automobile*), птица (*bird*), кошка (*cat*), олень (*deer*), собака (*dog*), лягушка (*frog*), лошадь (*horse*), корабль (*ship*) и грузовик (*truck*).

Набор данных CIFAR-10 доступен по адресу https://www.cs.toronto.edu/~kriz/cifar.html и состоит из 60 000 изображений в формате RGB с разрешением 32×32.

Чтобы понять, почему предлагаемая модель может успешно работать на LM3S6965, опишем архитектурные решения, которые делают эту сеть подходящей для нашего целевого устройства.

Как показано на рис. 7.1, модель имеет сверточную базу, которая действует как средство извлечения признаков, и классификационный блок, использующий извлеченные признаки для выполнения классификации.

Первые слои имеют большие пространственные размеры и выходные карты параметров (Output Feature Maps, OFM) с низкой детализацией выходных признаков для изучения простых объектов (например, простых линий). Более глубокие слои, напротив, имеют небольшие пространственные размеры и высокую детализацию OFM для изучения сложных объектов (например, форм).

Модель использует связанные слои для уменьшения вдвое пространственной размерности тензоров и уменьшения риска переобучения при увеличении OFM. Как правило, мы хотим, чтобы различные карты активации для глубоких слоев сочетали в себе как можно больше сложных функций.

Поэтому идея состоит в том, чтобы получить меньшие пространственные размеры, чтобы позволить себе больше возможностей.

В следующем подразделе мы объясним выбор структуры при использовании разделяемых сверточных слоев (**Depthwise Separable Convolution**, **DWSC**) вместо стандартной 2D-свертки.

Замена свертки 2D на DWSC

DWSC – прием, который обеспечил успех MobileNet V1 в наборе данных ImageNet и является сердцем предлагаемой нами архитектуры на основе свертки. Этот оператор взял на себя ведущую роль в MobileNet V1 для создания достоверной модели, которая также может работать на устройстве с ограниченной памятью и вычислительными ресурсами.

Как показано в главе 5 «*Распознавание интерьеров помещений с помощью TensorFlow Lite for microcontrollers и Arduino Nano*»[79] и представлено на сле-

[79] Смотрите в главе 5 раздел «*Изучение вариантов дизайна сети MobileNet*».

дующем рисунке, DWSC представляет собой свертку по глубине, за которой следует слой свертки с размером ядра 1×1 (т. е. **pointwise convolution** – точечная свертка):

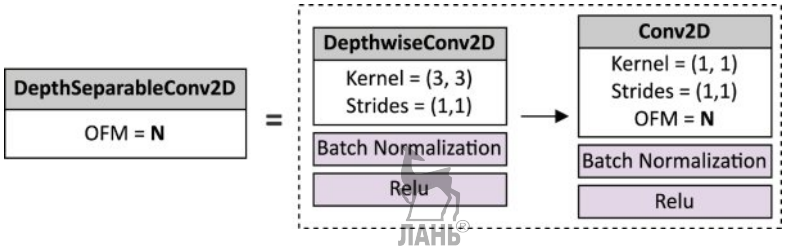


Рис. 7.2 ❖ DWSC

Чтобы продемонстрировать эффективность этого оператора, рассмотрим первый уровень DWSC в сети, представленный на рис. 7.1. Как показано на следующей блок-схеме, входной тензор имеет размерность 32×32×16, в то время как выходной тензор имеет размерность 32×32×24:

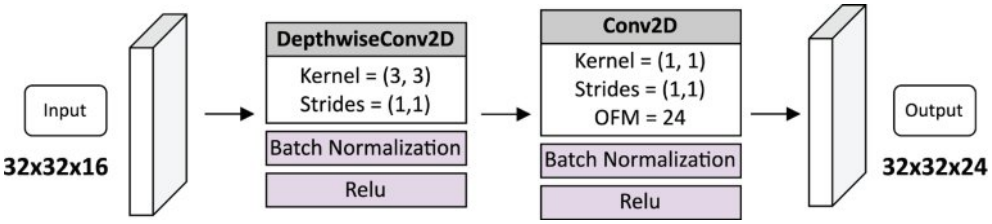


Рис. 7.3 ❖ Первый DWSC в модели CIFAR-10

Если мы заменим DWSC обычной сверткой 2D с размером фильтра 3×3, нам понадобится 3480 обучаемых параметров, из которых 3456 являются весами (3×3×16×24), а 24 – смещениями.

Вместо этого DWSC нуждается только в 560 обучаемых параметрах, распределенных следующим образом:

- 144 веса и 16 смещений для слоя свертки по глубине с размером фильтра 3×3;
- 384 веса и 24 смещения для точечной свертки.

Следовательно, в данном конкретном случае уровень DWSC дает примерно в шесть раз меньше обучаемых параметров, в отличие от обычного 2D-слоя свертки.

Уменьшение размера модели не единственное преимущество, которое предлагает этот слой. Другое преимущество использования DWSC заключается в сокращении арифметических операций. Фактически, хотя оба слоя состоят из нескольких операций умножения-накопления (**Multiply-Accumulate**, **MAC**), DWSC требует значительно меньше операций MAC, чем свертка 2D.

Этот факт демонстрируется следующими двумя формулами расчета общего количества MAC-операций для 2D-свертки и DWSC:

$MACs_{conv2d} = F_{size} \cdot W_{out} \cdot H_{out} \cdot C_{out} \cdot C_{in}$;

$MACs_{dwsc} = (F_{size} \cdot W_{out} \cdot H_{out} \cdot C_{in}) + (W_{out} \cdot H_{out} \cdot C_{out} \cdot C_{in})$.

Формулы расшифровываются следующим образом:

- $MACs_{conv2d}$: общее количество операций MAC для свертки 2D,
- $MACs_{dwsc}$: общее количество операций MAC для DWSC,
- F_{size}: размер фильтра,
- $W_{out} \cdot H_{out}$: ширина и высота выходного тензора,
- $C_{in} \cdot C_{out}$: количество входных и выходных карт параметров.

Расчет общего количества MAC для DWSC состоит из двух частей. Первая часть ($F_{size} \cdot W_{out} \cdot H_{out} \cdot C_{in}$) вычисляет операции MAC для свертки по глубине, предполагая, что входные и выходные тензоры имеют одинаковые карты параметров. Вторая часть ($W_{out} \cdot H_{out} \cdot C_{out} \cdot C_{in}$) вычисляет операции MAC для точечной свертки.

Если использовать эти две формулы для случая, представленного на рис. 7.3, мы обнаружим, что для 2D-свертки требуется 3 583 944 операции, в то время как для DWSC требуется всего 540 672 операций. Таким образом, с помощью DWSC вычислительная сложность снижается более чем в шесть раз.

Следовательно, эффективность уровня DWSC удваивается, поскольку он уменьшает количество и обучаемых параметров, и задействованных арифметических операций.

Теперь, когда мы знаем о преимуществах этого уровня, давайте узнаем, как создать модель, которая может работать на целевом устройстве.

Контроль поддержки требований модели к памяти

Наша цель – создать модель, которая может поместиться в 256 Кбайт программной памяти и работать с 64 Кбайт оперативной памяти. Требования к памяти программ может быть получено непосредственно из сгенерированной модели `.tflite`. В качестве альтернативы можете проверить значение **Total params**, возвращаемое методом Keras `summary()` (https://keras.io/api/models/model/#summary-method), чтобы иметь представление о том, насколько большой будет модель.

Total params представляет собой количество обучаемых параметров, и на него в основном влияют количество OFM и слоев. В нашем случае база свертки имеет пять обучаемых слоев максимум с 192 картами активации, что позволит нашей модели использовать всего 30 % общей памяти программ.

Оценка использования оперативной памяти немного сложнее и зависит от архитектуры модели. Все меняющиеся переменные, такие как сетевые входные, выходные и промежуточные тензоры, остаются в оперативной памяти. Однако, хотя сети может потребоваться несколько тензоров, TFLu имеет диспетчер памяти, способный эффективно делить память на части во время выполнения. Для последовательной модели, такой как наша, где каждый слой имеет один входной и один выходной тензор, приблизительный показатель использования оперативной памяти определяется суммой следующих слагаемых:

- памяти, необходимой для входных и выходных тензоров модели;
- двух самых больших промежуточных тензоров.

В нашей сети первый DWSC производит самый большой промежуточный тензор с 24 576 элементами (32×32×24), как показано на следующем рисунке:

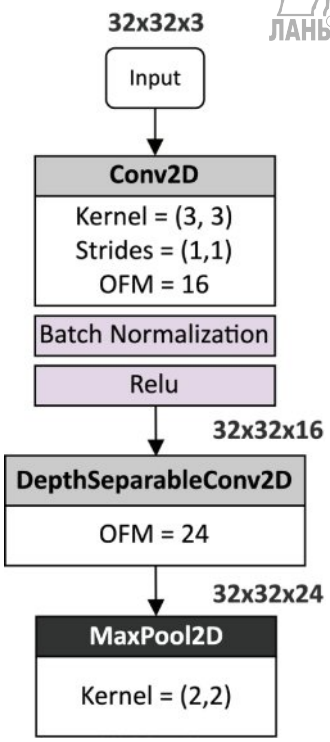


Рис. 7.4 ❖ Первый DWSC выдает самый большой промежуточный тензор

Как можно видеть из блок-схемы, первый DWSC создает тензор с 24 OFM, которые мы нашли как хороший компромисс между точностью и использованием оперативной памяти. Тем не менее вы можете рассмотреть возможность дальнейшего уменьшения этого параметра, чтобы сделать модель еще меньше и более производительной.

Как это делается...

Чтобы спроектировать и обучить квантизованную модель CIFAR-10 с помощью TFLite, создайте новый проект Colab и выполните следующие действия.

1. Загрузите набор данных CIFAR-10:

```
(train_imgs, train_lbls), (test_imgs, test_lbls) =
datasets.cifar10.load_data()
```

2. Нормализуйте значения пикселей в диапазоне от 0 до 1:

```
train_imgs = train_imgs / 255.0
test_imgs = test_imgs / 255.0
```

Этот шаг гарантирует, что все данные находятся в одном масштабе.

3. Создайте функцию Python для реализации DWSC:

```
def separable_conv(i, ch):
    x = layers.DepthwiseConv2D((3,3), padding="same")(i)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
    x = layers.Conv2D(ch, (1,1), padding="same")(x)
    x = layers.BatchNormalization()(x)
    return layers.Activation("relu")(x)
```

Функция `separable_conv()` принимает следующие входные аргументы:

- `i`: входные данные для глубокой свертки 2D;
- `ch`: количество продуцируемых OFMS.

`layers.BatchNormalization()` стандартизирует входные данные для слоя и делает обучение модели более быстрым и стабильным.

4. Создайте сверточную базу (см. рис. 7.1):

```
input = layers.Input((32,32,3))
x = layers.Conv2D(16, (3, 3), padding='same')(input)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = separable_conv(0, x, 24)
x = layers.MaxPooling2D((2, 2))(x)
x = separable_conv(0, x, 48)
x = layers.MaxPooling2D((2, 2))(x)
x = separable_conv(0, x, 96)
x = separable_conv(0, x, 192)
x = layers.MaxPooling2D((2, 2))(x)
```

Мы используем полносвязные слои для уменьшения пространственной размерности карт объектов по сети. Хотя мы можем использовать DWSC с неединичными шагами (*non-unit strides*) для выполнения аналогичной задачи субдискретизации, мы предпочли объединить слои, чтобы сохранить количество обучаемых параметров на низком уровне.

5. Разработайте классификационный блок:

```
x = layers.Flatten()(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(10)(x)
```

6. Сгенерируйте модель и выведите ее краткое описание:

```
model = Model(input, x)
model.summary()
```

Как показано на следующем рисунке, описание модели возвращает примерно 60 000 параметров:

```
=================================
Total params: 60,194
Trainable params: 59,074
Non-trainable params: 1,120
_________________________________
```

Рис. 7.5 ❖ Краткое описание модели CIFAR-10 (параметры обучения)

В случае 8-битной квантизации 60 000 параметров с плавающей запятой соответствуют 60 000 8-битных целых значений. Таким образом, веса вносят вклад в размер модели в 60 Кбайт, что значительно отличается от максимального целевого значения в 256 Кбайт. Однако мы не должны рассматривать это число как размер модели, поскольку то, что мы развертываем на микроконтроллере, – это файл TFLite, который также содержит сетевую архитектуру и параметры квантизации.

Приблизительный показатель использования оперативной памяти можно оценить по размеру каждого промежуточного тензора в сети. Эта информация может быть экстраполирована из выходных данных `model.summary()`. Как и указывалось в предыдущем разделе «*Подготовка*», промежуточные тензоры первого слоя DWSC содержат наибольшее количество элементов. Следующий рисунок демонстрирует выходные данные `model.summary()`, в том числе параметры этих двух тензоров:

act1 (Activation)	(None, 32, 32, 16)	0	
dwc0_dwsc2 (DepthwiseConv2D)	(None, 32, 32, 16)	160	DWSC
bn0_dwsc2 (BatchNormalization)	(None, 32, 32, 16)	64	
act0_dwsc2 (Activation)	(None, 32, 32, 16)	0	
conv0_dwsc2 (Conv2D)	(None, 32, 32, 24)	408	
bn1_dwsc2 (BatchNormalization)	(None, 32, 32, 24)	96	
act1_dwsc2 (Activation)	(None, 32, 32, 24)	0	
pool1 (MaxPooling2D)	(None, 16, 16, 24)	0	

Рис. 7.6 ❖ Краткое описание модели CIFAR-10 (первый DWSC)

Как вы можете видеть из области DWSC, отмеченной на рисунке прямоугольником, тензоры с наибольшим количеством элементов следующие:

- выход **act0_dwsc2: (None, 32, 32, 16)**,
- выход **conv0_dwsc2: (None, 32, 32, 24)**.

Следовательно, ожидаемое использование памяти для промежуточного тензора должно составлять порядка 41 Кбайт. К этому числу мы должны добавить память для входных и выходных узлов, чтобы получить более точную приблизительную оценку использования оперативной памяти.

Для входных и выходных тензоров требуется 3082 байта, из которых 3072 байта предназначены для входа и 10 байт – для выхода. В общей сложности мы ожидаем использовать 44 Кбайт оперативной памяти во время просчета модели, что меньше целевого показателя в 64 Кбайт.

На рис. 7.6 показаны три слоя с выходными параметрами **(None, 32, 32, 16)**: **conv0_dwsc2**, **bn1_dwsc2** и **act1_dwsc2**. Однако только уровень точечной свертки (`conv0_dwsc2`) должен учитывать использование памяти промежуточных тензоров, поскольку слой с пакетной нормализацией (bn1_dwsc2) и активационный слой (act1_dwsc2) будут объединены в свертку (conv0_dwsc2) с помощью конвертера TFLite.

7. Скомпилируйте и обучите модель с 10 периодами обучения:

```
model.compile(optimizer='adam',
loss = tf.keras.losses.SparseCategoricalCrossentropy(
from_logits=True), metrics=['accuracy'])

model.fit(train_imgs, train_lbls, epochs=10,
validation_data=(test_imgs, test_lbls))
```

   Через 10 периодов модель должна получить достоверность 73 % по тестовому набору данных.

8. Сохраните модель TF как `SavedModel`:

```
model.save("cifar10")
```

Модель CIFAR-10 теперь готова к квантизации с помощью преобразователя TFLite.

Оценка достоверности модели TFLite

Только что обученная малая модель может распознать 10 классов CIFAR-10 с достоверностью 73 %. Однако какова достоверность квантизованного варианта, сгенерированного конвертором TFLite?

В этом примере мы будем квантизовать модель с помощью конвертора TFLite и покажем, как выполнить оценку достоверности на тестовом наборе данных с помощью TFLite Python-интерпретатора.

После оценки достоверности мы преобразуем модель TFLite в C-байтовый массив.

Файл Colab *prepare_model.ipynb*, содержащий (в разделе *Evaluating the accuracy of the quantized model*) код, разбираемый в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter07/ColabNotebooks/prepare_model.ipynb.

Подготовка

В этом разделе мы объясним, почему достоверность модели TFLite может отличаться от первоначальной.

Как мы знаем, обученная модель должна быть преобразована в более компактное и легкое представление перед развертыванием на устройстве с ограниченными ресурсами, таком как микроконтроллер.

Квантизация является важной частью этого шага, позволяющего уменьшить размер модели и повысить производительность ее работы. Однако квантизация после обучения может изменить достоверность модели из-за арифметических операций с меньшей точностью. Поэтому крайне важно чтобы проверить, находится ли достоверность сгенерированной модели `.tflite` в пределах допустимого, прежде чем развертывать ее на целевом устройстве.

К сожалению, TFLite не предоставляет инструмент Python для оценки достоверности модели. Мы вынуждены использовать интерпретатор TFLite Python для выполнения этой задачи. Интерпретатор позволит нам передать входные данные в сеть и прочитать результат классификации. Достоверность будет указана как доля правильно распознанных образцов из тестового набора данных.

Как это делается...

Чтобы оценить достоверность квантизованной модели CIFAR-10 по тестовому набору данных, выполните следующие действия.

1. Выберите несколько сотен выборок из набора данных `train` для калибровки квантизации:

```
cifar_ds = tf.data.Dataset.from_tensor_slices(train_images).batch(1)
def representative_data_gen():
    for i_value in cifar_ds.take(100):
         i_value_f32 = tf.dtypes.cast(i_value, tf.float32)
         yield [i_value_f32]
```

   Конвертер TFLite использует набор данных `representative` для оценки параметров квантизации.

2. Инициализируйте конвертер TFLite для выполнения 8-битной квантизации:

```
tflite_conv = tf.lite.TFLiteConverter.from_saved_model("cifar10")
tflite_conv.representative_dataset = tf.lite.
RepresentativeDataset(representative_data_gen)
tflite_conv.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_conv.target_spec.supported_ops = [tf.lite.OpsSet.
TFLITE_BUILTINS_INT8]
tflite_conv.inference_input_type = tf.int8
tflite_conv.inference_output_type = tf.int8
```

Для квантизации модели TF до 8-разрядной мы импортируем каталог из *SavedModel* (`cifar10`) в конвертер TFLite и применяем полную целочисленную квантизацию.

3. Преобразуйте модель в формат файла TFLite и сохраните ее как `.tflite`:

```
tfl_model = tfl_conv.convert()
open("cifar10.tflite", "wb").write(tfl_model)
```

4. Оцените размер модели TFLite:

```
print(len(tfl_model))
```

Ожидаемый размер модели составляет 81 304 байта. Как вы можете видеть, модель поместится в 256 Кбайт программной памяти.

5. Оцените достоверность квантизованной модели, используя тестовый набор данных. Для этого запустите интерпретатор TFLite и выделите тензоры:

```
tfl_inter = tf.lite.Interpreter(model_content=tfl_model)
tfl_inter.allocate_tensors()
```

Получите параметры квантизации входного и выходного узлов:

```
i_details = tfl_inter.get_input_details()[0]
o_details = tfl_inter.get_output_details()[0]
i_quant = i_details["quantization_parameters"]
i_scale = i_quant['scales'][0]
i_zero_point = i_quant['zero_points'][0]
o_scale = o_quant['scales'][0]
o_zero_point = o_quant['zero_points'][0]
```

Инициализируйте переменную `num_correct_samples` равной нулю, чтобы отслеживать верные классификации:

```
num_correct_samples = 0
num_total_samples = len(list(test_imgs))
```

Выполните перебор по тестовым образцам:

```
for i_value, o_value in zip(test_imgs, test_lbls):
    input_data = i_value.reshape((1, 32, 32, 3))
    i_value_f32 = tf.dtypes.cast(input_data, tf.float32)
```

Квантизуем каждый тестовый образец:

```
    i_value_f32 = i_value_f32 / i_scale + i_zero_point
    i_value_s8 = tf.cast(i_value_f32, dtype=tf.int8)
```

Инициализируйте входной узел квантизованной выборкой и запустите распознавание:

```
    tfl_conv.set_tensor(i_details["index"], i_value_s8)
    tfl_conv.invoke()
```

Прочитайте результат и деквантизуйте выходные данные до значения с плавающей запятой:

```
o_pred = tfl_conv.get_tensor(o_details["index"])[0]
o_pred_f32 = (o_pred - o_zero_point) * o_scale
```

Сравните результат классификации с ожидаемым выходным классом:

```
if np.argmax(o_pred_f32) == o_value:
     num_correct_samples += 1
```

6. Выведите достоверность квантизованной модели TFLite:

```
print("Accuracy:", num_correct_samples/num_total_samples)
```

Через несколько минут результирующая достоверность окажется в журнале вывода. Ожидаемое значение по-прежнему должно составлять около 73 %.

7. Преобразуйте модель TFLite в C-байтовый массив с помощью `xxd`:

```
!apt-get update && apt-get -qq install xxd
!xxd -i cifar10.tflite > model.h
```

Вы можете загрузить файлы *model.h* и *cifar10.tflite* с левой панели Colab.

Преобразование цифрового изображения в C-байтовый массив

Наше приложение будет запущено на виртуальной платформе без доступа к модулю камеры. Поэтому нам необходимо предоставить действительное тестовое входное изображение, чтобы проверить, работает ли модель так, как ожидалось.

В этом примере мы получим изображение из тестового набора данных, которое должно возвращать правильную классификацию для класса *ship* (*корабль*). Затем образец будет преобразован в C-массив типа `int8_t` и сохранен в виде файла `input.h`.

Файл Colab *prepare_model.ipynb*, содержащий (в разделе *Converting a NumPy image to a C-byte array*) код, разбираемый в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter07/Colab-Notebooks/prepare_model.ipynb.

Подготовка

Чтобы подготовиться к этому примеру, нужно знать, как подготовить C-файл, содержащий входное тестовое изображение. Структура этого файла довольно проста и представлена на следующем рисунке:

Рис. 7.7 ❖ Структура C-файла заголовка для входного тестового изображения

Как вы можете видеть из структуры файла, нам нужен только массив и две переменные для описания входного тестового образца:

- `g_test`: массив типа `int8_t`, содержащий изображение корабля с нормализованными и квантизованными значениями пикселей. Пиксели, хранящиеся в массиве (`// data`), должны представлять собой целочисленные значения, разделенные запятыми;
- `g_test_len`: целочисленная переменная для размера массива. Поскольку входная модель представляет собой изображение RGB с разрешением 32×32, мы ожидаем массив с 3072 элементами типа `int8_t`;
- `g_test_ilabel`: целочисленная переменная для индекса класса входного тестового изображения.

Поскольку у нас изображение корабля, ожидаемый индекс класса равен 8.

Входное изображение будет получено из тестового набора данных. Следовательно, нам нужно будет реализовать функцию на Python для преобразования изображения, хранящегося в формате NumPy, в C-массив.

Как это делается...

Выполните следующие действия, чтобы из тестового набора данных сгенерировать заголовочный файл C, содержащий изображение корабля.

1. Напишите функцию для преобразования 1D-массива NumPy значений `np.int8` в единую строку целых значений, разделенных запятыми:

```
def array_to_str(data):
    NUM_COLS = 12
    val_string = ''
    for i, val in enumerate(data):
         val_string += str(val)
         if (i + 1) < len(data):
              val_string += ','
         if (i + 1) % NUM_COLS == 0:
              val_string += '\n'
    return val_string
```

В приведенном коде переменная NUM_COLS ограничивает количество значений в одной строке. В нашем случае NUM_COLS имеет значение 12, чтобы мы могли добавлять символ новой строки после каждых 12 значений.

2. Напишите функцию для генерации файла заголовка C, содержащего входное тестовое изображение, хранящееся в массиве int8_t. Для этого у вас может быть строка шаблона со следующими полями:
   - размер массива (size);
   - значения, которые нужно поместить в массив (data);
   - индекс класса, присвоенного входному изображению (ilabel).

```
def gen_h_file(size, data, ilabel):
    str_out = f'int8_t g_test[] = '
    str_out += "\n{\n"
    str_out += f'{data}'
    str_out += '};\n'
    str_out += f"const int g_test_len = {size};\n"
    str_out += f"const int g_test_ilabel = {ilabel};\n"
    return str_out
```

Как вы можете видеть из предыдущего кода, функция ожидает, что {data} будет представлять собой одну строку целых значений, разделенных запятыми.

3. Создайте pandas DataFrame из тестового набора данных CIFAR-10:

```
imgs = list(zip(test_imgs, test_lbls))
cols = [Image, 'Label']
df = pd.DataFrame(imgs, columns = cols)
```

4. Выделите из pandas DataFrame только изображения кораблей:

```
cond = df['Label'] == 8
ship_samples = df[cond]
```

В приведенном коде 8 – это индекс класса ship.

5. Запустите просчет по всем изображениям кораблей:

```
c_code = ""

for index, row in ship_samples.iterrows():
    i_value = np.asarray(row['Image'].tolist())
    o_value = np.asarray(row['Label'].tolist())
    o_pred_f32 = classify(i_value, o_value)
```

6. Проверьте, выводится ли изображение корабля. Если это так, преобразуйте входное изображение в C-байтовый массив и выйдите из цикла:

```
    if np.argmax(o_pred_f32) == o_value:
         i_value_f32 = i_value / i_scale + i_zero_point
         i_value_s8 = i_value_f32.astype(dtype=np.uint8)
         i_value_s8 = i_value_s8.ravel()
```

```
        # Generate a string from NumPy array
        val_string = array_to_str(i_value_s8)

        # Generate the C header file
        c_code = gen_h_file(
        i_value_s8.size, val_string, "8")
        break
```

7. Сохраните сгенерированный код в файле *input.h*:

```
with open("input.h", 'w') as file:
    file.write(c_code)
```

Вы можете загрузить файл `input.h`, содержащий входное тестовое изображение, с левой панели Colab.

Подготовка основы проекта TFLu

Всего несколько шагов отделяют нас от завершения этого проекта. Теперь, когда у нас есть входное тестовое изображение, мы можем покинуть среду Colab и сосредоточиться на приложении с Zephyr OS.

В этом рецепте мы подготовим каркас проекта TFLu из имеющегося образца TFLu *hello_world*, доступного в Zephyr SDK.

Файлы C *main.c*, *main_functions.cc* и *main_functions.h*, содержащие код, упомянутый в этом рецепте, доступны по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter07/ZephyrProject/Skeleton.

Подготовка

Цель этого раздела – обеспечить основу для запуска нового проекта TFLu на ОС Zephyr с нуля.

Самый простой способ создать проект – скопировать и отредактировать один из готовых образцовых примеров для TFLu. Образцы доступны в папке *~/zephyrproject/zephyr/samples/modules/tflite-micro*. На момент написания этого текста есть два готовых к использованию образцовых примера:

- *hello_world*: образец, показывающий основы TFLu для воспроизведения синусоидальной функции: https://docs.zephyrproject.org/latest/samples/modules/tflite-micro/hello_world/README.html;
- *magic_wand*: образец, показывающий, как реализовать приложение TFLu для распознавания жестов с помощью данных акселерометра: https://docs.zephyrproject.org/latest/samples/modules/tflite-micro/hello_world/README.html.

В этом примере мы будем основывать наше приложение на образце *hello_world*. На следующем рисунке показано, что находится в каталоге образца:

Рис. 7.8 ❖ Содержимое папки с образцовым примером *hello_world*

Папка *hello_world* содержит три вложенные папки, для нас интерес представляет *src/*, содержащая исходный код приложения. Не все файлы в этой папке необходимы для нашего проекта. Например, *assert.cc*, *константы.h*, *константы.c*, *model.cc*, *модель.h*, *output_handler.cc* и *output_handler.h* требуются только для примера развертывания синусоиды. Файлы, необходимые для нового проекта TFLu, следующие:

- *main.c*: этот файл содержит стандартную функцию C/C++ `main()`, отвечающую за запуск и завершение выполнения программы. Функция `main()` состоит из функции `setup()`, вызываемой один раз, и функции `loop()`, выполняемой 50 раз. Таким образом, функция `main()` более или менее повторяет поведение программы Arduino;
- *main_functions.h* и *main_functions.cc*: эти файлы содержат объявление и определение функций `setup()` и `loop()`.

Наконец, файлы *CMakeList.txt* и *prj.conf* в каталоге *hello_world* необходимы для сборки приложения. Мы узнаем больше об этих файлах в последнем примере этой главы.

Как это делается...

Для создания нового проекта TFLu откройте терминал и выполните следующие действия.

1. Перейдите в каталог *~/zephyrproject/zephyr/samples/modules/tflitemicro/* и создайте новую папку с именем *cifar10*:

```
$ cd ~/zephyrproject/zephyr/samples/modules/tflite-micro/
$ mkdir cifar10
```

2. Скопируйте содержимое каталога *hello_world* в *cifar10*:

```
$ cp -r hello_world/* cifar10
```

3. Перейдите в каталог *cifar10* и удалите следующие файлы из каталога *src/*: *constants.h*, *constants.c*, *model.c*, *model.h*, *output_handler.cc*, *output_handler.h* и *assert.cc*. Эти файлы могут быть удалены, поскольку они требуются только для примера синусоидальной волны, как описано в разделе «*Подготовка*» выше.
4. Скопируйте файлы *model.h* и *input.h*, созданные в двух предыдущих примерах, в папку *cifar10/src*.
   После выполнения этих действий папка *cifar10/src* должна содержать следующие файлы:

Рис. 7.9 ❖ Содержимое папки *hello_word/src*

Прежде чем продолжить, убедитесь, что у вас есть файлы, показанные на этом скриншоте.

Теперь откройте ваш С-редактор по умолчанию (например, **Vim**) для внесения изменений в код, содержащийся в файлах *main.c* и *main_functions.cc*.

5. Откройте файл *main.c* и замените `for (int i = 0; i < NUM_LOOPS; i++)` на `while(true)`. Код в *main.c* должен выглядеть следующим образом:

```
int main(int argc, char *argv[]) {
    setup();
    while(true) {
         loop();
    }
    return 0;
}
```

Приведенный код в точности повторяет поведение скетча Arduino, где `setup()` вызывается один раз, а `loop()` повторяется бесконечно.

6. Откройте *main_functions.cc* и удалите следующее:
   - *constants.h* и *output_handler.h* из списка заголовочных файлов;
   - переменную `inference_count` и все ее варианты использования. Эта переменная не потребуется в нашем приложении;
   - код внутри функции `loop()`.

   Затем замените `g_model` в файле *model.h* именем массива. Переменная `g_model` используется при вызове `tflite::getModel()`.

Теперь, когда у нас готова структура проекта, мы можем, наконец, реализовать наше приложение.

Создание и запуск приложения TFLu на QEMU

Каркас проекта Zephyr готов, поэтому для распознавания входного тестового изображения нам просто нужно доработать приложение.

В этом примере мы увидим, как создать приложение TFLu и запустить программу на эмулируемом микроконтроллере на основе Arm Cortex-M3.

Файлы *main.c, main_functions.cc* и *main_functions.h*, содержащие код, разбираемый в этом примере, доступны по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter07/ZephyrProject/CIFAR10.

Подготовка

Большинство компонентов, необходимых для разработки этого примера, относятся к TFLu и уже обсуждались в главе 3 «*Создание метеостанции с помощью библиотеки TensorFlow Lite for microcontrollers*» и главе 5 «*Распознавание интерьеров помещений с помощью TensorFlow Lite for microcontrollers и Arduino Nano*». Однако есть одна небольшая деталь TFLu, оказывающая большое влияние на использование памяти программ, которую мы еще не обсуждали.

В этом разделе мы поговорим об интерфейсе `tflite::MicroMutableOpResolver`.

Как мы знаем из наших предыдущих проектов, интерпретатор TFLu отвечает за подготовку вычислений для данной модели. Одна из вещей, которую должен знать интерпретатор, – указатель функции для каждого выполняемого оператора. До сих пор мы предоставляли эту информацию с помощью `tflite::AllOpsResolver`. Однако его не рекомендуется использовать из-за интенсивного использования памяти программ. Этот интерфейс может помешать созданию нашего приложения из-за малого объема памяти программ на целевом устройстве. Поэтому TFLu предлагает более эффективный альтернативный интерфейс `tflite::MicroMutableOpResolver` для загрузки только тех операторов, которые требуются модели.

Чтобы узнать, какие операторы потребуются модели, вы можете визуализировать файл модели TFLite (*.tflite*) с помощью веб-приложения Netron (https://netron.app).

Как это делается...

Давайте начнем этот рецепт с изучения архитектуры файла модели TFLite CIFAR-10 (*cifar10.tflite*) с помощью Netron.

На рис. 7.10 показан фрагмент нашей модели, визуализированный с помощью этого инструмента.

Проверка модели с помощью Netron показывает, что модель использует только пять операторов: **Conv2D**, **DepthwiseConv2D**, **MaxPool2D**, **Reshape** и **FullyConnected**. Эта информация будет использована для инициализации `tflite::MicroMutableOpResolver`.

Теперь откройте свой С-редактор по умолчанию и откройте файл *main_functions.cc*.

Для создания приложения TFLu выполните следующие действия.

1. Используйте директивы `#include` для добавления заголовочного файла входного тестового изображения (*input.h*):

```
#include "input.h"
```

2. Увеличьте размер памяти, требуемой интерпретатором TFLu (`tensor_arena_size`), до 52 000:

```
constexpr int tensor_arena_size = 52000;
```

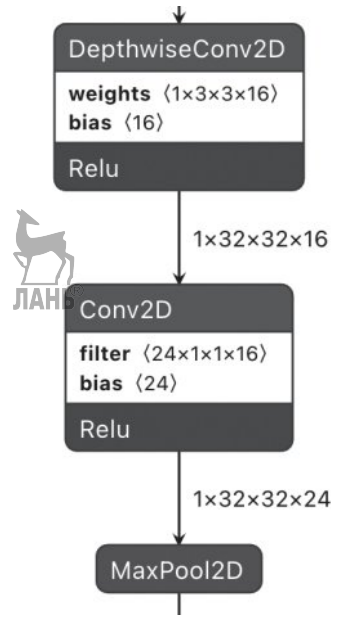


Рис. 7.10 ❖ Визуализация фрагмента модели CIFAR-10 в Netron (любезно предоставлено *netron.app*)

Исходное имя этой переменной – `kTensorArenaSize`. Чтобы сохранить соответствие с соглашением об именовании в нижнем регистре, используемым в книге, мы переименовали эту переменную в `tensor_arena_size`.

TFLu tensor arena – это часть памяти, выделяемая пользователем для размещения сетевых входных, выходных, промежуточных тензоров и других структур данных, требуемых TFLu. Размер этой части должен быть кратен 16, чтобы данные были выровнены по 16 байтам.
Как мы видели из дизайна модели CIFAR-10, ожидаемое использование оперативной памяти для расчета модели составляет порядка 44 Кбайт. Следовательно, 52 000 байт подходят для нашего случая, потому что это больше 44 Кбайт, кратно 16 и меньше максимального объема оперативной памяти в 64 Кбайт.

3. Замените `uint8_t tensor_arena[tensor_arena_size]` на `uint8_t *tensor_arena = nullptr`:

```
uint8_t *tensor_arena = nullptr;
```

Фрагмент памяти `tensor arena` слишком велик для размещения в стеке. Следовательно, мы должны динамически выделять эту память в функции `setup()`.

4. Объявите глобальный объект `tflite::MicroMutableOpResolver` для загрузки только операций, необходимых для запуска модели CIFAR-10:

```
tflite::MicroMutableOpResolver<5> resolver;
```

Этот объект создается с помощью предоставления максимального количества различных операций, которые требуются модели в качестве типовых аргументов.

5. Объявите две глобальные переменные для выходных параметров квантизации:

```
float o_scale = 0.0f;
int32_t o_zero_point = 0;
```

6. В функции setup() удалите экземпляр объекта tflite::AllOpsResolver. Затем загрузите операторы, используемые моделью, в объект tflite::MicroMutableOpResolver перед инициализацией интерпретатора TFLu:


```
resolver.AddConv2D();
resolver.AddDepthwiseConv2D();
resolver.AddMaxPool2D();
resolver.AddReshape();
resolver.AddFullyConnected();
static tflite::MicroInterpreter static_interpreter(model, resolver,
tensor_arena, tensor_arena_size, error_reporter);
interpreter = &static_interpreter;
```

7. В функции `setup()` получите выходные параметры квантизации из выходного тензора:

```
const auto* o_quantization =
reinterpret_cast<TfLiteAffineQuantization*>(
output->quantization.params);
o_scale = o_quantization->scale->data[0];
o_zero_point = o_quantization->zero_point->data[0];
```

8. В функции loop() инициализируйте входной тензор содержимым входного тестового изображения:


```
for(int i = 0; i < g_test_len; i++) {
     input->data.int8[i] = g_test[i];
}
```

Далее запустите расчет:

```
TfLiteStatus invoke_status = interpreter->Invoke();
```

9. После окончания вычислений выведите выходной класс с наивысшим баллом:

```
size_t ix_max = 0;
float pb_max = 0;
for (size_t ix = 0; ix < 10; ix++) {
  int8_t out_val = output->data.int8[ix];
  float pb = ((float)out_val - o_zero_point) * o_scale;
  if(pb > pb_max) {
```

```
            ix_max = ix;
            pb_max = pb;
        }
    }
```

Представленный код выполняет итерацию по квантизованным выходным значениям и возвращает класс (в переменной `ix_max`) с наивысшим баллом.

10. В конце проверьте, равен ли результат классификации (`ix_max`) индексу имени, присвоенного входному тестовому изображению (`g_test_label`):

```
if(ix_max == g_test_ilabel) {
```

Если да, выведите сообщение «*CORRECT classification!*» и результат классификации:

```
    static const char *label[] = {"airplane", "automobile", "bird",
"cat", "deer", "dog", "frog", "horse", "ship", "truck"};
    printf("CORRECT classification! %s\n", label[ix_max]);
    while(1);
}
```

Теперь откройте терминал и используйте следующую команду создания проекта для `qemu_cortex_m3`:

```
$ cd ~/zephyrproject/zephyr/samples/modules/tflite-micro/cifar10
$ west build -b qemu_cortex_m3 .
```

Через несколько секунд инструмент `west` должен вывести в терминале следующую таблицу, подтверждающую, что программа была успешно скомпилирована:

Memory region	Used Size	Region Size	%age Used
FLASH:	137668 B	256 KB	52.52%
SRAM:	4536 B	64 KB	6.92%
IDT_LIST:	0 GB	2 KB	0.00%

Рис. 7.11 ❖ Сводка использования памяти

Из сводки, сгенерированной инструментом `west`, вы можете видеть, что наше приложение на базе CIFAR-10 использует 52.57 % программной памяти (FLASH) и 6.92 % оперативной памяти (SRAM).

Однако использование оперативной памяти не должно вводить нас в заблуждение. На самом деле в сводке не учитывается объем памяти, который мы выделяем динамически. Следовательно, к 4536 байтам, статически выделенным в оперативной памяти, мы должны добавить 52 000 байт памяти `tensor_arena`, что приводит нас к 88 % использования оперативной памяти.

Теперь, когда приложение создано, мы можем запустить его на виртуальной платформе с помощью следующей команды:

```
$ west build -t run
```

Инструмент `west` загрузит виртуальное устройство и выведет следующий текст, подтверждающий, что модель правильно классифицировала изображение корабля:

```
-- west build: running target run
[1/1] To exit from QEMU enter: 'CTRL+a, x'[QEMU] CPU: cortex-m3
qemu-system-arm: warning: nic stellaris_enet.0 has no peer
Timer with period zero, disabling
*** Booting Zephyr OS build v2.7.99-1639-g73a957e4b316  ***
CORRECT classification!: ship
```

Рис. 7.12 ❖ Ожидаемый результат расчета модели

Как вы можете видеть на рисунке, виртуальное устройство выводит сообщение «*CORRECT classification!*», подтверждающее успешное выполнение нашей малой модели CIFAR-10.

Глава 8

К следующему поколению TinyML с microNPU

Здесь мы находимся на последней остановке нашего путешествия в мир TinyML. Хотя эта глава является окончанием книги, на самом деле она посвящена теме на самом переднем крае машинного обучения (ML). В ходе повествования мы узнали, насколько важно энергопотребление для эффективных и долговечных приложений TinyML. И все же именно вычислительные мощности – ключ к открытию новых вариантов использования и к тому, чтобы сделать «вещи» вокруг нас еще более интеллектуальными. По этой причине был разработан новый усовершенствованный процессор с большей вычислительной мощностью и одновременно энергоэффективностью при рабочих нагрузках ML. Это микронейронный процессор **Micro-Neural Processing Unit** (**microNPU**).

В этой заключительной главе мы узнаем, как запустить квантизованную модель CIFAR-10 на виртуальном Arm Ethos-U55 microNPU.

Мы начнем эту главу с изучения того, как работает этот процессор в зависимости от программного обеспечения для сборки и запуска модели на виртуальной платформе **Arm Corstone-300 Fixed Virtual Platform** (**Corstone-300 FVP**). Далее мы будем использовать компилятор **TVM**[80] для преобразования предварительно обученной модели **TensorFlow Lite** (**TFLite**) в код C/C++. В конце концов мы покажем, как скомпилировать и развернуть код, сгенерированный TVM, в Corstone-300 FVP для выполнения вычислений с помощью Ethos-U55 microNPU.

Цель этой главы – ознакомиться с Arm Ethos-U55 microNPU, новым поколением процессоров для работы ML на микроконтроллерах.

---

[80] TVM – компилятор глубокого обучения с открытым исходным кодом, который может быть ориентирован на различные аппаратные устройства. Подробнее см. далее раздел «*Установка TVM с поддержкой Arm Ethos-U*».

Поскольку некоторые инструменты, представленные в этой главе, все еще находятся в стадии интенсивной разработки, существует вероятность того, что наши рекомендации могут измениться в будущем. Поэтому мы рекомендуем проверять репозитории библиотек программного обеспечения, используя указанный далее хеш Git commit[81].

В этой главе мы собираемся реализовать следующие примеры.

- Настройка Arm Corstone-300 FVP.
- Установка TVM с поддержкой Arm Ethos-U.
- Установка набора инструментов Arm и стека драйверов Ethos-U.
- Генерация C-кода с помощью TVM.
- Генерация C-байтовых массивов для входа, выхода и меток.
- Создание и запуск модели на Arm Ethos-U55.

Технические требования

Чтобы реализовать все практические примеры этой главы, нам понадобится следующее:

- ноутбук/ПК с Ubuntu 18.04+ на x86-64.

Исходный код и дополнительные материалы доступны в папке Chapter08 по адресу https://github.com/PacktPublishing/TinyML-Cookbook/tree/main/Chapter08.

Настройка Arm Corstone-300 FVP

Arm Ethos-U55 – это первый микропроцессор, разработанный Arm для расширения возможностей ML на микроконтроллерах на базе Cortex-M. К сожалению, на момент написания этой статьи аппаратное обеспечение этого нового процессора пока еще недоступно. Однако Arm предлагает бесплатный виртуальный инструмент **Fixed Virtual Platform** (**FVP**) на базе системы Arm Corstone-300, позволяющий проводить эксперименты с моделями ML на этом процессоре без необходимости использования физических устройств.

В этом примере мы более подробно расскажем о вычислительных возможностях Arm Ethos-U55 microNPU и установим Corstone-300 FVP.

Подготовка

Давайте начнем с представления Corstone-300 FVP и Ethos-U55 microNPU.

Corstone-300 FVP (https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps) представляет собой виртуальную платформу, основанную на процессоре Arm Cortex-M55 и Ethos-U55 microNPU.

[81] О системе контроля версий Git см. https://habr.com/ru/company/intel/blog/344962/.

Arm Ethos-U55 (https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u55) – это процессор для расчетов моделей ML, который работает параллельно с процессором Cortex-M, как показано на следующей блок-схеме:

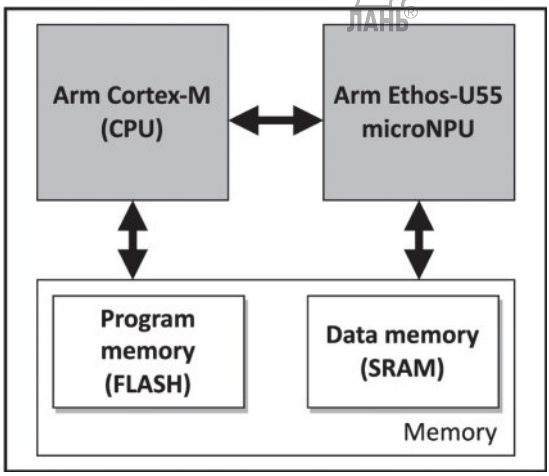


Рис. 8.1 ❖ Микроконтроллер с процессором Arm Cortex-M и Ethos-U55 microNPU

Роль центрального процессора (CPU) заключается в управлении загруженностью microNPU, который независимо выполняет расчет модели ML. Arm Ethos-U55 был разработан для эффективного вычисления большинства элементарных операций, которые мы можем найти в квантизованных 8-битных/16-битных нейронных сетях. Это такие операции, как **Multiply and Accumulate** (**MAC**), лежащие в основе полносвязных (*fully connected*) и глубинных слоев свертки (*depthwise convolution*).

В следующей таблице приведены сведения о некоторых операторах, поддерживаемых Arm Ethos-U55:

Таблица 8.1. Некоторые операции, поддерживаемые Arm Ethos-U55 microNPU[82]

Convolution 2D	Depthwise convolution 2D	Deconvolution	Max pooling
Average pooling	Fully connected	LSTM/GPU	Add/Sub/Mul
Softmax	Relu/Relu6/Tanh/Sigmoid	Reshape	многое другое…

С точки зрения программирования микроконтроллера нам все еще нужно предоставить модель в виде программы на C/C++ и загрузить ее в микроконтроллер. Кроме того, веса, смещения и параметры квантизации все еще могут храниться в памяти программ, в то время как входные и выходные тензоры хранятся в SRAM, как показано на следующем рисунке:

[82] Толкование приведенных в таблице англоязычных терминов см. в предыдущих главах книги, а также в дальнейшем по тексту этой главы.

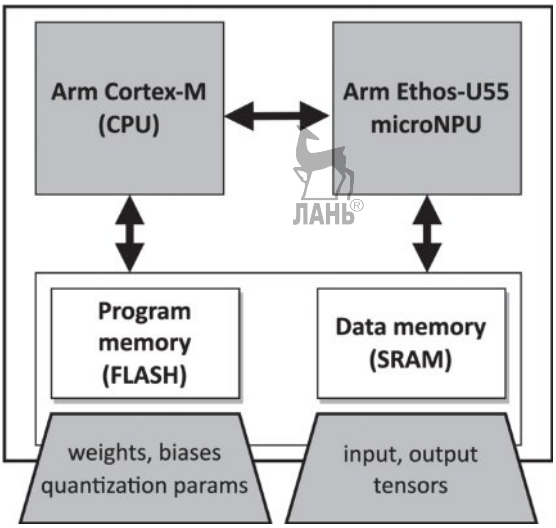


Рис. 8.2 ❖ Веса (*weights*) и смещения (*biases*) все еще могут храниться в памяти программы

Таким образом, ничего не меняется по сравнению с тем, что мы видели в предыдущих главах относительно распределения памяти для параметров ML и входных/выходных тензоров. Что отличается от традиционных вычислений на процессоре Cortex-M, так это то, как мы программируем запуск модели на Arm Ethos-U55. При запуске расчета модели на микропроцессоре программа представляет собой последовательность команд (т. е. поток команд), указывающий процессору, какие операции следует выполнить и где читать/записывать данные из памяти и в память.

Как только программа загружена в микроконтроллер, мы можем разгрузить вычисления на microNPU, указав область памяти для потока команд и область SRAM, выделенную для входных и выходных тензоров. Arm Ethos-U55 выполняет все команды независимо, записывая выходные данные в определенную пользователем область памяти данных и запуская прерывание по завершении. Центральный процессор может использовать прерывание, чтобы знать, когда следует считывать выходные данные.

Как это делается...

Откройте терминал и создайте новую папку с именем *project_npu* в домашнем каталоге (~/):

```
$ cd ~/ && mkdir project_npu
```

Войдите в папку *~/project_npu* и создайте три папки с именами *binaries*, *src* и *sw_libs*:

```
$ cd ~/project_npu
```

```
$ mkdir binaries
$ mkdir src
$ mkdir sw_libs
```

Эти три папки будут содержать следующее:

- двоичные файлы для сборки и запуска приложения на Arm Corstone-300 FVP *(binaries/)*;
- исходный код приложения (*src/*);
- программные библиотеки, необходимые для проекта (*sw_libs/*).

Теперь выполните следующие действия, чтобы установить Arm Corstone-300 на компьютер с Ubuntu/Linux.

1. Откройте веб-браузер и перейдите в Arm Ecosystem FVPs (https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps).
2. Нажмите на **Corstone-300 Ecosystem FVPs**, а затем нажмите на кнопку **Download Linux**, как показано на следующем скриншоте:

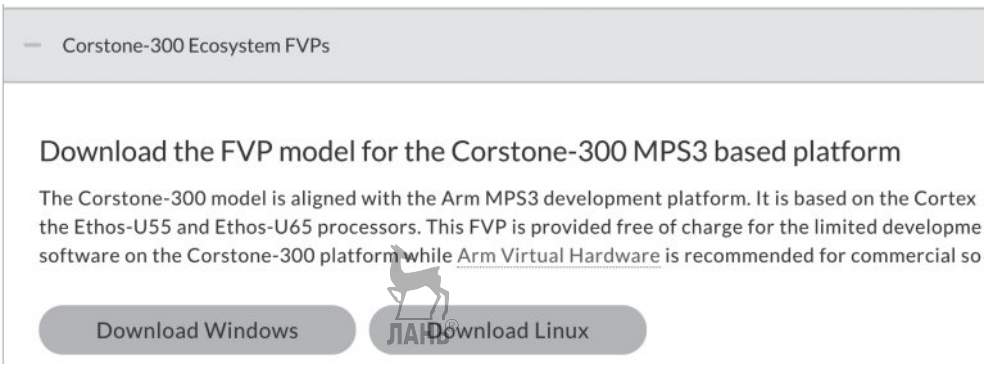


Рис. 8.3 ❖ Кнопка загрузки Linux-версии Corstone-300 FVP

Загрузите файл *.tgz* и извлеките сценарий *FVP_Corstone_SSE-300.sh*.

3. Снова откройте терминал и сделайте *FVP_Corstone_SSE-300.sh* исполняемым:

```
$ chmod +x FVP_Corstone_SSE-300.sh
```

4. Выполните сценарий *FVP_Corstone_SSE-300.sh*:

```
$ ./FVP_Corstone_SSE-300.sh
```

Следуйте инструкциям на терминале, чтобы установить двоичные файлы для Corstone-300 FVP в папку *~/project_npu/binaries*. Для этого введите `~/project_npu/binaries/FVP_Corstone_SSE-300` в ответ на вопрос **Where would you like to install to?** (*Куда бы вы хотели установить?*)

5. Обновите переменную среды `$PATH`, чтобы сохранить путь к двоичным файлам Corstone-300. Для этого откройте файл *.bashrc* с помощью любого текстового редактора (например, **gedit**):

```
$ gedit ~/.bashrc
```

Затем добавьте следующую строку в конец файла:

```
export PATH=~/project_npu/binaries/FVP_Corstone_SSE-300/
models/Linux64_GCC-6.4:$PATH
```

Эта строка обновляет переменную среды $PATH с указанием местоположения двоичных файлов Corstone-300.
Сохраните и закройте файл.

6. Перезагрузите файл *.bashrc* в терминале:

```
$ source ~/.bashrc
```

В качестве альтернативы использованию команды `source` вы можете просто закрыть и повторно открыть терминал.

7. Проверьте, установлены ли двоичные файлы Corstone-300, распечатав информацию о версии FVP_Corstone_SSE_Ethos-U55:

```
$ FVP_Corstone_SSE_Ethos-U55 -version
```

Если переменная среды `$PATH` была успешно обновлена, приведенная команда должна в терминале вернуть версию Corstone-300:

```
~ $FVP_Corstone_SSE-300_Ethos-U55 --version

Fast Models [11.15.24 (Aug 17 2021)]
Copyright 2000-2021 ARM Limited.
All Rights Reserved.
```

Рис. 8.4 ❖ Сообщение, выведенное в результате команды

Как показано на рисунке, команда возвращает версию исполняемого файла Corstone-300.

Виртуальное оборудование с Arm Cortex-M55 и Ethos-U55 теперь установлено и готово к использованию.

Установка TVM с поддержкой Arm Ethos-U

В предыдущем примере мы кратко рассказали о программе Ethos-U55, потоке команд, используемом для указания операций, которые должны выполняться на microNPU. Однако как генерируется поток команд? В этой главе мы будем использовать компилятор глубокого обучения TVM, предназначенный для генерации кода C из модели ML для конкретного целевого устройства.

В этом примере мы узнаем, что такое TVM, подготовив среду разработки, которую будем использовать в этой главе позже.

Подготовка

Цель этого примера – установить компилятор TVM из исходного кода. Для установки необходимы следующие предварительные условия:

- CMake 3.5.0 или более поздней версии,
- компилятор C++ с поддержкой C++14 (например, g++ 5 или более поздней версии),
- LLVM 4.0 или более поздней версии,
- Python 3.7 или Python 3.8.

Прежде чем приступить к работе, мы рекомендуем установить инструмент **Python virtual environment** (**virtualenv**) для создания изолированной среды Python. Вы можете обратиться к главе 7 «*Запуск модели TinyML CIFAR-10 на виртуальной платформе ОС Zephyr*», чтобы узнать, как установить и активировать виртуальную среду.

Однако, прежде чем показать, как установить TVM, мы хотим дать обзор основных характеристик этой технологии, поскольку вы, возможно, не имеете предварительных знаний об этом инструменте и стеках компиляторов DL.

Мотивация, лежащая в основе TVM

TensorFlow Lite для микроконтроллеров (TFLu) – это библиотека программного обеспечения, которая сделала возможным создание DL-приложений, описанных в предыдущих главах. TFLu использует преимущества оптимизированных для конкретного поставщика библиотек операторов (**performance libraries**) для эффективного выполнения модели на целевом устройстве. Например, TFLu может делегировать вычисления библиотеке CMSIS-NN, что обеспечивает превосходную производительность и низкое использование памяти на микроконтроллерах на базе Arm Cortex-M.

Как правило, эти библиотеки предоставляют набор вручную созданных операторов, оптимизированных для архитектуры конкретного процессора (например, Arm Cortex-M0 или Cortex-M4) и базовых аппаратных возможностей. С учетом необходимости переноса DL на широкий спектр устройств и многочисленных функций для оптимизации становится очевидной необходимость значительных инженерных усилий, необходимых для разработки таких библиотек. Поэтому исследовательская группа из Вашингтонского университета, движимая необходимостью обеспечить эффективные акселераторы DL на различных платформах, разработала TVM – стек компиляторов для генерации оптимизированного кода из моделей DL.

Как TVM оптимизирует работу модели

Apache TVM (https://tvm.apache.org/) – это полноценный компилятор с открытым исходным кодом, целью которого является преобразование моделей DL (например, моделей TFLite) в оптимизированный код для любых типов процессоров:

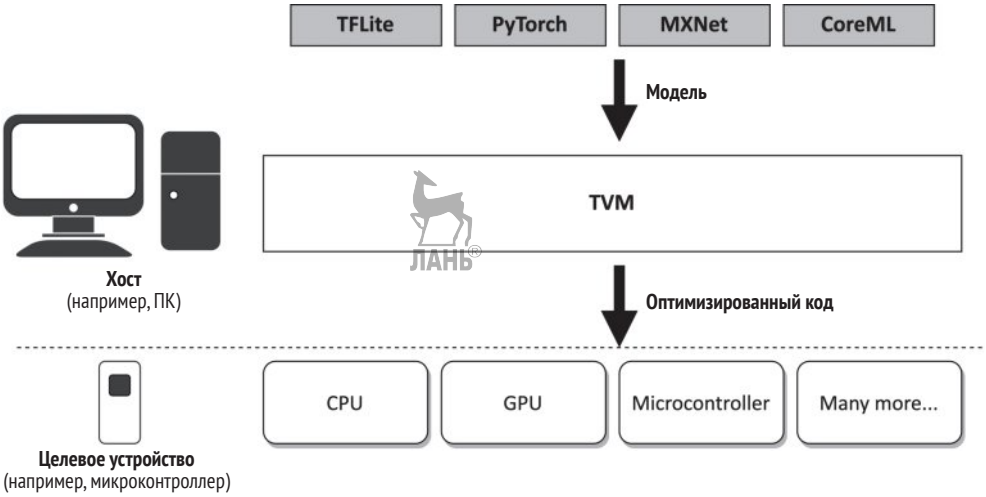


Рис. 8.5 ❖ TVM генерирует оптимизированный код из предварительно обученной модели

Существенным преимуществом наличия стека компиляторов является автоматическое получение эффективного кода для новых акселераторов DL без необходимости быть экспертом по оптимизации производительности.

Как показано на рисунке выше, TVM принимает предварительно подготовленную модель в различных форматах (например, TFLite и PyTorch) и выполняет оптимизацию кода в два основных этапа, как показано на следующей блок-схеме:

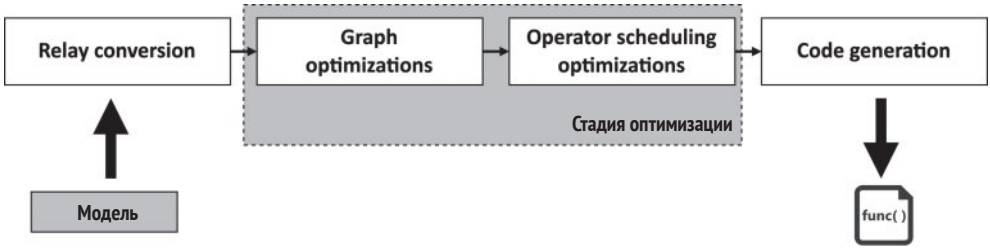


Рис. 8.6 ❖ Основные этапы оптимизации в TVM

На блок-схеме показано, как TVM сначала преобразует входную модель во внутренний язык нейронной сети высокого уровня (**relay**). Далее компилятор выполняет первый шаг оптимизации на уровне модели (**graph level optimizations**). Слияние (**fusion**) – распространенный метод оптимизации, применяемый на этом уровне, целью которого является объединение двух или более операторов вместе для повышения эффективности вычислений. Когда TVM обнаруживает шаблоны слияния, он преобразует модель, заменяя исходные операторы новым слитым, как показано в следующем примере:

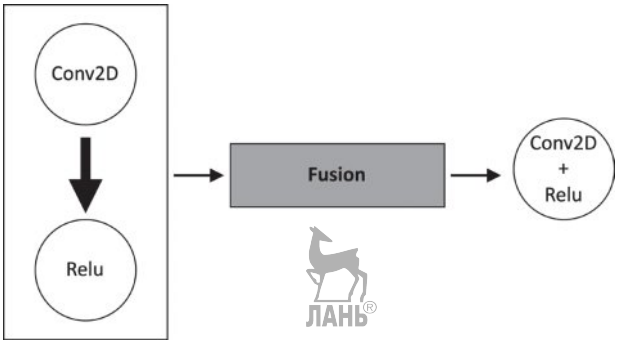


Рис. 8.7 ❖ Слияние Conv2D + ReLU

В этом примере операция *fusion* нацелена на создание единого оператора для 2D-свертки (Conv2D) и активации ReLU вместо двух отдельных, как в оригинальной модели.

Когда происходит слияние, как правило, время вычислений уменьшается, поскольку код содержит меньше арифметических инструкций и перемещений из/в основную память.

Второй шаг оптимизации, выполняемый TVM, выполняется на уровне оператора (**operator scheduling**), целью которого является поиск наиболее эффективного способа выполнения каждого оператора на целевом устройстве. Эта оптимизация выполняется на уровне кода и влияет на принятие вычислительных стратегий, таких как разбиение на макрооперации (*tiling*), развертывание циклов (*unrolling*) и векторизация (*vectorization*). Как легко сообразить, наилучший метод вычисления будет зависеть от целевой платформы.

То, что мы только что описали, – лишь основные моменты, чтобы дать вам общую картину того, как работает эта технология компилятора. Для получения дополнительной информации об архитектуре TVM, пожалуйста, обратитесь к вводному руководству по TVM, в котором содержится пошаговое объяснение оптимизации модели: https://tvm.apache.org/docs/tutorial/introduction.html#sphxglr-tutorial-introduction-py.

Как это делается...

Установка TVM состоит из трех частей:

1) установки необходимых компонентов TVM,
2) создания библиотеки TVM C++ из исходного кода,
3) настройки среды Python.

Установить TVM можно с помощью следующих шагов.

1. Используйте Ubuntu **Advanced Packaging Tool** (**APT**) для установки необходимых компонентов TVM:

```
$ sudo apt-get install -y python3 python3-dev python3-setuptools
gcc libtinfo-dev zlib 1g-dev build-essential cmake
libedit-dev libxml2-dev llvm-dev
```

Проверьте версии Python, CMake, g++ и llvm-config:

```
$ python -version && cmake -version && g++ --version && llvm-config -version
```

Проверьте, соответствуют ли версии минимально необходимой версии для TVM, указанной в разделе «*Подготовка*». Если нет, вы можете перейти по следующим ссылкам, чтобы обновить версии вручную:

- CMake: https://cmake.org/download/,
- LLVM: https://apt.llvm.org/,
- g++: https://gcc.gnu.org/,
- Python: https://www.python.org/downloads/.

2. Войдите в папку *~/project_npu* и клонируйте исходный код TVM из репозитория GitHub:

```
$ git clone -recursive https://github.com/apache/tvm tvm
```

3. Войдите в папку *tvm/* и сделайте так, чтобы TVM указывал на Git commit dbfbd164c3:

```
$ cd ~/project_npu/tvm
$ git checkout dbfbd164c3
```

4. Создайте новый каталог с именем *build* внутри папки *tvm/*:

```
$ mkdir build
```

5. Скопируйте файл *cmake/config.cmake* в каталог *build/*:

```
$ cp cmake/config.cmake build
```

6. Отредактируйте файл *build/config.cmake*, чтобы включить поддержку **microTVM**, **Ethos-U** и **LLVM**. Для этого вы должны указать в этом файле `set(USE_MICRO ON)`, `set(USE_LLVM ON)` и `set(USE_ETHOSU ON)`. Как мы увидим позже в этой главе, microTVM является расширением TVM для микроконтроллерных платформ.
7. Создайте библиотеку TVM C++ из исходного кода:

```
$ cd build
$ cmake ..
$ make -j8
```

Мы рекомендуем указать флаг `-j`, чтобы запускать процесс сборки одновременно для разных заданий. Количество заданий должно быть установлено соответственно количеству ядер, доступных в системе, например 8 для системы с восемью ядрами.

8. Обновите переменную окружения `$PYTHONPATH`, чтобы указать Python, где найти библиотеку, созданную на предыдущем шаге. Для этого откройте файл *.bashrc* с помощью любого текстового редактора (например, *gedit*):

```
$ gedit ~/.bashrc
```

9. Добавьте следующую строку в конец файла:

```
export PYTHONPATH=~/project_npu/tvm/python:${PYTHONPATH}
```

   После обновления переменной среды `$PATH` сохраните и закройте файл.
10. Перезагрузите файл *.bashrc*:

```
$ source ~/.bashrc
```

   Если у вас в той же среде был активирован *virtualenv*, снова запустите виртуальную среду Python.
11. Проверьте, правильно ли Python разместил библиотеку TVM Python в директории *~/project_npu/tvm/python*:

```
$ python -c "import sys; print(sys.path)"
```

   Эта команда выводит список каталогов, которые интерпретатор Python проверяет для поиска модулей. Поскольку `sys.path` инициализируется из `PYTHONPATH`, в выведенном списке каталогов в консоли вы должны увидеть путь *~/project_npu/tvm/python*.
12. Установите необходимые компоненты Python для TVM:

```
$ pip3 install --user numpy decorator attrs scipy
```

13. Проверьте, правильно ли установлена TVM:

```
$ python -c "import tvm; print('HELLO WORLD')"
```

   Этот код должен вывести «*HELLO WORLD*» в терминале.
14. Установленные необходимые компоненты Python перечислены в *~/project_npu/tvm/apps/microtvm/ethosu/requirements.txt*:

```
$ cd ~/project_npu/tvm/apps/microtvm/ethosu
$ pip3 install -r requirements.txt
```

   TVM требует некоторые компоненты, установленные на этом шаге, для генерации кода Ethos-U55 microNPU.

TVM теперь может генерировать C-код для процессоров Cortex-M с Ethos-U microNPU.

Установка набора инструментов Arm и стека драйверов Ethos-U

TVM генерирует C-код для целевого устройства с использованием модели TFLite в качестве входных данных. Однако сгенерированный исходный код необходимо скомпилировать вручную, чтобы запустить его на Corstone-300 FVP. Кроме того, процессору Cortex-M55 требуются дополнительные программные библиотеки для управления вычислениями на микропроцессоре Ethos-U55.

В этом примере мы установим набор инструментов Arm GCC для кросс-компиляции кода для Arm Cortex-M55 и сохранения привязки к программным библиотекам, необходимым для нашего приложения.

Подготовка

В этом разделе мы дадим вам обзор трех оставшихся необходимых компонентов для нашего приложения: набора инструментов **Arm GCC**, драйвера **Ethos-U core driver** и библиотеки **Arm Ethos-U core platform**.

Corstone-300 FVP – это виртуальная платформа, основанная на Arm Cortex-M55, и при создании приложения для этого целевого устройства требуется специальный компилятор. Такой компилятор обычно называют кросс-компилятором, потому что целевой процессор (например, ArmCortex-M55) отличается от процессора компьютера, на котором создается приложение (например, x86-64). Для кросс-компиляции для Arm Cortex-M55 нам понадобится набор инструментов **GNU Arm Embedded toolchain** (https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads/product-release), предлагающий бесплатную коллекцию инструментов программирования, включая компилятор, компоновщик, отладчик и программные библиотеки. Набор инструментов доступен для различных операционных систем, таких как Linux, Windows и macOS.

Однако набор инструментов не единственное, что потребуется. Процессору Cortex-M55 требуется **Arm Ethos-U core driver** для разгрузки работы ML на Arm Ethos-U55 (https://review.mlplatform.org/plugins/gitiles/ml/ethos-u/ethos-u-core-driver/). Драйвер Arm Ethos-U core предлагает интерфейс для выполнения потоков команд на Ethos-U microNPU. Драйвер не зависит от ОС, что означает, что он не использует никаких примитивов ОС, таких как очереди (*queues*) или мьютексы (*mutexes*)[83]. Таким образом, он может быть кросс-скомпилирован для любого поддерживаемого процессора Cortex-M и работать с любой ОС реального времени (**RTOS**).

Последняя оставшаяся библиотека, необходимая для нашего приложения, – это **Arm Ethos-U core platform** (https://review.mlplatform.org/plugins/gitiles/ml/ethos-u/ethos-u-core-platform/). Это ПО в основном содержит примеры запуска ML на платформах Arm Ethos-U, включая Corstone-300 FVP. Из него мы будем использовать операцию Makefile для создания приложения.

Как это делается...

Откройте терминал и выполните следующие действия, чтобы установить GNU Arm Embedded toolchain и получить остальные программные компоненты для нашего приложения.

1. Войдите в папку *~/project_npu/binaries* и установите GNU Arm Embedded toolchain для Linux x86-64. Для этого создайте новую папку с именем *toolchain* в каталоге *~/project_npu/binaries*:

[83] О понятиях queues и mutexes см. на русском языке https://habr.com/ru/post/182610/.

```
$ cd ~/project_npu/binaries
$ mkdir toolchain
```

2. Загрузите GNU Arm Embedded toolchain. Удобно использовать инструмент `curl` и распаковать загруженный файл в папку *toolchain*:

```
$ gcc_arm='https://developer.arm.com/-/media/Files/downloads/gnu-rm/10-2020q4/
gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2?revision=ca0cbf9c-9de2-
491cac48-898b5bbc0443&la=en&hash=68760A8AE66026BCF99F05AC017A6A50C6FD832A'

$ curl --retry 64 -sSL ${gcc_arm} | \tar -C toolchain --strip-components=1 -jx
```

Эта операция может занять несколько минут, в зависимости от скорости интернет-соединения.

3. Откройте файл *.bashrc* с помощью любого текстового редактора (например, *gedit*):

```
$ gedit ~/.bashrc
```

4. Добавьте следующую строку в конец файла, чтобы включить путь к *toolchain* в переменную среды `$PATH`:

```
export PATH=~/project_npu/binaries/toolchain/gcc-armnone-eabi-10.3-2021.10/bin:$PATH
```

Теперь сохраните и закройте файл.

5. Перезагрузите файл *.bashrc*:

```
$ source ~/.bashrc
```

6. Проверьте, правильно ли установлен GNU Arm Embedded toolchain, распечатав список поддерживаемых процессоров:

```
$ arm-none-eabi-gcc -mcpu=.
```

Возвращаемый список поддерживаемых процессоров должен включать процессор Cortex-M55, как показано на следующем рисунке:

```
arm-none-eabi-gcc: note: valid arguments are: arm8 arm810 strongarm strongarm110 fa526
fa626 arm7tdmi arm7tdmi-s arm710t arm720t arm740t arm9 arm9tdmi arm920t arm920 arm922t
arm940t ep9312 arm10tdmi arm1020t arm9e arm946e-s arm966e-s arm968e-s arm10e arm1020e a
rm1022e xscale iwmmxt iwmmxt2 fa606te fa626te fmp626 fa726te arm926ej-s arm1026ej-s arm
1136j-s arm1136jf-s arm1176jz-s arm1176jzf-s mpcorenovfp mpcore arm1156t2-s arm1156t2f-
s cortex-m1 cortex-m0 cortex-m0plus cortex-m1.small-multiply cortex-m0.small-multiply c
ortex-m0plus.small-multiply generic-armv7-a cortex-a5 cortex-a7 cortex-a8 cortex-a9 cor
tex-a12 cortex-a15 cortex-a17 cortex-r4 cortex-r4f cortex-r5 cortex-r7 cortex-r8 cortex
-m7 cortex-m4 cortex-m3 marvell-pj4 cortex-a15.cortex-a7 cortex-a17.cortex-a7 cortex-a3
2 cortex-a35 cortex-a53 cortex-a57 cortex-a72 cortex-a73 exynos-m1 xgene1 cortex-a57.co
rtex-a53 cortex-a72.cortex-a53 cortex-a73.cortex-a35 cortex-a73.cortex-a53 cortex-a55 c
ortex-a75 cortex-a76 cortex-a76ae cortex-a77 neoverse-n1 cortex-a75.cortex-a55 cortex-a
76.cortex-a55 neoverse-v1 neoverse-n2 cortex-m23 cortex-m33 cortex-m35p cortex-m55 cort
ex-r52
```

Рис. 8.8 ❖ Список поддерживаемых процессоров должен включать cortex-m55

7. Войдите в папку *~/project_npu/sw_libs* и клонируйте библиотеку CMSIS:

```
$ cd ~/project_npu/sw_libs
$ git clone "https: //github.com/ARM-software/ CMSIS_5.git" cmsis
```

Далее проверьте версию 5.8.0:

```
$ cd cmsis
$ git checkout -f tags/5.8.0
$ cd ..
```

8. Войдите в папку *~/project_npu/sw_libs* и клонируйте драйвер ядра Arm Ethos-U:


```
$ cd ~/project_npu/sw_libs
$ git clone "https:// review.mlplatform.org/ml /ethos-u/ethos-u-core-driver" core_
driver
```

Далее проверьте версию 21.11:

```
$ cd core_driver
$ git checkout tags/21.11
$ cd ..
```

9. Клонируйте библиотеку Arm Ethos-U core platform:

```
$ git clone "https: //review.mlplatform.org/ml/ethos-u/ethos-u-core-platform" core_
platform
$ cd core_platform
```

Далее проверьте версию 21.11:

```
$ git checkout tags/21.11
$ cd ..
```


Теперь мы определенно готовы к обработке нашего приложения для запуска его на Corstone-300 FVP!

Генерация C-кода с помощью TVM

Компиляция модели TFLite в код C с помощью TVM делается просто. TVM требуется только входная модель, указание целевого устройства и одна командная строка для создания архива TAR со сгенерированным кодом C.

В этом примере мы покажем, как преобразовать предварительно подготовленную модель CIFAR-10 в код C с помощью microTVM, расширения TVM для развертывания на микроконтроллерах.

Скрипт Bash *compile_model_microtvm.sh*, содержащий команды, упомянутые в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter08/BashScripts/compile_model_microtvm.sh.

Подготовка

В этом разделе мы рассмотрим, как TVM может генерировать C-код, и объясним, что такое microTVM.

TVM – это технология компилятора DL, которую мы можем использовать в Python и в той же среде, где мы создаем, обучаем и квантизуем модель с помощью TFLite. Хотя изначально TVM предлагает Python API, существует альтернативный и более простой API, основанный на интерфейсе командной строки под названием **TVMC**.

TVMC – это драйвер командной строки, предоставляющий те же функции, которые TVM предлагает с помощью Python API, но с преимуществом уменьшения количества строк кода. Только одна командная строка потребуется для компиляции модели TFLite в C-код в нашем конкретном случае.

На этом этапе вы можете задаться вопросом: где мы можем найти инструмент TVMC?

TVMC является частью установки TVM Python, и вам просто нужно будет выполнить команду `python -m tvm.driver.tvmc compile <options>` в вашем терминале, чтобы скомпилировать модель TFLite. Параметры, требуемые командой компиляции, будут представлены в разделе «*Как это делается...*» далее.

Чтобы узнать больше о TVMC, мы рекомендуем ознакомиться со следующей документацией: https://tvm.apache.org/docs/tutorial/tvmc_command_line_driver.

Хотя мы сказали, что будем генерировать C-код из модели, традиционно TVM создает следующие выходные файлы:

- *.so*: библиотека C++, содержащая оптимизированные операторы для выполнения модели. Среда выполнения TVM C++ будет отвечать за загрузку этой библиотеки и выполнение вывода на целевом устройстве;
- *.json*: файл JSON, содержащий график вычислений и весá;
- *.params*: файл, содержащий параметры предварительно обученной модели.

К сожалению, эти три файла не подходят для развертывания на микроконтроллере по следующим причинам:

- микроконтроллеры не имеют модуля управления памятью (MMU), поэтому мы не можем загружать динамические библиотеки во время выполнения;
- веса хранятся во внешнем файле (.json), что не идеально для микроконтроллеров по двум причинам: первая заключается в том, что у нас может не быть ОС, предоставляющей API для чтения внешних файлов. Вторая в том, что веса, загруженные из внешнего файла, попадают в SRAM, которая обычно меньше, чем память программ.

По вышеуказанным причинам было предложено расширение TVM с целью получения подходящего для микроконтроллеров выходного сигнала: **microTVM**.

Запуск TVM на микроконтроллерах с помощью microTVM

microTBM (https://tvm.apache.org/docs/topic/microtvm/index.html) – это расширение TVM, предоставляющее альтернативный формат вывода, не требующий операционной системы и динамического выделения памяти.

Устройства без операционной системы обычно называют «чисто железными» (*bare-metal*) устройствами.

Формат вывода, на который мы ссылаемся, – это формат библиотеки моделей **Model Library Format** (**MLF**), представляющий собой архив TAR, содержащий C-код. Код, сгенерированный TVM/microTVM, необходимо будет интегрировать в приложение и скомпилировать для конкретной целевой платформы.

Как это делается...

1. Создайте новую папку с именем *build/* в каталоге *~/project_npu/src*:

```
$ cd ~/project_npu/src
$ mkdir build
```

2. Загрузите предварительно обученную квантизованную модель CIFAR-10 из репозитория TinyML-Cookbook GitHub: https://github.com/PacktPublishing/TinyMLCookbook/blob/main/Chapter08/cifar10_int8.tflite.
   В качестве альтернативы вы можете повторно использовать модель CIFAR-10, созданную вами в главе 7 «*Запуск модели TinyML CIFAR-10 на виртуальной платформе ОС Zephyr*».
   Сохраните модель в папке *~/project_npu/src/*.
3. Войдите в папку *~/project_npu/src/* и скомпилируйте модель CIFAR-10 в формат MLF с компилятором TVMC:

```
$ cd ~/project_npu/src/
$ python3 -m tvm.driver.tvmc compile \
--target="ethos-u -accelerator_config=ethos-u55-256, c" \
--target-c-mcpu=cortex-m55 \
--runtime=crt \
--executor=aot \
--executor-aot-interface-api=c \
--executor-aot-unpacked-api=1 \
--pass-config tir.disable_vectorize=1 \
--output-format=mlf \ cifar10_int8.tflite
```

   В приведенном коде мы передаем несколько аргументов подкоманде компиляции TVMC. Расшифруем самые важные из них:
   - `--target="ethos-u -accelerator_config=ethos-u55-256, c"`: этот параметр определяет целевые процессоры для вывода ML. В нашем случае у нас есть два целевых процессора: Arm Ethos-U55 и Cortex-M CPU. Основной целью является микропроцессор Ethos-U55. Как мы знаем, Ethos-U microNPU – это процессор, способный очень эффективно выполнять операции MAC. При передаче аргумента `ethos-`

`u55-256` мы сообщаем TVM, что вычислительный движок Ethos-U55 имеет 256 MAC. Это значение не программируется пользователем, а фиксируется аппаратно. Следовательно, Corstone-300 FVP должен использовать ту же конфигурацию Ethos-U55 для правильного запуска приложения. Другой процессор, указанный в команде `-target` с помощью параметра `c`, – это центральный процессор Cortex-M. Центральный процессор выполняет только те слои, которые не могут быть выгружены на microNPU;

- `--target-c-mcpu=cortex-m55`: этот параметр указывает целевому процессору выполнять неподдерживаемые слои на microNPU;
- `--runtime=crt`: этот параметр определяет тип среды выполнения. В этом случае мы должны указать среду выполнения C (`crt`), поскольку мы будем запускать приложение на «чисто железной» (*bare-metal*) платформе;
- `--executor=aot`: этот параметр предписывает microTVM построить графический образ модели заблаговременно (AoT), а не во время выполнения. Другими словами, это означает, что приложению не нужно загружать модель во время выполнения программы, поскольку график уже сгенерирован и известен заранее. Это указание позволяет управляющей программе сократить использование SRAM;
- `--executor-aot-interface-api=c`: этот параметр определяет тип интерфейса для управляющей программы AoT. Мы передаем параметр `c`, потому что мы генерируем код C;
- `--pass-config tir.disable_vectorize=1`: этот параметр указывает TVM отключить векторизацию кода, поскольку в C нет собственных векторных типов;
- `--output-format=mlf`: этот параметр определяет формат выходных данных, генерируемых TVM. Поскольку нам нужен формат MLF, мы должны передать `mlf`;
- `cifar10_int8.tflite`: это входная модель для компиляции в C-код.

Через несколько секунд TVM сгенерирует файл формата TAR с именем *module.tar* и выведет на консоль следующий текст:

```
./
./codegen/
./codegen/host/
./codegen/host/include/
./codegen/host/include/tvmgen_default.h
./codegen/host/src/
./codegen/host/src/default_lib2.c
./codegen/host/src/default_lib0.c
./codegen/host/src/default_lib1.c
./metadata.json
./parameters/
./parameters/default.params
./src/
./src/relay.txt
```

Рис. 8.9 ❖ Вывод TVM после генерации кода

Файлы и каталоги, распечатанные TVM на консоли, включены в файл *module.tar*.

4. Распакуйте сгенерированный файл *module.tar* в папку *~/project_npu/src/build*:

```
$ tar -C build -xvf module.tar
```

Теперь у вас файлы и каталоги, перечисленные TVM на рис. 8.9, должны оказаться в папке *~/project_npu/src/build*.

Генерация C-байтовых массивов для входа, выхода и меток

Код C, созданный TVM, не включает входные и выходные тензоры, поскольку они должны быть явно предоставлены пользователем.

В этом примере мы разработаем скрипт на Python для генерации трех C-байтовых массивов, содержащих входные и выходные тензоры и названия меток, необходимые для представления результата классификации в приложении. Входной тензор также будет заполнен действительным изображением для проверки вывода на microNPU.

Скрипт на Python *prepare_assets.py*, содержащий код, разбираемый в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter08/PythonScripts/prepare_assets.py.

Подготовка

Чтобы подготовиться к выполнению этого примера, нам нужно знать, как структурировать скрипт на Python для генерации C-байтового массива.

Скрипт Python должен создать файл заголовка для каждого массива. Сгенерированные файлы сохраняются в папке *~/project_npu/src/include* и называются следующим образом:

- *inputs.h*: входной тензор,
- *outputs.h*: тензор выходных данных,
- *labels.h*: названия меток.

Заголовочные файлы C должны использовать указанные имена файлов, потому что наше приложение будет основано на предварительно созданном примере, который ожидает именно эти файлы.

Чтобы создать C-байтовый массив для входного тензора, скрипт должен принять путь к файлу изображения в качестве аргумента командной строки.

Однако мы не можем напрямую добавить исходное изображение в формате *raw*. Как мы знаем из главы 7 «*Запуск модели TinyML CIFAR-10 на виртуальной платформе ОС Zephyr*», модели требуется входное изображение RGB с разрешением 32×32 с нормализованными и квантизованными значениями

пикселей. Следовательно, изображение необходимо предварительно обработать, прежде чем сохранять его в массиве.

Генерация C-байтовых массивов для выхода и меток проще, чем для входа из-за следующего:

- выходной массив содержит 10 значений типа `int8_t` и может быть инициализирован всеми нулями;
- массив labels содержит 10 строк, содержащих название каждого класса (*airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship* и *truck*).

Как мы упоминали в первом примере этой главы, процессор Cortex-M должен информировать микропроцессор Ethos-U55 о местоположении входных и выходных тензоров. Однако не все области памяти доступны для чтения и записи microNPU.

Поэтому нам нужно обратить внимание на то, где мы храним эти массивы. Следующая таблица дает обзор системы памяти Corstone-300 FVP и областей, к которым можно получить доступ с помощью Arm Ethos-U55:

Таблица 8.2. Система памяти Corstone-300 FVP

Память	**Размер**	**Доступ microNPU**
ITCM	512 Кбайт	нет
DTCM	516 Кбайт	нет
SSE-300 SRAM	2 Мбайт	да
Data SRAM	2 Мбайт	да
DDR	32 Мбайт	да

Как вы можете видеть из таблицы, Ethos-U55 не может получить доступ к памяти **Instruction Tightly Coupled Memory** (**ITCM**) и памяти **Data Tightly Coupled Memory** (**DTCM**), которые являются памятью программ и данных для процессора Cortex-M.

Если мы явно не определим хранилище памяти для входных и выходных массивов, их содержимое может быть помещено в ITCM или DTCM. Например, если мы инициализируем входной массив фиксированными значениями, компилятор может предположить, что это постоянные данные, которые можно поместить в память программ. Чтобы гарантировать, что входные и выходные тензоры находятся в областях памяти, доступных Ethos-U55 microNPU, необходимо явно указать раздел памяти при объявлении массивов. В этом проекте мы будем хранить входные и выходные тензоры в DDR.

Следующий код показывает, как поместить массив `int8_t` с именем `K` в DDR с выравниванием по 16 байтам на Corstone-300 FVP:

```
int8_t K[4] __attribute__((section("ethosu_scratch"), aligned(16)));
```

Имя (`ethosu_scratch`), переданное в спецификацию раздела `__attribute__`, и выравнивание (16) должны соответствовать тому, что указано в скрипте компоновщика, используемом для компиляции нашего приложения. В нашем случае мы будем использовать файл компоновщика, доступный по ссылке: https://github.com/apache/tvm/blob/main/apps/microtvm/ethosu/corstone300.ld.

Как это делается...

Прежде чем разрабатывать скрипт на Python, давайте извлечем входные параметры квантизации из модели CIFAR-10. Вы можете просто использовать веб-приложение Netron (https://netron.app/) для этой цели. В Netron нажмите на кнопку **Open Model...** и прочитайте параметры квантизации, отображаемые для первого уровня сети, как показано на следующем рисунке:

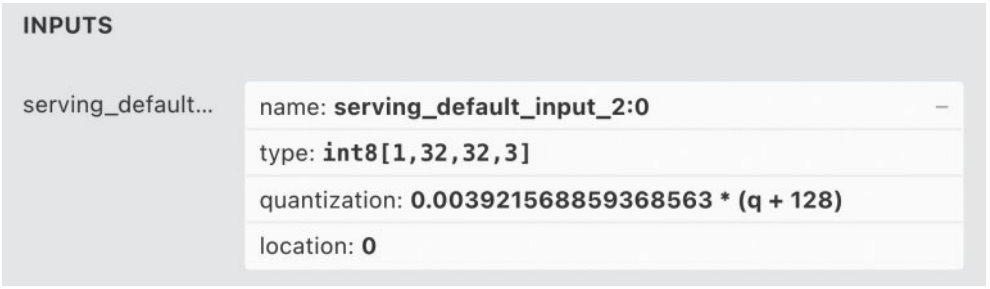


Рис. 8.10 ❖ Параметры Netron для первого слоя

Поле **quantization** содержит формулу преобразования 8-битного квантизованного значения в значение с плавающей запятой, описанное также в главе 3 «*Создание метеостанции с помощью библиотеки TensorFlow Lite for microcontrollers*». Следовательно, масштабирующий параметр равен 0,0039215688..., а нулевая точка равна –128.

Обратите внимание на значение нулевой точки. Этот параметр не равен +128, поскольку 8-битная формула квантизации должна вычитать нулевую точку из целого 8-битного значения.

Теперь откройте предпочитаемый вами редактор Python и создайте новый файл с именем *prepare_assets.py* в папке *~/project_npu/src*.

Откройте *prepare_assets.py* и выполните следующие действия, чтобы сгенерировать C-байтовые массивы для входа, выхода и меток.

1. Объявите две переменные для сохранения входных параметров квантизации модели CIFAR-10:

```
input_quant_offset = -128
input_quant_scale = 0.003921568859368563
```

2. Создайте функцию для генерации содержимого входных и выходных файлов C-заголовков:

```
def gen_c_array(name, size, data):
  str_out = "#include <tvmgen_default.h>\n"
  str_out += f"const unsigned int {name}_len = {size};\n"
  str_out += f'int8_t {name}[] __attribute__
((section("ethosu_scratch"), aligned(16))) = '
```

```
  str_out += "\n{\n"
  str_out += f'{data}'
  str_out += '\n};'
  return str_out
```

Поскольку формат, тип и хранилище данных одинаковы для входных и выходных тензоров, мы можем использовать типовую строку с заменой только следующих отдельных частей:

- имени массива (*name*),
- размера массива (*size*),
- значений для хранения в массиве (*data*).

Как вы можете видеть из приведенного кода, функция ожидает, что {data} будет представлять собой одну строку целочисленных значений, разделенных запятыми.

3. Создайте функцию для преобразования 1D-массива NumPy значений np.int8 в единую строку целочисленных значений, разделенных запятыми:

```
def array_to_str(data):
  NUM_COLS = 12
  val_string = ''
  for i, val in enumerate(data):
    val_string += str(val)
    if (i + 1) < len(data):
      val_string += ','
    if (i + 1) % NUM_COLS == 0:
      val_string += '\n'
  return val_string
```

В приведенном коде переменная NUM_COLS ограничивает количество значений в одной строке. В нашем случае NUM_COLS имеет значение 12, чтобы добавлять символ новой строки после каждых 12 значений.

4. Определите функцию для генерации входного С-байтового массива:

```
def gen_input(img_file):
  img_path = os.path.join(f"{img_file}")
  img_resized = Image.open(img_path).resize((32, 32))
```

В приведенном коде функция gen_input() принимает путь к файлу изображения (image_name) в качестве аргумента. Затем изображение загружается и изменяется до размера 32×32 с помощью библиотеки Python Pillow.

5. Преобразуйте измененное изображение в числовой массив значений с плавающей запятой:

```
  img_data = np.asarray(img_resized).astype("float32")
```

Затем нормализуйте и квантизуйте значения пикселей:

```
  img_data /= 255.0
  img_data /= input_quant_scale
  img_data += input_quant_offset
```

6. Преобразуйте квантизованное изображение в `np.int8` и преобразуйте его в единую строку целочисленных значений:

```
input_data = img_data.astype(np.int8)
input_data = input_data.ravel()
val_string = array_to_str(input_data)
```

   В этом коде мы использовали функцию NumPy `ravel()` для получения непрерывного выровненного массива, поскольку функция `array_to_str()` принимает входной массив только как 1D-объект.

7. Сгенерируйте входной массив C-байт в виде строки и сохраните его как файл заголовка C (*inputs.h*) в папке *include/*:

```
c_code = gen_c_array("input", input_data.size, val_string)
with open("include/inputs.h", 'w') as file:
  file.write(c_code)
```

8. Создайте функцию для генерации файла заголовка C выходного тензора (*outputs.h*) в папке *include/*:

```
def gen_output():
  output_data = np.zeros([10], np.int8)
  val_string = array_to_str(output_data)
  c_code = gen_c_array("output", output_data.size, val_string)
  with open("include/outputs.h", 'w') as file:
    file.write(c_code)
```

9. Создайте функцию для генерации файла заголовка C для меток (*labels.h*) в папке *include/*:

```
def gen_labels():
  val_string = "char* labels[] = "
  val_string += '{"airplane", "automobile", "bird", '
  val_string += '"cat", "deer", "dog", '
  val_string += '"frog", "horse", "ship", "truck"};'
  with open("include/labels.h", 'w') as file:
    file.write(val_string)
```

10. Выполните функции `gen_input()`, `gen_output()` и `gen_labels()`:

```
if __name__ == "__main__":
    gen_input(sys.argv[1])
    gen_output()
    gen_labels()
```

    Как вы можете видеть из приведенного кода, первый аргумент командной строки передается в gen_input(), чтобы указать путь к файлу изображения, предоставленному пользователем.

На данный момент скрипт на Python готов, и нам осталось доработать приложение для запуска модели CIFAR-10 на Ethos-U55 microNPU.

Создание и запуск модели на Arm Ethos-U55

Только этот пример остается в завершение книги. Все инструменты установлены, и модель TFLite преобразована в C-код, так что же нам остается? Нам осталось создать приложение для распознавания изображений с помощью модели CIFAR-10. Как только приложение будет готово, необходимо скомпилировать его и запустить на Corstone-300 FVP.

Хотя кажется, что еще многое предстоит сделать, в этом примере мы модифицируем готовый образец для Ethos-U microNPU, чтобы упростить все остальные технические детали.

В этом примере мы покажем вам, как изменить пример Ethos-U, доступный в TVM, чтобы запустить выполнение CIFAR-10. Затем приложение будет скомпилировано с помощью Makefile и скрипта-компоновщика, предоставленных в готовом образце, и, наконец, выполнено на Corstone-300 FVP.

Скрипт Bash *build_and_run.sh*, содержащий команды, рассмотренные в этом примере, доступен по адресу https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter08/BashScripts/build_and_run.sh.

Подготовка

Готовый пример, рассмотренный в этом разделе, доступен в исходном коде TVM в папке *tvm/apps/microtvm/ethosu*. Образец представляет собой демонстрационный пример для выполнения классификации одного изображения с помощью MobileNet V1 на Ethos-U55. Внутри папки с образцом вы найдете следующее:

- исходный код приложения в подкаталогах `include/` и `src/`,
- скрипты для создания демоверсии для Corstone-300 FVP (*Makefile*, *arm-none-eabigcc.cmake* и *corstone300.ld*),
- скрипты на Python для генерации входных, выходных и заголовочных C-файлов (*convert_image.py* и *convert_labels.py*),
- скрипт для запуска демоверсии на Corstone-300 FVP (*run_demo.sh*).

Из всех этих файлов нам нужен только исходный код приложения и скрипты для создания демоверсии.

Как это делается...

Для создания и запуска приложения CIFAR-10 на Ethos-U55 откройте терминал и выполните следующие действия.

1. Скопируйте исходный код приложения из папки с образцами *~/project_npu/tvm/apps/microtvm/ethosu/* в каталог *~/project_npu/src*:

```
$ cp -r ~/project_npu/tvm/apps/microtvm/ethosu/include ~/project_npu/src/
$ cp -r ~/project_npu/tvm/apps/microtvm/ethosu/src ~/project_npu/src/
```

2. Скопируйте сценарии сборки (*Makefile*, *arm-none-eabi-gcc.cmake* и *corstone300.ld*) из папки с образцами *~/project_npu/tvm/apps/microtvm/ethosu/* в каталог *~/project_npu/src*:

```
$ cp -r ~/project_npu/tvm/apps/microtvm/ethosu/Makefile~/project_npu/src/
$ cp -r ~/project_npu/tvm/apps/microtvm/ethosu/arm-noneeabi-gcc.cmake ~/project_npu/
src/
$ cp -r ~/project_npu/tvm/apps/microtvm/ethosu/corstone300.ld ~/project_npu/src/
```

3. Загрузите изображение *ship.jpg* из репозитория TinyML-Cookbook на GitHub: https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter08/ship.jpg (источник: Pixabay). Сохраните файл в папке *~/project_npu/src*.
4. Просмотрите список каталогов и файлов в папке *~/project_npu/src*:

```
$ sudo apt-get install tree
$ cd ~/project_npu/src/
$ tree
```

Ожидаемый результат в терминале показан на следующем рисунке:

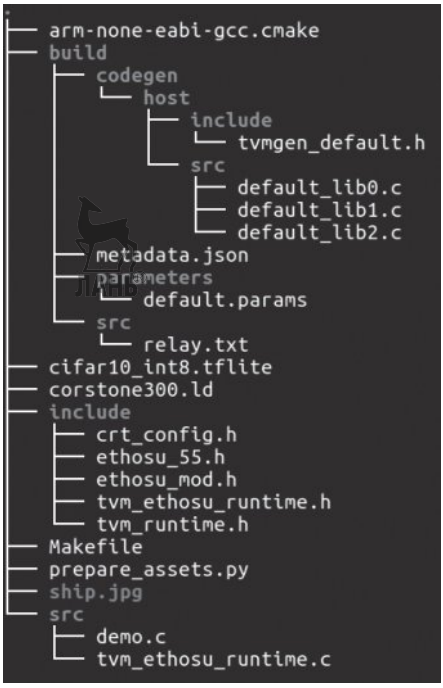


Рис. 8.11 ❖ Ожидаемый результат выполнения команды `tree`

Прежде чем перейти к следующему шагу, проверьте, есть ли у вас все файлы и каталоги, показанные на предыдущем рисунке.

5. Используйте скрипт на Python *prepare_assets.py* для создания заголовочных C-файлов входа, выхода и меток:

```
$ cd ~/project_npu/src
$ python3 prepare_assets.py ship.jpg
```

В этом коде мы передаем файл *ship.jpg* в качестве аргумента командной строки для инициализации входного тензора содержимым изображения корабля.
Скрипт Python сохранит заголовочные файлы C в папке *~/project_npu/src/include*.

6. Откройте файл *demo.c* в каталоге *~/project_npu/src/src* и перейдите к строке 46. Замените имя поля `.input` именем, используемым TVM в структуре `tvmgen_ default_inputs`. Структура `tvmgen_default_inputs` объявлена по умолчанию в файле *~/project_npu/src/build/codegen/host/include/tvmgen_default.h*. Если вы загрузили предварительно обученную модель CIFAR-10 из репозитория TinyML-Cookbook GitHub, имя должно быть `serving_default_input_2_0`. Следовательно, файл *demo.c* должен содержать следующую правку:

```
.serving_default_input_2_0 = input;
```

7. Откройте скрипт *Makefile* в каталоге *~/project_npu/src* с помощью любого текстового редактора. Перейдите к строке 25 и замените путь */opt/arm/ethosu* на *${HOME}/project_npu/sw_libs*:

```
ETHOSU_PATH=${HOME}/project_npu/sw_libs
```

Такое изменение требуется для информирования скрипта *Makefile* о расположении программных библиотек, установленных в разделе «*Установка набора инструментов Arm и стека драйверов Ethos-U*».
Затем сохраните и закройте файл.

8. Создайте приложение с помощью команды `make`:

```
$ make
```

Скрипт *Makefile* сгенерирует двоичный файл с именем *demo* в папке *~/project_npu/src/build*.

9. Запустите демонстрационный исполняемый файл на Corstone-300 FVP:

```
$ FVP_Corstone_SSE-300_Ethos-U55 -C cpu0.CFGDTCMSZ=15 \
-C cpu0.CFGITCMSZ=15 -C mps3_board.uart0.out_file=\"-\" \
-C mps3_board.uart0.shutdown_tag=\"EXITTHESIM\" \
-C mps3_board.visualisation.disable-visualisation=1 \
-C mps3_board.telnetterminal0.start_telnet=0 \
-C mps3_board.telnetterminal1.start_telnet=0 \
-C mps3_board.telnetterminal2.start_telnet=0 \
-C mps3_board.telnetterminal5.start_telnet=0 \
-C ethosu.extra_args="--fast" \
-C ethosu.num_macs=256 ./build/demo
```

В приведенной команде обратите внимание на аргумент `ethosu.num_macs=256`. Этот параметр относится к количеству MAC-адресов в вычислительном ядре Ethos-U55 microNPU и должен соответствовать тому, что указано в TVM при компиляции модели TFLite.

После запуска команды Corstone-300 вы должны увидеть следующий результат в консоли:

```
ethosu_invoke COMMAND_STREAM
handle_command_stream: cmd_stream=0x6100fc60, cms_length 534
QBASE=0x000000006100fc60, QSIZE=2136, base_pointer_offset=0x00000000
BASEP0=0x00000000610104c0
BASEP1=0x0000000060003010
BASEP2=0x0000000060003010
BASEP3=0x0000000060000010
BASEP4=0x0000000060000c10
CMD=0x00000005Interrupt. status=0xffff0022, qread=2136
CMD=0x00000006

CMD=0x0000000c
ethosu_release_driver - Driver 0x20000a18 released
10
The image has been classified as 'ship'
```

Рис. 8.12 ❖ Ожидаемый результат выполнения CIFAR-10

Как сообщается в последней строке результата, изображение правильно классифицировано как корабль.

С этим последним примером приложения на новом Arm Ethos-U55 вы определенно готовы создавать еще более интеллектуальные решения TinyML на микроконтроллерах Cortex-M!

Предметный указатель

A

activations, 32
ANN (artificial neural networks), 31
Arduino IDE, 47
Arduino Nano 33 BLE Sense, 27, 46
Arduino Web Editor, 47
Arm Cortex-M55, 270, 280

C

CIFAR-10, 248, 264
CNNS (сверточные нейронные сети), 32
Colab, 48, 191
convolution (свертка), 34
Corstone-300 FVP, 270, 287

D

DL (deep learning), 30
DNN (deep neural network), 31
DWSC, 249

E

Edge Impulse, 49, 128, 220
EON Tuner, 127, 145
Ethos-U55, 270

F

feature maps (карты параметров), 34
FIFO, 120, 153
fully connected NN, 33

I

I2C, 209
Impulse, 134
ISR. См. *Прерывание*

K

Keras, 20, 189

L

LED (светодиод), 65

M

Mbed OS, 233
Mel-спектрограмма, 137
MFCC, 127, 137
microNPU, 270
ML на микроконтроллерах, 26
ML (machine learning), 25
MobileNet, 190, 249

P

pooling layers, 35
pull-down-резистор, 75
pull-up-резистор, 75
PyAutoGUI, 241

R

RAM (память данных), 43
Raspberry Pi Pico, 46
real-time applications, 41
ReLU, 32
RGB-светодиод, 159
ROM (память программ), 42

S

SRAM, 43

T

TensorFlow, 47
TensorFlow Lite (TFLu), 49, 88, 109, 122, 195, 255
TinyML, 26
  проблемы, 28
  развертывание, 28
  Foundation, 30
TVM (компилятор), 275, 282

U

UART (последовательный порт), 55

W

weights (веса), 31
WorldWeatherOnline, 90

Z

Zephyr ОС, 245

А

Акселерометр, 214
АЦП, 159, 164

Б

Батареи, 82
  подключение, 84
  срок службы, 86
  энергоемкость, 82

В

Веса (weights), 31

Д

Датчик
  микрофон MAX9814, 158
  DHT22 (AM2302), 116
  HTS221, 114
  IMU MPU-6050, 207
Дискретное преобразование Фурье (DFT), 135
Достоверность (accuracy), 92, 255

И

Интерполяция
  bicubic, 199
  bilinear, 199
  nearest-neighbor, 199

К

Камера OV7670, 172
Карты параметров (feature maps), 34
Квантизация (quantizing), 35, 109, 255

М

Макетная плата, 60
Матрица ошибок, 107
Микроконтроллер, 39
Монитор последовательного порта, 56
Мощность, 38

Н

Набор данных
  обучающий, 100
  подготовка, 92
  проверочный, 100
  сбалансированный, 92
  тестовый, 100
Напряжение, 37

О

Обработка изображений, 199
Оперативная память, 43

П

Память данных (RAM), 43
Память программ (ROM), 42
Периферийные устройства, 43
  аналого-цифровые преобразователи, 45
  коммуникационные, 45
  таймеры, 45
  GPIO, 44, 69
Полносвязные сети, 33. См. *Fully connected*
Полнота (recall), 106

Получение аудиосемплов, 129
Прерывание, 79

Р

Распознавание жестов, 221
Распознавание слов (KWS), 127
Режим реального времени, 151
Резистор, 37

С

Свертка (convolution), 34
Сверточные нейронные сети (CNNS), 32
Светодиод (LED), 65

Т

Тензор, 34
Ток, 36
Точность (precision), 106
Трансфертное обучение, 189

Ф

Формат
  QQVGA, 173
  QVGA, 173
  RGB444, 173
  RGB565, 173, 179
  RGB888, 177, 179, 184
  YCbCr422, 173, 184

Э

Энергия, 38
Энергопотребление, 39, 41

Книги издательства «ДМК ПРЕСС»
можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).
Адрес: г. Москва, пр. Андропова, 38, оф. 10;
тел.: **(499) 782-38-89**, электронная почта: **books@alians-kniga.ru**.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **http://www.galaktika-dmk.com/**.

Джан Марко Йодиче

TinyML. Книга рецептов

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Зам. главного редактора *Сенченкова Е. А.*
Перевод *Ревич Ю. В.*
Корректор *Абросимова Л. А.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Гарнитура PT Serif. Печать цифровая.
Усл. печ. л. 24,21. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**