МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ANDROID-ПРИЛОЖЕНИЯ ДЛЯ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ НА ОСНОВЕ ГЛУБОКОГО ОБУЧЕНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Путилова Даниила Ивановича

Научный руководитель
доцент, к. ф.-м. н. _____ Ю. Н. Кондратова

Заведующий кафедрой
к.ф.-м., доцент _____ С. В. Миронов

Саратов 2021

ВВЕДЕНИЕ

Мобильные приложения — это один из наиболее перспективных рынков продвижения продуктов бизнеса. Согласно статистике Statcounter GlobalStats в январе 2020 года трафик проходящий через мобильные устройства превысил трафик PC-приложений, а за период 2009-2020 имеется ярко выраженная тенденция роста трафика, проходящего через мобильные устройства. Таким образом, можно сделать вывод, что популярность мобильных приложений будет только расти.

Наблюдая тенденции развития методов глубокого обучения, нельзя не обратить внимание на то, как нейронные сети проникают во все сферы общества. Повсеместное использование искусственного интеллекта позволяет не только уменьшить затратность проекта, но и улучшить эффективность работы. Очевидно, использование нейронных сетей в мобильной разработке так же стало популярным решением.

Целью работы является разработка мобильного приложения, определяющего съедобность гриба по его изображению.

В соответствии с поставленной целью необходимо решить следующие задачи:

- обучить нейронную сеть для классификации изображений;
- разработать мобильное приложение на базе Android;
- настроить импорт модели нейронной сети в приложение с помощью инструментов API TensorFlow Lite;
- реализовать вывод результатов работы модели в мобильном приложении;
- протестировать работу мобильного приложения.

1 Основное содержание работы

Для решения поставленной в работе задачи может применяться несколько подходов:

— Обучение нейронной сети для распознавания вида гриба изображенного на фотографии, затем определение его съедобности исходя из информации о виде;

— Обучение модели для распознавания всего двух классов (съедобный, несъедобный).

Первый подход позволит создать более многофункциональное и объемлющее приложение, но потребует большого количества изображений для каждого вида грибов, чего нет в имеющемся датасете. Из чего следует, что с использованием имеющихся данных о съедобности и несъедобности каждого вида, датасет разделяется на два класса. Данные для датасета будут получены с помощью сайта $Mushroom\ World$ [1].

Первый шаг: заполнение двух массивов ($classes$, $imgs$ из соответствующих $json$-файлов с помощью $Python$-библиотеки $pandas$. Первый массив заполняется данными о виду гриба изображенном на фото и его съедобности, массив $imgs$ — списком путей к каждому изображению. Затем оба массива объединяются и к выходному массиву применяется функция $load\_images$, которая считывает изображения используя $file\_path$ из массива $info$ как путь к изображениям. Каждое изображение считывается с помощью библиотеки $imageio$, а затем масштабируется к размеру 224x224 с помощью библиотеки $OpenCV2$. $OpenCV$ — библиотека компьютерного зрения и машинного обучения с открытым исходным кодом. В нее входят более 2500 алгоритмов, в которых есть как классические, так и современные алгоритмы для компьютерного зрения и машинного обучения.

В массивы X и y загружаются изображения и метки (съедобность) соответственно. Таким образом, в результате имеются все необходимые для обучения модели данные.

Для оценки эффективности модели, мы предъявляем ей новые размеченные данные (размеченные данные, которые она не видела раньше). По этой причине массивы $features$ и $target$ делятся на несколько выборок: обучающую, тестовую и, реже, валидационную. Это необходимо для того, чтобы после обучения модели на обучающей выборке, была возможность проверить

работу модели на данных, ответы к которым она еще не знает.

Большинство алгоритмов обучения ожидают числовые значения для лучшей эффективности. Для этого кодируем метки ($labels$) как числовые значения с помощью $LabelEncoder$ из пакета $scikit-learn$.

Теперь закодированные метки можно было бы конвертировать в двоичные вектора, где у каждого класса будет свой столбец, так как нейронные сети могут иногда находить определенные порядки и иерархии в числовых значения меток. Однако в данном случае это будет лишним, так как метки уже определяются двоичными значениями из-за того, что классов всего два.

В библиотеке $scikit-learn$ есть функция $train\_test\_split$, которая перемешивает набор данных и разбивает его на две части. Эта функция отбирает в обучающий набор определенную часть данных с соответствующими метками в зависимости от параметра $test\_size$. Оставшаяся часть данных с метками идут во второй набор (тестовый или валидационный, в зависимости от ситуации) [2].

Для эффективного обучения нейронных сетей практически всегда рекомендуется выполнять некоторое масштабирование входных значений. Мы можем нормализовать значения пикселей в диапазоне 0 и 1, разделив каждое значение на максимальные значения 255 [3].

Затем, с помощью функции $reshape$, каждое изображение приводится к формату (1, 224, 224, 3), что для входного слоя модели будет значить, что одно изображение - это один обучающий образец формата (224, 224, 3).

Используем $ImageDataGenerator$ для аугментации фото. В параметрах отключим нормализации и забеливание изображения, так как в нашем случае это негативно отразится на эффективности. Главными параметрами, на которых будет заточено внимания, станут: $zoomrange = 0.4$, то есть приближение и отдаление изображения в пределах 40% от всей картинки, $rotationrange = 90$, поворот на угол до 90 градусов, $horizontal_flip = True$ отражение фото по горизонтали. Эти параметры позволят подготовить модель к человеческому фактору и ослабить необходимость в жестком выборе фото для анализа.

Эффективнейшей во многих фундаментальных задачах, таких как распознавание изображений, распознавание речи, предсказание временных рядов и многих других, является архитектура сверточных нейронных сетей, именно она будет использоваться в решении данной задачи.

Одной из архитектур специально разработанных для использования в мобильных приложениях является $MobileNet$, которая вместо обычного блока сверточной сети, использует $depthwise-convolution$ свертки 3x3 и свертки 1x1 с параметром $stride$ равным 2 для понижения пространственной размерности. Данная модель выполняет одну свертку для каждого цветового канала, а не объединяет все три и выравнивает их. Это имеет эффект фильтрации входных каналов.

Импорт нейронной сети $MobileNet$ производится с помощью $keras.applications$. Для всей работы с моделями нейронными сетей (построением, обучением, вычислением) мы будем использовать $API\ Keras$.

С помощью функции $MobileNet$ архитектура загружается в объявленный слой $mnet$. Параметр $include\_top$ отвечает за загрузку классификатора (полносвязного слоя модели), который будет задан вручную позднее, так как задача требует иной классификации чем та, что использовалась при обучении сети.

Параметр $trainable$ замораживает обучение слоев $MobileNet$, чтобы при обучении сети использовать уже имеющиеся в них карты признаков.

Модель типа $Sequential$ — это простейший вид моделей в $Keras$ для нейронных сетей, которые построены из одного набора слоев соединенных последовательно. В начало модели помещается сверточный блок $MobileNet$. Следующий слой $Dropout$ отключает случайные 30% нейронов каждую итерацию работы модели, чтобы ослабить эффект переобучения. Далее — слой, который преобразует данные двумерной матрицы в вектор, называемый $Flatten$. Он позволяет обрабатывать выходные данные стандартными полносвязными слоями. Выходной полносвязный слой имеет первый параметр как его выходной размер и функцию активации $sigmoid$, которая приводит результирующее значение в интервал от 0 до 1.

Во время обучения модели желательно добавить функцию автоматического ослабления темпа обучения, как только обучение начинает стагнировать. $ReduceLROnPlateau$ будет следить за точностью модели на валидационной выборке ($valaccuracy$) и если в течении двух эпох не будет происходить изменений, то темп обучения будет умножен на $factor$. Это нужно, чтобы алгоритм смог добраться до минимума с помощью более меньших шагов.

С помощью функции fit модель обучается на x_train, y_train как обу-

чающей выборке, предварительно пропуская их через генератор изображений $datagen$. Параметр $batchsize$ отвечает за количество изображений загружаемых в модель за одну итерацию обучения. В параметре $validationdata$ указана валидационная выборка, а функция уменьшения темпа обучения в $callbacks$.

Результатом на тестовой выборке стала точность — 75.15%. Как и ожидалось, небольшой размер датасета оказал сильное негативное влияние на обучение модели, вследствие чего она стала слишком восприимчивой к изображениям из обучающей выборки, но менее гибкой в целом.

Последний шаг — конвертирование модели и её запись в формат $.tflite$ для использования в мобильном приложении с помощью инструментов $Tensorflow\ Lite$.

Для разработки приложения была использована Android Studio — официальная интегрированная среда разработки приложений для $Android$.

Первым этапом проектирования приложения является создание дизайна или графического интерфейса. Для этого подойдет встроенный редактор представлений ($views$). Данный редактор позволяет с помощью кода или графического интерфейса создать xml-разметку приложения.

В первой строке кода указан вид основного элемента $Layout$. Для данного приложения был выбран $Constraint\ Layout$, он позволяет устанавливать расположение элементов с помощью различных ограничений ($constraints$). Параметры $android : layout\_width$ и $android : layout\_height$ указывают на то, что данный элемент заполняет весь экран. Параметр $android : background$ устанавливает фон элемента [4]. Значения атрибутов, начинающиеся с @, такие как $@drawable/grad$ в строке 6, представляют ресурсы со значениями, определяемыми в других файлах.

Для работы приложения понадобится несколько кнопок ($Button$) с помощью которых можно будет выбрать фото из галереи или сделать новое фото с помощью камеры.

Первая кнопка создана с помощью элемента $Button$. Первым параметром является $android : id = "@ + id/bt\_choosepic"$. Знак + в синтаксисе @+id указывает на создание нового свойства id с идентификатором, следующим за косой чертой (/). Для параметра $style$ выбран стиль $"@style/Widget.AppCompat.Button.Colored"$, что окрасит кнопку с базовый цвет приложения.

Для кнопки открывающей камеру использован элемент $ImageButton$.

Это позволит представить кнопку в виде иконки камеры с помощью параметра $app : srcCompat = "@android : drawable/ic\_menu\_camera"$.

На центре экрана размещен элемент $ImageView$, на котором будет показано выбранное изображение. Прямо над ним размещены два элемента $TextView$ отвечающие за вывод вероятности и за оповещающую надпись соответственно.

В данном примере кода, элемент $TextView$ имеет отступ сверху в $16dp$, ограничение снизу другим элементом $TextView$, ограничение справа и слева границей экрана, ограничение сверху.

У каждого приложения существует тема, определяющая оформление стандартных компонентов, которые вы используете. Тема приложения указывается в файле $AndroidManifest.xml$ приложения. Так, определяя ресурсы в файле $styles.xml$, находящемся в папке $res/values$ приложения, определены базовые цвета приложения.

Файл $AndroidManifest.xml$ генерируется средой разработки при создании нового проекта приложения. В файле хранятся многие настройки, вводимые в диалоговом окне $Create\ New\ Project$: имя приложения, имя пакета, имя класса активности и т. д.

В отличие от многих приложений $Java$, приложения $Android$ не содержат метода $main$. Вместо этого в них используются активности, определяемые как субклассы $Activity$ (пакет $android.app$). Приложение может иметь много активностей, одна из которых — первое, что пользователь видит при запуске приложения.

В данном приложении можно обойтись одной активностью, так как все действия производятся в пределах одного окна. Данный файл называется $MainActivity.kt$ и он будет написан на языке $Kotlin$, так как это облегчит взаимодействие с элементами интерфейса.

Файл начинается с определения класса $MainActivity$ приложения как субкласс $AppCompatActivity$ — непрямого субкласса $Activity$, обеспечивающего использование новых средств
$Android$ на современных и старых платформах $Android$ [5].

В приложениях для каждой активности переопределяется метод $onCreate$. Этот метод вызывается исполнительной системой $Android$ при запуске активности — то есть когда ее графический интерфейс готов к отображению, чтобы

пользователь мог взаимодействовать с активностью.

Первой строкой метода будет $setContentView(R.layout.activity_main)$, который указывает какой $.xml$-файл данная активность использует как разметку. Этот процесс называется заполнением ($inflating$) графического интерфейса. Пакет $kotlinx.android.synthetic$ позволит обращаться к элементам интерфейса по id без необходимости «находить» их. Определим обработчики щелчка $setOnClickListener$ к каждой кнопке. Теперь при нажатии на кнопку будет запускаться тело данного метода.

Перед тем как использовать камеру, нажатие кнопки $bt\_takepic$ должно запрашивать разрешение. Для этого определена функция $hasPermission()$, которая будет проверять есть ли разрешение и, если его нет, запрашивать. Условие ($Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.M$) проверяет является ли версия $Android$, на устройстве запускающем приложение, допустимой. Метод $checkSelfPermission$ определяет выдано ли определенное разрешение приложению и возвращает $PERMISSION\_GRANTED$ в успешном случае.

В случае, если разрешение не выдано, вызывается функция $requestPermission$. Сначала условие проверяет является ли версия $Android$, на устройстве запускающем приложение, допустимой. Метод $shouldShowRequestPermissionRationale$ проверяет выдано ли разрешение на использовании камеры и, в противном случае, выдает всплывающее уведомление пользователю с помощью объекта $Toast$ и его метода $makeText$. После вызова метода $requestPermissions$ активируется $callback$ $onRequestPermissionsResult$ в который передаются: результат запроса разрешения на использование камеры и вручную указанный код этого запроса [6].

Метод $onRequestPermissionsResult$ вызывается на каждый запрос $requestPermission$ и позволяет запускать необходимое действие кнопки сразу после получения разрешения. Таким образом, метод сначала определяет запрос какого именно разрешения вызвал данный метод с помощью проверки полученного кода ($requestCode == ...$), затем проверяет выдано ли данное разрешение функцией $hasAllPermissions$ и, наконец, вызывает функцию открытия камеры. Полный код приведен в приложении **??**.

Аналогичный код прописан для кнопки выбора изображения из галереи. Сначала условие ($Build.VERSION.SDK\_INT >= Build.VERSION$

$\_CODES.M$) проверяет является ли версия $Android$, на устройстве запускающем приложение, допустимой. Метод $shouldShowRequestPermissionRationale$ проверяет выдано ли разрешение на использовании камеры. После вызова метода $requestPermissions$ активируется $callback$ $onRequestPermissionsResult$ в который передаются: результат запроса разрешения на использование внешнего хранилища и вручную указанный код этого запроса.

Один из способов для использования камеры в $Android$ это вызов $Intent$. Намерения ($Intent$) — это асинхронные сообщения, позволяющие компонентам приложения запрашивать функциональность от других компонентов $Android$. $Intents$ позволяют взаимодействовать с другими компонентами из тех же приложений, так же как и с компонентами созданные другими приложениями. Намерения могут применяться для трансляции сообщений по системе. Любое приложение способно зарегистрировать широковещательный приемник и отслеживать эти намерения с возможностью на них реагировать. Это позволяет создавать приложения, использующие событийную модель, в основе которой лежат внутренние, системные или сторонние события, передаваемые внешними программами [7].

Таким образом, чтобы открыть камеру создается $Intent$ с действием $ACTION\_IMAGE\_CAPTURE$. Далее, функция $getPhotoFile$ с помощью метода $getExternalFilesDir$ получает ссылку на временный файл изображения, который будет находится на телефоне в директории «Изображения». Ссылка на директорию уже сохранена в классе $Environment$.

Теперь, необходимо получить ссылку на данный временный файл. Это можно сделать с помощью метода $FileProvider.getUriForFile$ из класса $FileProvider$, указав полномочие $com.example.shroomfic.fileprovider$ и файл $photoFile$. Следующий шаг, передача ссылки в намерение с помощью метода $putExtra$ и параметра $MediaStore.EXTRA_OUTPUT$. Обработчик $startActivityForResult$ будет запускаться каждый раз, когда пользователь будет открывать камеру [8].

Аналогичный метод для выбора изображения из галереи будет основываться на намерении $ACTION\_PICK$. Оно имеет параметр $type$, который задает директорию из которой будет выбран файл. В данном случае, это директория $image$. После выбора файла, намерение возвращает его. В завершение, $Intent$ вместе с кодом действия передается в обработчик $startActivityForResult$.

Для того, чтобы внедрить полученную модель нейронной сети в мобильное приложение используются инструменты $Tensorflow\ Lite$.

Чтобы делать прогнозы с помощью модели $tflite$ используется интерпретатор, он применяет статический порядок графов и настраиваемый (менее динамичный) распределитель памяти для обеспечения минимальной нагрузки, инициализации и задержки выполнения.

После создания нового $Java$ класса $Tensorflow\ Classifier$, объявим в нем класс $TensorFlowImageClassifier$. который будет имплементировать $Java$ интерфейс $Classifier$. Объявим будущие константы: $BATCH\_SIZE-$ размер набора для прогнозирования и $PIXEL\_SIZE-$ количество каналов входного изображения [9].

Метод $create$ будет отвечать за инициализацию классификатора. На вход он принимает $AssetManager$, путь к файлу модели $modelPath$ и входной размер изображения $inputSize$. Каждый экземпляр данного класса будет иметь новый интерпретатор, загружая в него модель с помощью метода $loadModelFile$. Список меток для классификации будет состоять только из одной метки «съедобный». Также добавлен параметр $inputSize$.

Одним из простейших способов доступа к модели будет использовать встроенную функцию $TensorFlowLite$, которая загрузит файл модели в папку ml, которая находится на одном и том же уровне, что и res. Для доступа к файлам используется класс $AssetManager$.

Следующий этап, преобразование данных. Изображения полученные из галереи или камеры, будут сначала преобразованы в $Bitmap$, а затем в $ByteBuffer-$тип необходимый для входа модели. Для первого преобразования будет достаточно встроенных инструментов, однако чтобы преобразовать $Bitmap$ в $ByteBuffer$ понадобится собственная функция. Метод $convertBitmapToByteBuffer$ будет приводить $Bitmap$ изображения к форме (1, 224, 224, 3) типа $ByteBuffer$. Сначала выделяем память переменной $bytebuffer$ с помощью функции $allocateDirect$. Размер памяти равен произведению входных размеров, количества каналов и количества изображений. Функция $getPixels$ получает копию данных изображения в формате $int[]$ массива, которая затем преобразуется в $Float$ с помощью операций сдвига (перед этим нормализуясь).

Теперь, можно переопределить стандартный метод $recognizeImage$. По-

лучая на вход $bitmap$ изображение, он преобразует его в $ByteBuffer$. Для хранения вывода модели объявлен $float$ массив $result$. Метод интерпретатора run выполняет вывод модели, если модель принимает только один вход и предоставляет только один выход, поэтому подходит для нашего случая. Массив $byteBuffer$ с входным изображением подается на вход интерпретатора, а массив $result$ на выход. В $float$ переменной $confidence$ сохраняется результат работы модели — вероятность съедобности.

В $MainActivity$ остается подготовки изображений к отправке на модель. Обработчик $onActivityResult$ реагирует на активацию любого из двух намерений. В случае работы с камерой, условие $requestCode == REQUEST\_CODE \&\& BitmapFactory.decodeFile(photoFile.absolutePath) != null$ проверяет код действия и факт существования изображения. Объявляем переменную опций работы с классом $BitmapFactory$ из $android.graphics$. Полученный файл изображения декодируется к типу $Bitmap$ с помощью команды $decodeFile$, принимающей путь файла на вход.

Полученное изображение необходимо перевернуть к обычному виду. Чтобы получить данные о ориентации телефона во время съемки используется специальный класс $android.media.ExifInterface$. Он хранит специальные метаданные, описывающий условия съемки, размеры изображения, авторство, модель устройства и т. п. Метаданные записываются и считываются в соответствии со специальным стандартом $EXIF$ [10].

Имея путь к файлу, интерфейс можно получить мгновенно. С помощью метода $getAttributeInt$ и тэга $TAG\_ORIENTATION$ записываем значение поворота в переменную $orientation$. Изображение вращается с помощью функции $rotateImage$ В зависимости от значения переменной ориентации.

Функция $rotateImage$ принимает на вход $Bitmap$ изображение и значение поворота $angle$. Объявляем пустую матрицу $matrix$ и применяем к ней встроенную функцию $postRotate$, которая по градусу поворота вычисляет необходимую для данного преобразования матрицу. С помощью метода $createBitmap$ получившего на вход исходное изображение, его размеры и матрицу преобразования, возвращаем перевернутое изображение.

После преобразований, функция $ImageClassify$ инициирует классификатор методом $create$ и отправляет $Bitmap$ изображение на вход классификато-

ра методом $recognizeImage$. Результат прогноза форматируется и помещается в $textView$.

При запуске приложения на реальном устройстве перед пользователем открылось окно с двумя кнопками и небольшим текстом. Пользователь может выбрать фото из галереи, нажав на кнопку «выбрать фото», или, может сделать новое фото, нажав на иконку камеры.

После выбора изображения, оно автоматически проходит через модель и на экран выводится вероятность того, что гриб изображенный на фото является съедобным. Скорость работы — чуть меньше секунды. Модель вычислила вероятность 93% съедобности, после анализа скачанного из интернета фото подосиновика.

После выбора изображения, оно автоматически проходит через модель и на экран выводится вероятность того, что гриб изображенный на фото является съедобным. Скорость работы — 0,83 секунды. Модель вычислила вероятность 1.5% съедобности, после анализа скачанного из интернета фото мухомора.

В сравнении с другими доступными приложениями по данной тематике можно выявить как недостатки, так и преимущества. Приложение $ShroomID-MushroomIdentifier$ позволяет определить вид гриба по фото. Оно выводит список видов от наиболее подходящего вида до наименее подходящего. Имеет огромную библиотеку данных о грибах, а также может показать места их обитания. Очевидно, разработанное приложение проигрывает в большинстве пунктов, кроме разве что скорости получения информации о съедобности гриба, если это то, что ищет пользователь. Средняя скорость идентификации гриба — 4,23 секунды, что приблизительно в 5 раз медленнее идентификации разработанным приложением на том же самом фото.

Приложение $Shroomify$, находящееся в топ-3 распознавателей грибов, имеет огромную библиотеку видов с самой различной информацией. Однако, чтобы выяснить съедобность гриба, нужно сначала определить его вид, для чего нужно указать все его внешние признаки, что, как отметили многие пользователи, бывает затруднительно для любителя. В данном случае, разработанное приложение будет гораздо удобнее и эффективнее если пользователю нужно узнать съедобность гриба по фото.

ЗАКЛЮЧЕНИЕ

В ходе работы было реализовано мобильное приложение, которое, после импорта в него модели нейронной сети, может выполнять распознавание съедобности изображенного на фото гриба.

Для разработки приложения была использована официальная интегрированная среда разработки приложений Android Studio и набор инструментов для работы с нейронными сетями на мобильных устройствах. Это позволило быстро создать простое и эффективно выполняющее свою задачу приложение. Обучение нейронной сети проходило на архитектуре MobileNet.

Приложение можно использовать на Android начиная с версии 5.0 Lollipop. В приложение можно загрузить имеющееся изображение из памяти устройства или сфотографировать гриб, используя камеру устройства. При работе выводится результат анализа изображения: съедобный или несъедобный гриб и вероятность данного утверждения. При сравнении с мобильным приложением ShroomID для распознавания грибов выявлено, что разработанное приложение работает быстрее в 5,15 раз.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Mushroom World, сайт с библиотекой о грибах. [Электронный ресурс]. — URL: https://www.mushroom.world/ (Дата обращения 29.04.2021). Загл. с экр. Яз. рус.

2 *Мюллер, А.* Введение в машинное обучение с помощью Python. / А. Мюллер, С. Гвидо. — Москва: Диалектика, 2017. — 480 с.

3 *Geron, A.* Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow / A. Geron. — Sebastopol: O'Reilly Media, 2019. — 346 pp.

4 *Дейтел, П.* Android для программистов / П. Дейтел, Х. Дейтел, Э. Дейтел, М. Моргано. — СПб.: Питер, 2013. — 560 с.

5 Официальный сайт разработчиков приложений для Android [Электронный ресурс]. — URL: https://developer.android.com/index.html (Дата обращения 29.04.2021). Загл. с экр. Яз. рус.

6 *Дарвин, Я. Ф.* Android. Сборник рецептов: задачи и решения для разработчиков приложений, 2-е изд. : Пер. с англ. / Я. Ф. Дарвин. — СПб.: Альфа-книга, 2015. — 768 с.

7 *Марсикано, К.* Android. Программирование для профессионалов.Android. Программирование для профессионалов. / К. Марсикано, Б. Гарднер, Б. Филлипс, К. Стюарт. — СПб.: Питер, 2017. — 704 с.

8 *Скин, Д.* Kotlin. Программирование для профессионалов. / Д. Скин, Д. Гринхол. — СПб.: Питер, 2018. — 464 с.

9 Официальный сайт TensorFlow Lite [Электронный ресурс]. — URL: https://www.tensorflow.org/lite/guide. (Дата обращения 02.05.2021). Загл. с экр. Яз. англ.

10 Форум Android-разработчиков. [Электронный ресурс]. — URL: https://android-developers.googleblog.com/2016/12/introducing-the-exifinterface-support-library.html (Дата обращения 05.05.2021). Загл. с экр. Яз. англ.