ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. А.Н.Тихонова

**Домашнее задание**

**часть 2**

по направлению 09.03.01 Информатика и вычислительная техника студента образовательной программы бакалавриата

«Информатика и вычислительная техника»

Дисциплина:

«[Проектирование систем на кристалле](https://edu.hse.ru/course/view.php?id=188925)»

Преподаватель:

Американов Александр Александрович

Выполнил:

Студенты группы: БИВ215

Борисова Софья Алексеевна

Розанцева Валерия Викторовна

Дата сдачи: 11.06.2024

Москва 2024

**Оглавление**

[1 Постановка задачи 4](#_gjdgxs)

[2 Детекция животных 6](#_30j0zll)

[2.1 Задание 6](#_1fob9te)

[2.2 Нейронная сеть 6](#_z7eh4dk1rnhj)

[2.3 Обучение данных 12](#_4jz9f4reibe)

[3 Приложение TFLite Object Detection 14](#_ayknx0h0w2fh)

[3.1 Скачивание проекта 14](#_sv0j3p6yw01)

[3.2 Импорт проекта 14](#_bzigz8d26j4s)

[3.3 Подключение смартфона 17](#_25tk86nql5vt)

[3.4 Запуск приложения 21](#_ys212sv3nse3)

[3.5 Модели 27](#_hg14keur01cj)

[4 Детекция Пикачу 29](#_j8jvi78sb8jp)

[4.1 Создание собственной пользовательской модели 29](#_v9ghfm2767im)

[4.2 Создание набора данных и обработка изображений 30](#_rqrp9lgzjs0f)

[4.3 Обучение модели 31](#_5xzyjw6p6ub)

[4.4 Экспорт модели 33](#_ycd2nq51awis)

[4.5 Запуск 34](#_8gu5nywqu4wa)

[4.6 Ошибки 35](#_jjv4ubuqjrst)

[5 SSD MobileNet с TensorFlow 37](#_hu0myqx9zrfp)

[5.1 Описание 37](#_nz28h2rvvoxh)

[5.2 Методы архитектуры 37](#_bvf7fxhdalpr)

[5.3 Нейронная сеть и обучение 39](#_vlkihnvb4hyu)

[5.4 Запуск приложения 40](#_bchvk8f8oiqu)

[3 Выводы 43](#_tyjcwt)

[4 Список литературы](#_3dy6vkm) 44

# 1 Постановка задачи

На данный момент есть реализация MobileNet на PC (Python), Raspberry Pi (Linux + Python + Tensorflow), Raspberry Pi (Linux + C/C++), OpenVINO (Verilog, аппаратный уровень) для сравнения с этими реализациями требуется сделать реализацию под Android. Необходима работоспособная версия на телефоне, которая снимает и классифицирует объекты при съемке на камеру телефона.

Задачу можно разбить на следующие пункты:

1. Запустить существующую реализацию (номер 1):   
   <https://github.com/juandes/pikachu-detection>  
   <https://towardsdatascience.com/detecting-pikachu-on-android-using-tensorflow-object-detection-15464c7a60cd>  
   Она основана на Tensorflow и Android Studio. По результатам сделать подробный гайд в картинках как это было сделано, с детальным описанием каждого шага.
2. Запустить существующую реализацию (номер 2)  
   <https://github.com/androidthings/sample-tensorflow-imageclassifier>
3. На базе пункта 2) сделать детекцию людей, животных и машин (у нас есть модельки от Keras - можно сконвертировать в TF)
4. На базе пункта 1) Реализовать MobileNet + SSD модель и обучить её для детекции людей, животных и машин.
5. Протестировать скорость работы всех реализаций на телефоне

Представление решения

1. Пункт 1 и 2: Детальный гайд с подробным описанием и картинками с описанием процесса запуска. Демонстрация работы на телефоне на базе Android.
2. Пункт 3 и 4: Весь код необходимый для запуска в репозитории ИППМ
3. Пункт 5: Документ XLS с измерениями скорости работы.

Необходимо выбрать один из предложенных проектов, реализовать его.

Защита проекта состоит из 2-х частей: презентация длительностью 5–10 минут с демонстрацией работы проекта; отчет 15–20 страниц.

# 

# 2 Детекция животных

## 2.1 Задание

На базе пункта 2) сделать детекцию людей, животных и машин (у нас есть модельки от Keras - можно сконвертировать в TF).

## 2.2 Нейронная сеть

*Распознавание изображений (классификация)*

CIFAR-10 – это набор данных изображений, состоящий из 60 000 цветных изображений размером 32x32, разделенных на 10 классов. Классы включают в себя:

Самолет, Автомобиль, Птица, Кот, Олень, Собака, Лягушка, Лошадь, Корабль, Грузовик

Обучение нейронных сетей на CIFAR-10 включает в себя следующие шаги:

1. Предварительная обработка: Изображения масштабируются и нормализуются, чтобы улучшить производительность обучения.

2. Разбиение набора данных: Набор данных делится на тренировочный и тестовый наборы. Тестовый набор обычно содержит 10 000 изображений (20% от общего числа).

Создание нейронной сети для CIFAR-10 для классификации изображений CIFAR-10 с архитектурой CNN:

• Сверточные слои: Извлекают локальные признаки из изображений.

• Объединяющие слои

• Полносвязные слои: Объединяют признаки из сверточных слоев и делают прогнозы.

• Выходные слои: Обычно используют softmax-функцию для классификации изображений.

Нейронная сеть обучается с использованием алгоритма обратного распространения и оптимизатора. Целевая функция обычно представляет собой энтропию перекрестной энтропии, которая измеряет разницу между прогнозами сети и истинными метками.

Среда разработки – Google Collab.

Импорт библиотек (Рисунок 1).

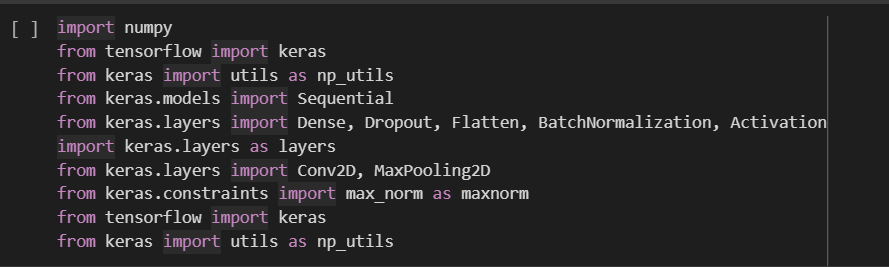


Рисунок 1. Библиотеки

Импорт набора данных (Рисунок 2).



Рисунок 2. Набор данных

Нормализация данных заключается в переводе в формат с плавающей запятой и преобразовании в оттенки серого (Рисунок 3).

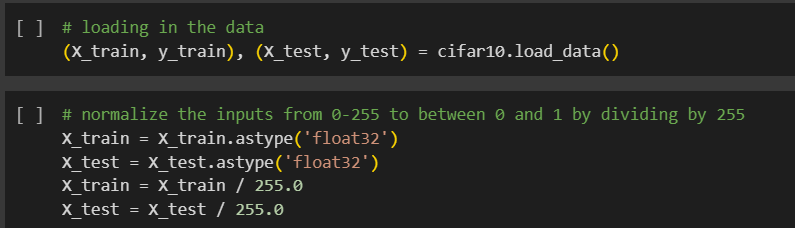


Рисунок 3. Загрузка данных и нормализация

Преобразование меток в формат one hot encoding (Рисунок 4). Изображение либо принадлежит к классу, либо нет – бинарный код. Cluss\_num – количество классов в наборе данных.

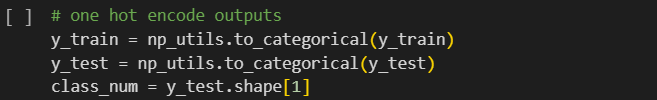


Рисунок 4. Подготовка данных для сети

Последовательная модель Sequential (Рисунок 5).

Слои модели:

Сверточный слой 1:

Первый слой модели - это сверточный слой. Он будет принимать входные данные и пропускать их через сверточные фильтры. При реализации в Keras, необходимо указать количество каналов (фильтров), которое нужно (32), размер фильтра (3 x 3), форму входа (при создании первого слоя), функцию активации и отступы. Наиболее распространенной функцией активации является relu, а отступы определяются через padding = 'same', то есть, размер изображения не меняется.

Далее необходимо создать исключающий слой для предотвращения переобучения, который случайным образом устраняет соединения между слоями (0,2 означает, что он отбрасывает 20% существующих соединений). Также выполняется пакетная нормализация, которая нормализует входные данные, поступающие в следующий слой, гарантируя, что сеть всегда создает функции активации с тем же распределением, которое нужно.

Слой 1: Сверточный слой

• Параметры:

\* 32: Количество фильтров (сверточных ядер)

\* (3, 3): Размер фильтров

\* input\_shape=x\_train.shape[1:]: Форма входных данных (высота, ширина, каналы)

\* padding='same': Режим заполнения, который сохраняет размерность входных данных

Слой 2: Функция активации ReLU

Слой 3: Dropout

• Параметры: 0.2: Частота отбрасывания (20% нейронов будут случайным образом отключены). Создан для предотвращения переобучения.

Слой 4: BatchNormalization()

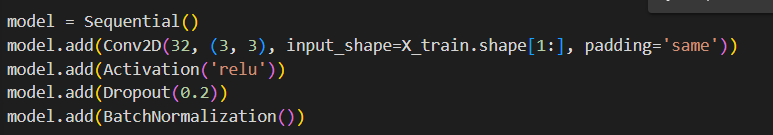


Рисунок 5. Создание модели, 4 слоя

Слой 5: Сверточный слой

• Параметры:

\* 64: Количество фильтров

\* (3, 3): Размер фильтров

\* padding='same': Режим заполнения

Слой 6: Функция активации ReLU

Слой 7: Объединяющий слой

• Параметры:

\* pool\_size=(2, 2): Размер пула

Слой 8: Dropout

• Параметры: 0.2:

Слой 9: BatchNormalization() (Рисунок 6).

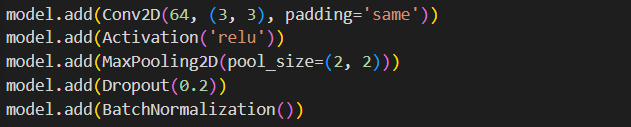


Рисунок 6. Создание модели, 5 слоев

Далее повтор слоев (Рисунок 7).

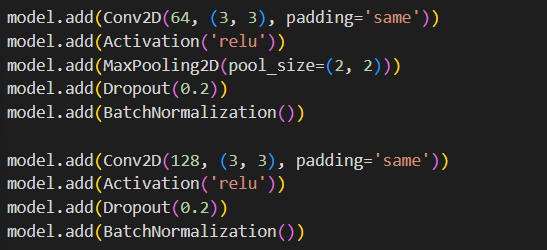


Рисунок 7. Создание модели, повтор слоев

Слой 19: Полносвязный слой (Рисунок 8).

• Параметры:

\* 256: Количество нейронов

\* kernel\_constraint=maxnorm(3): Ограничение нормы для весов

Далее повтор слоев.

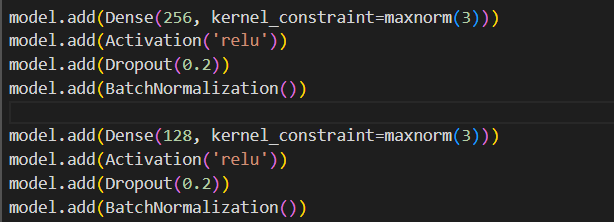


Рисунок 8. Создание модели, первый полносвязный слой

Слой 27: Слой сглаживания (Рисунок 9).

Обеспечивает совместимость между сверточными слоями и полносвязными слоями в нейронных сетях.

Слой 28: Dropout



Рисунок 9. Создание модели, Flatten

Так как в модели 10 классов, то на выходе должен быть вектор (None, 10).

Слой 37: Полносвязный слой

• Параметры:

\* class\_num: Количество классов

Функция активации softmax (Рисунок 10).

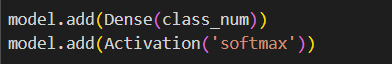


Рисунок 10. Создание модели, полносвязный слой и функция активации

Использование 25 эпох и алгоритма Адама в качестве оптимизатора (Рисунок 11).



Рисунок 11. Эпохи, оптимизатор и компиляция

Распечатка модели:

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 32, 32, 32) 896

activation (Activation) (None, 32, 32, 32) 0

dropout (Dropout) (None, 32, 32, 32) 0

batch\_normalization (Batch (None, 32, 32, 32) 128

Normalization)

conv2d\_1 (Conv2D) (None, 32, 32, 64) 18496

activation\_1 (Activation) (None, 32, 32, 64) 0

max\_pooling2d (MaxPooling2 (None, 16, 16, 64) 0

D)

dropout\_1 (Dropout) (None, 16, 16, 64) 0

batch\_normalization\_1 (Bat (None, 16, 16, 64) 256

chNormalization)

conv2d\_2 (Conv2D) (None, 16, 16, 64) 36928

activation\_2 (Activation) (None, 16, 16, 64) 0

max\_pooling2d\_1 (MaxPoolin (None, 8, 8, 64) 0

g2D)

dropout\_2 (Dropout) (None, 8, 8, 64) 0

batch\_normalization\_2 (Bat (None, 8, 8, 64) 256

chNormalization)

conv2d\_3 (Conv2D) (None, 8, 8, 128) 73856

activation\_3 (Activation) (None, 8, 8, 128) 0

dropout\_3 (Dropout) (None, 8, 8, 128) 0

batch\_normalization\_3 (Bat (None, 8, 8, 128) 512

chNormalization)

flatten (Flatten) (None, 8192) 0

dropout\_4 (Dropout) (None, 8192) 0

dense (Dense) (None, 256) 2097408

activation\_4 (Activation) (None, 256) 0

dropout\_5 (Dropout) (None, 256) 0

batch\_normalization\_4 (Bat (None, 256) 1024

chNormalization)

dense\_1 (Dense) (None, 128) 32896

activation\_5 (Activation) (None, 128) 0

dropout\_6 (Dropout) (None, 128) 0

batch\_normalization\_5 (Bat (None, 128) 512

chNormalization)

dense\_2 (Dense) (None, 10) 1290

activation\_6 (Activation) (None, 10) 0

=================================================================

Total params: 2264458 (8.64 MB)

Trainable params: 2263114 (8.63 MB)

Non-trainable params: 1344 (5.25 KB)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

None

## 2.3 Обучение данных

Вызвать функцию .fit() и передать выбранные параметры (Рисунок 12).

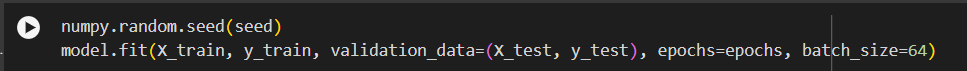


Рисунок 12. Обучение модели

Просмотр эпох (Рисунок 13). Точность с каждой эпохой увеличивалась, поэтому переобучения не было.

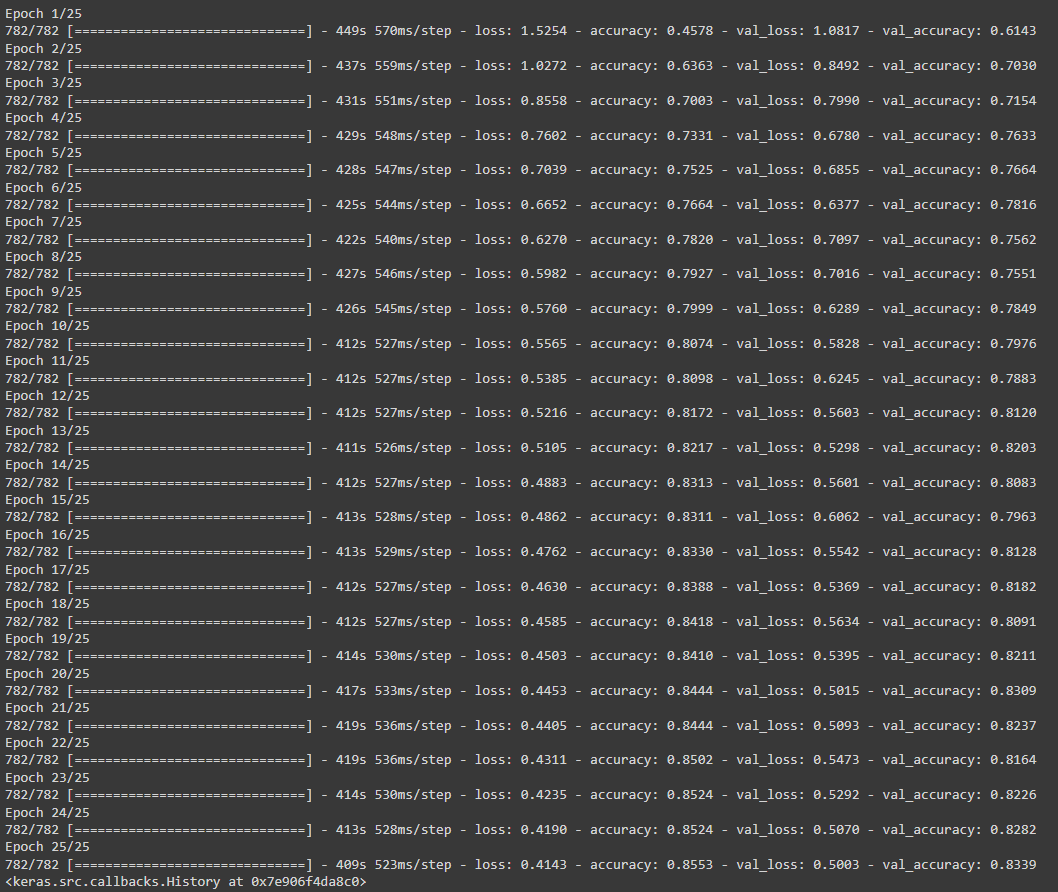


Рисунок 13. Эпохи

Для оценки модели используется model.evaluate() (Рисунок 14).

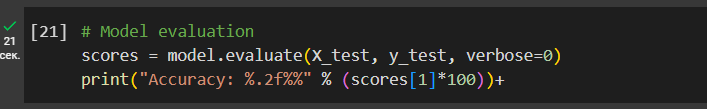


Рисунок 14. Оценка модели

Результат: Accuracy: 83.39%.

Сохранение модели (Рисунок 15). Код для создания модели и сама обученная модель содержатся в [6] в нашем GitHub.

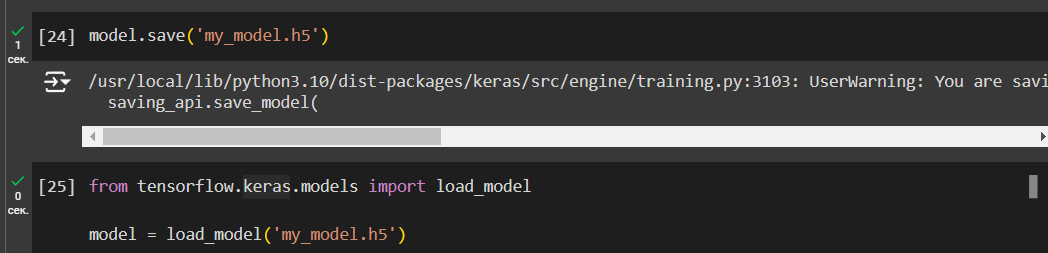


Рисунок 15. Сохранение модели

# 3 Приложение TFLite Object Detection

### 3.1 Скачивание проекта

В github [2] содержатся коды для запуска приложения обнаружения объектов, обученные на данных COCO.

На сайте [3] указаны системные требования:

* Android Studio версии 2021.1.1 (Bumblebee) или выше.
* Android SDK версии 31 или выше
* Устройство Android с минимальной версией ОС SDK 24 (Android 7.0 – Nougat) с включенным режимом разработчика.

В Git Bash перейти в директорию, в которую сохранится будущий проект, с помощью команды cd. Клонировать репозиторий git (рисунок 27) командой git clone.

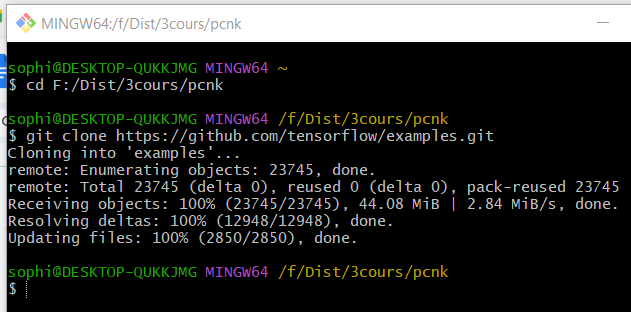


Рисунок 26. Клонирование репозитория

### 3.2 **Импорт** проекта

Запустить Android Studio (использована версия 2023.3.1.19). Нажать Open (рисунок 28) и выбрать папку F:\Dist\3cours\pcnk\examples\lite\examples\object\_detection\android.

# 

Рисунок 26. Запуск Android Studio

Провести синхронизацию Gradle, нажав Try again.. во всплывающем окне.

Ошибка (рисунок 29) возникает из-за несовместимости версий между плагином Android Gradle и плагином Kotlin Gradle.



Рисунок 29. Ошибка при запуске проекта

Необходимо проверить версии плагинов. Открыть Project Structure (рисунок 30).



Рисунок 29. Project Structure

Установить Android Gradle Plugin 7.1.2 и Gradle 7.6 (рисунок 30).

# 

Рисунок 29. Android Gradle Plugin и Gradle версии

Проверить наличие Android SDK (рисунок 30).

# 

Рисунок 29. Android SDK путь

Также при сборке проекта нужно иметь установленный JDK (был использован jdk-21\_windows-x64\_bin.exe) и создать системную переменную JAVA\_HOME с указанием пути до папки с jdk (рисунок 30).

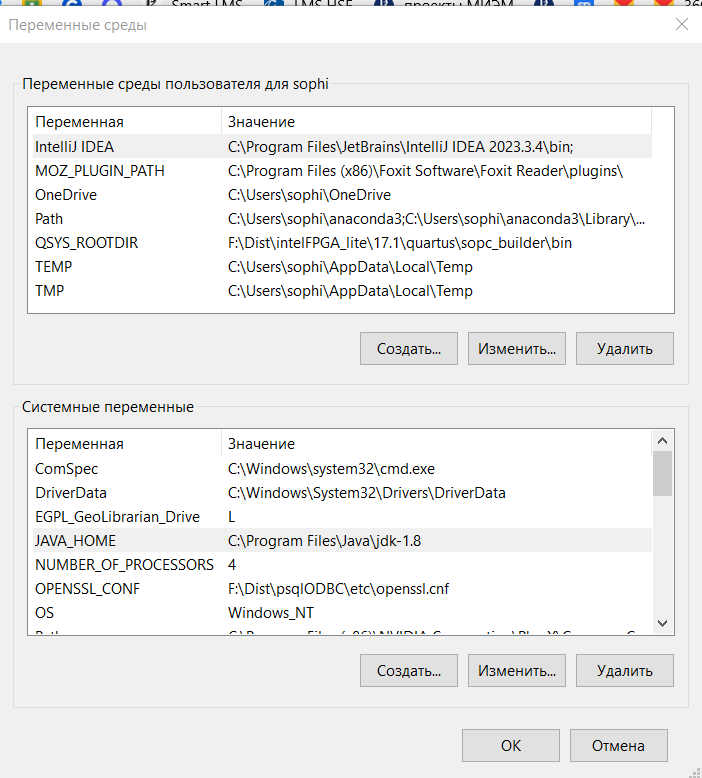


Рисунок 29. Создание JAVA\_HOME переменной

Провести повторную синхронизацию.

### 3.3 **Подключение** смартфона

На смартфоне зайти в раздел настроек “О телефоне” и нажать на версию Android несколько раз, до появления уведомления о включении режима разработчика (Рисунок ).

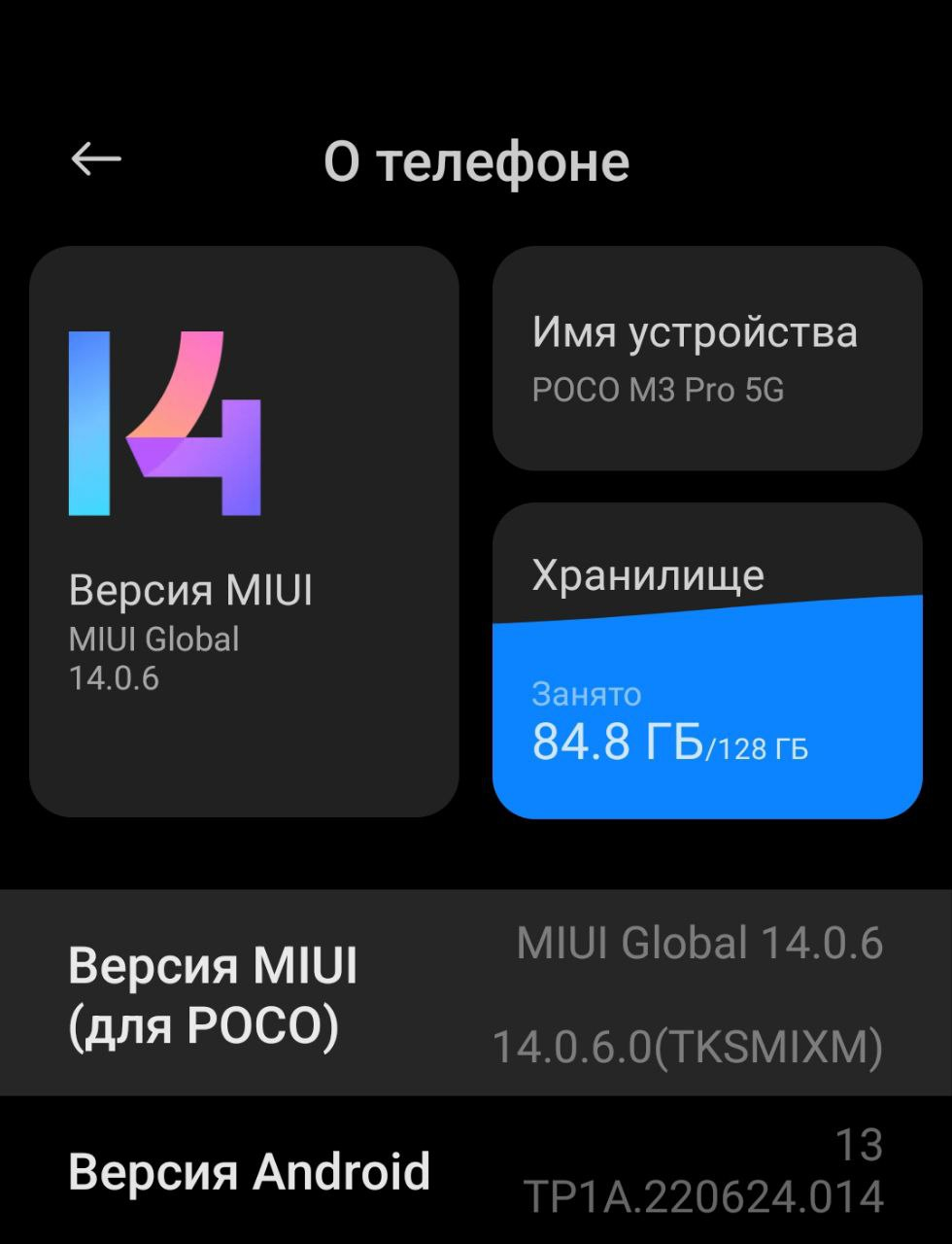


Рисунок 29. Вызов режима разработчика

С помощью поиска в настройках найти режим разработчика (Рисунок 34).

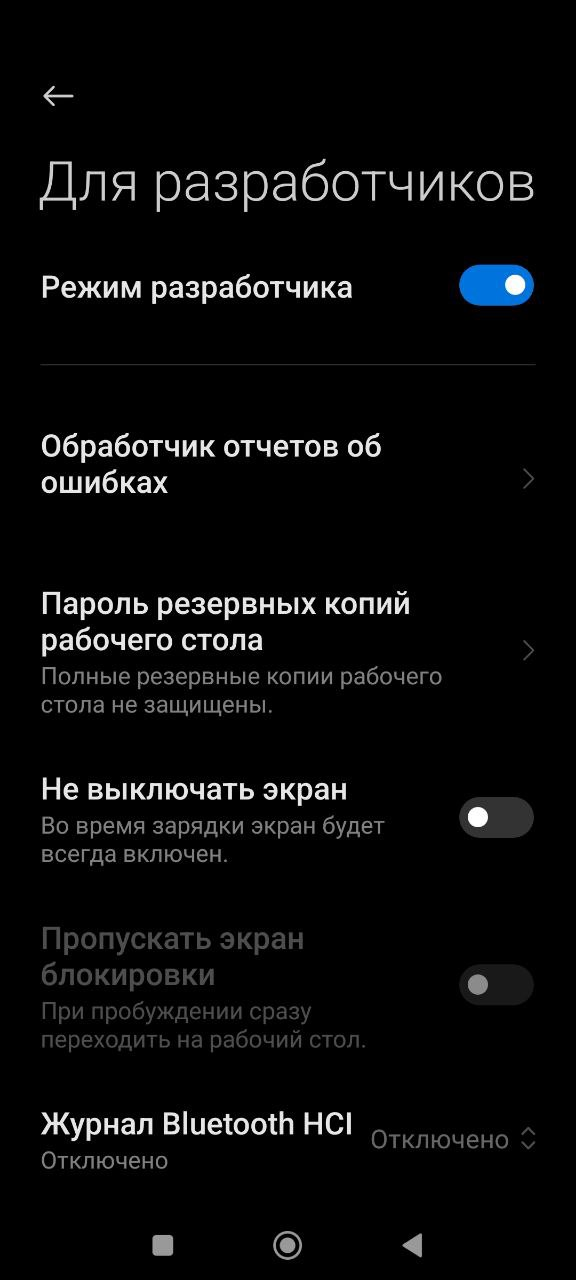


Рисунок 29. Режима разработчика

Разрешить отладку по USB и установку приложений через USB (Рисунок 34).

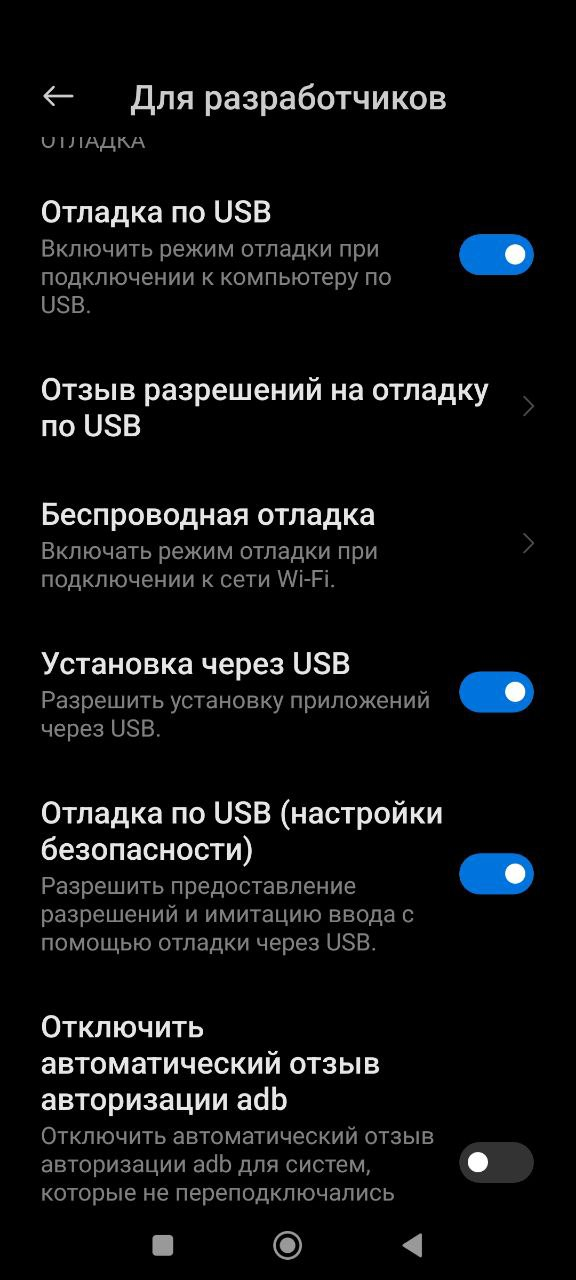


Рисунок 29. Отладка по USB

Добавить телефон в Device Manager, нажав на + (рисунок 34).

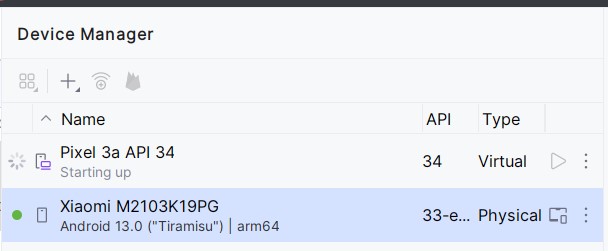


Рисунок 35. Device Manager

Нажать run в Android Studio (рисунок 34).

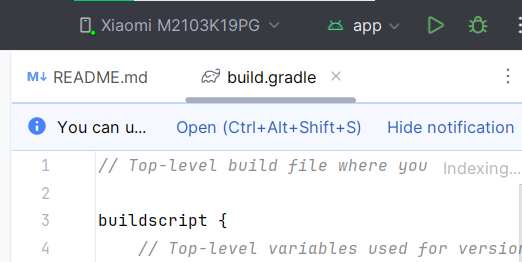


Рисунок 35. Запуск в Android Studio

Сообщения в Android Studio (рисунок 34).

# 

Рисунок 35. Сообщение после запуска

Во всплывающем уведомлении будет вопрос о разрешении на загрузку (рисунок 34).

# 

Рисунок 35. Разрешение на отладку и загрузку

### 3.4 Запуск приложения

Приложение TFLite Object Detection и его иконка (рисунок 34).

# 

Рисунок 35. Иконка приложения

Обнаружение собаки (рисунок 34), ноутбука и клавиатуры (рисунок 34).

# 

Рисунок 35. Обнаружение собаки

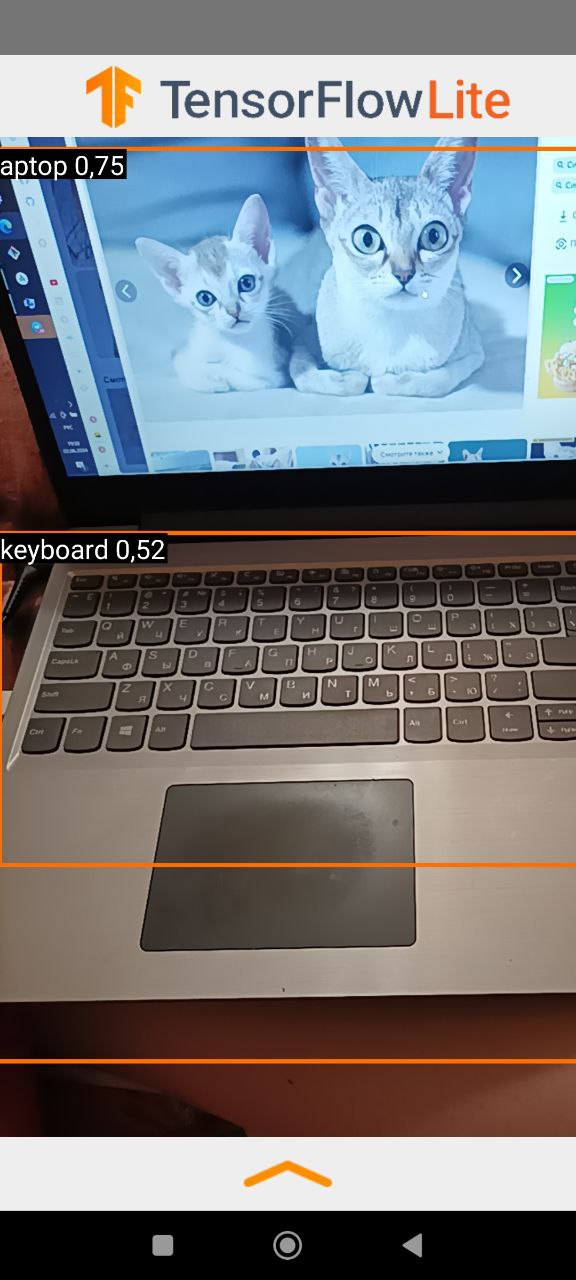


Рисунок 35. Обнаружение ноутбука и клавиатуры

В приложении можно менять предварительно обученную модель и параметры.

Модели:

EfficientDet-Lite0 – для набора проверочных данных COCO 2017 средняя средняя точность составляет 25,69%.

EfficientDet-Lite1 – точность 30,55%.

EfficientDet-Lite2 – точность 33,97%.

MobileNetV1-SSD – точность 21%.

Параметры:

* Inference Time: Время вывода в миллисекундах. Меньшие значения указывают на более быстрые выводы, но могут снизить точность.
* Threshold: пороговое значение для обнаруженных объектов. Только объекты выше порогового значения будут отображаться.
* Max Results: Максимальное количество обнаруживаемых объектов.
* Number of Threads: Количество потоков, используемых для обнаружения объектов. Большее количество потоков может ускорить вывод, но может увеличить использование памяти.
* Delegate: Устройство, используемое для выполнения выводов. Возможные значения:

CPU: Использование центрального процессора устройства;

GPU: Использование графического процессора устройства;

NNAPI: Использование ускорителя нейронных сетей Android (если он доступен).

Модель EfficientDet-Lite0 с Threshold=0.5, обнаружен ноутбук и один из котов (рисунок 34).

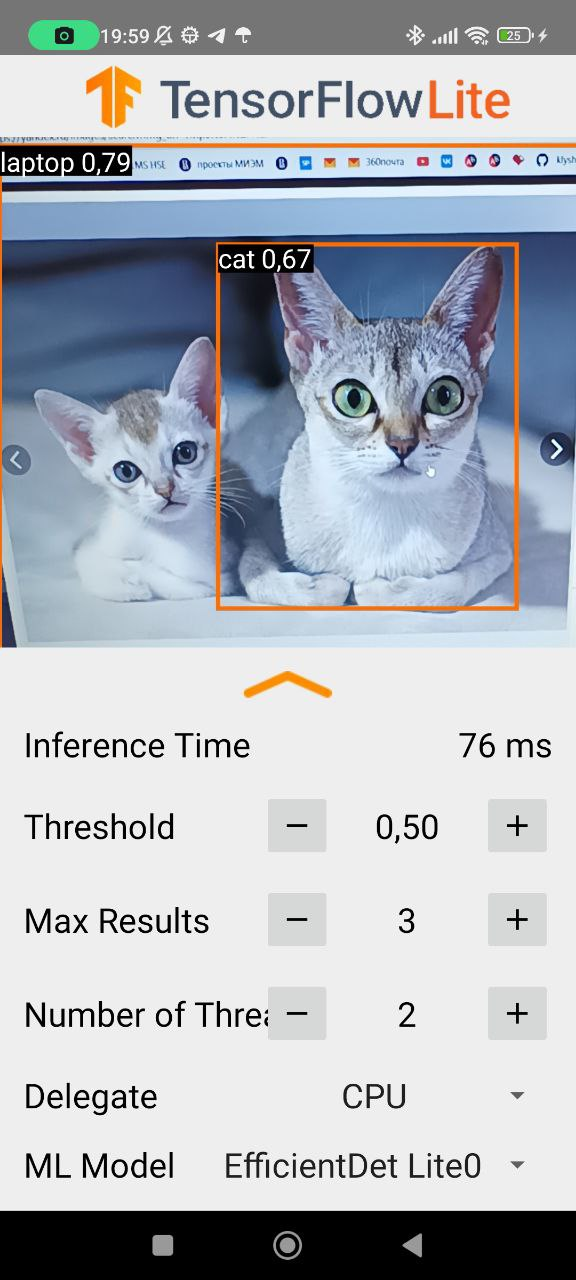


Рисунок 35. EfficientDet-Lite0 с Threshold=0.5

Модель EfficientDet-Lite0 с Threshold=0.4, два кота определен как один (рисунок 34).

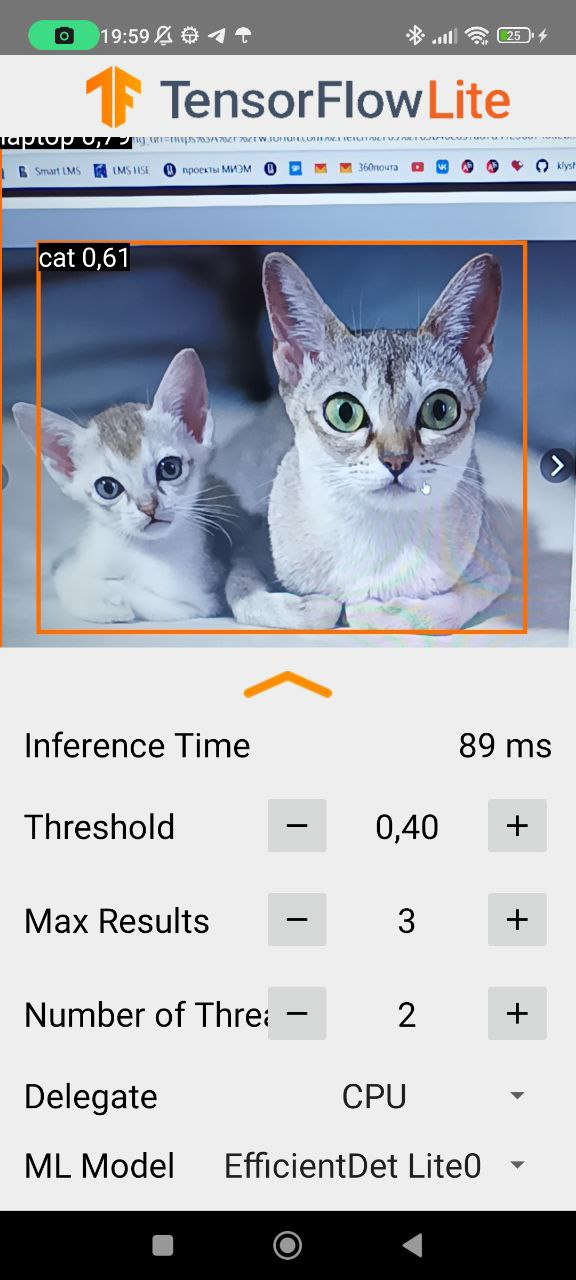


Рисунок 35. EfficientDet-Lite0 с Threshold=0.4

Модель EfficientDet-Lite1 с Threshold=0.5, обнаружены ноутбук, два кота (рисунок 34).

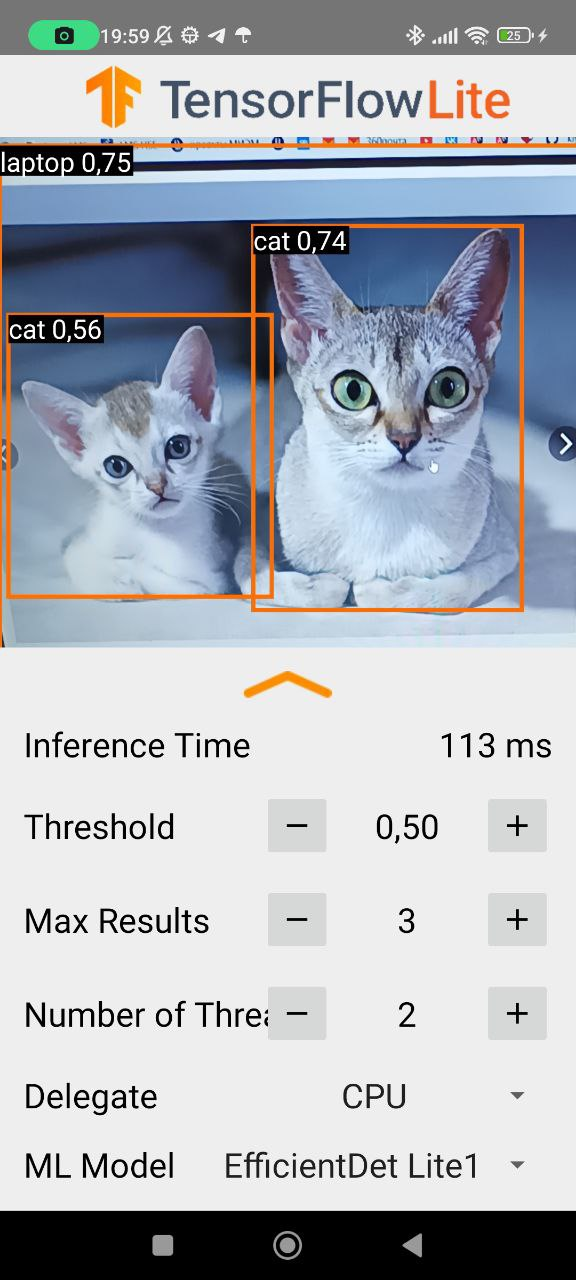


Рисунок 35. EfficientDet-Lite1 с Threshold=0.5

Модель EfficientDet-Lite2 с Threshold=0.4, обнаружены ноутбук, два кота (рисунок 34).

# 

Рисунок 35. EfficientDet-Lite2 с Threshold=0.4

Модель MobileNetV1 с Threshold=0.5, обнаружены ноутбук, два кота (рисунок 34).

# 

Рисунок 35. MobileNetV1 с Threshold=0.5

Таким образом, три объекта распознали модели EfficientDet-Lite1, EfficientDet-Lite2, MobileNetV1.

### 3.5 **Модели**

*examples/tensorflow\_examples/models/dcgan/dcgan.py*

Этот код реализует обучение генеративно-состязательной сети (GAN) на наборе данных MNIST с использованием TensorFlow. Он включает в себя следующие этапы: определение командных флагов для настройки гиперпараметров, создание и масштабирование обучающего набора данных, построение архитектур генератора и дискриминатора, а также обучение этих моделей. Генератор создаёт изображения, похожие на рукописные цифры, а дискриминатор учится различать настоящие изображения из набора данных MNIST и поддельные, созданные генератором. Класс `Dcgan` управляет процессом обучения, включая расчёт потерь, обновление градиентов и сохранение контрольных точек модели. В итоге, GAN обучается генерировать реалистичные изображения цифр.

*examples/tensorflow\_examples/models/densenet/densenet.py*

Этот код реализует архитектуру DenseNet с использованием TensorFlow и Keras. DenseNet — это тип свёрточной нейронной сети, в которой каждый слой подключается ко всем предыдущим слоям, улучшая передачу градиента и повторное использование параметров. Код включает функции для расчёта количества слоёв в блоках на основе разных режимов (глубина модели, список слоёв или целое число), а также классы для свёрточных, переходных и плотных блоков. Основной класс `DenseNet` собирает полную архитектуру сети, состоящую из начального свёрточного слоя, нескольких плотных и переходных блоков, и завершается слоем глобального усреднённого пуллинга и плотным слоем для классификации. Эта архитектура используется для улучшения задач классификации изображений.

*examples/tensorflow\_examples/models/nmt\_with\_attention/nmt.py*

Этот код реализует модель нейронной машинной трансляции с вниманием (NMT with Attention) с использованием TensorFlow. Он содержит классы для энкодера, внимания Бахданау и декодера. Энкодер преобразует входные последовательности во внутреннее представление, а декодер генерирует выходную последовательность, используя контекстное внимание к входу. Модель обучается на данных параллельных последовательностей для задачи машинного перевода.

*examples/tensorflow\_examples/models/pix2pix/pix2pix.py*

Этот код реализует модель Pix2pix для преобразования изображений с использованием TensorFlow. Он включает загрузку и подготовку данных, создание наборов данных, определение архитектур генератора (U-Net) и дискриминатора (PatchGAN), а также обучение модели. Основные шаги: обработка изображений (resize, normalize, jitter), создание обучающих и тестовых наборов данных, определение функций downsample и upsample для свёрточных операций, инициализация и обучение Pix2pix, где реализованы вычисление потерь и обновление весов моделей. Завершающий этап — обучение модели с возможностью сохранения контрольных точек.

# 4 Детекция Пикачу

### 4.1 **Созда**ние собственной пользовательской модели

Приложение для обнаружения Пикачу, основанное на Tensorflow [4].

Необходимо установить TensorFlow (Рисунок 16).

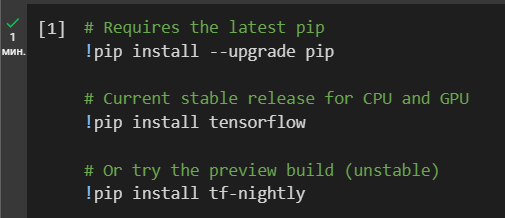


Рисунок 16. Установка TensorFlow

Затем необходимо клонировать репозиторий, в который включен API обнаружения объектов (Рисунок 17). После клонирования репозитория следует перейти в каталог «research».

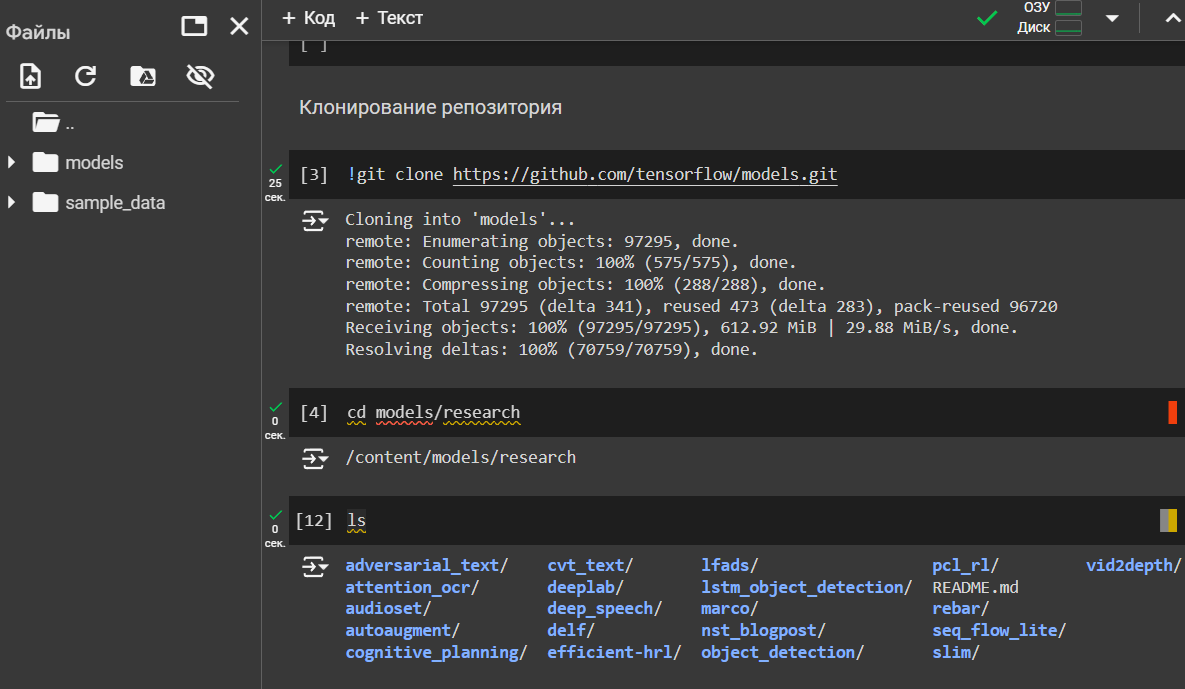


Рисунок 17. Клонирование репозитория

Далее следует скомпилировать библиотеки Protobuf (Рисунок 18). Установка Protobuf по ссылке [5].

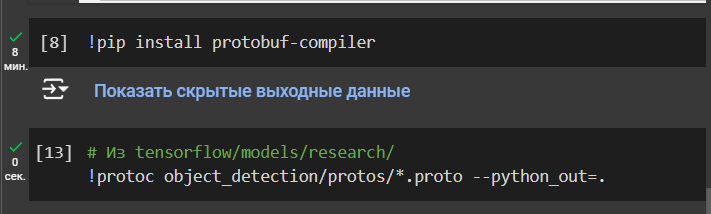


Рисунок 18. Установка Protobuf

Далее нужно добавить библиотеки в файл PYTHONPATH (Рисунок 19).



Рисунок 19. Файл PYTHONPATH

### 4.2 **Соз**дание набора данных и обработка изображений

Было загружено 230 изображений Пикачу среднего размера в каталог images (Рисунок 20).

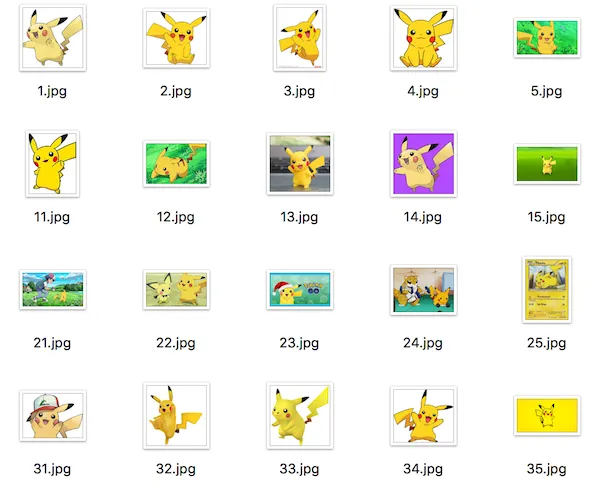


Рисунок 20. Набор данных

Маркировка изображений в RectLabel для создание рамок вокруг объекта, ограничивающую область для обучения.

*Наборы данных для обучения и тестирования*

Разделение набора данных на наборы обучающих и тестовых данных. Внутри того же каталога, где находятся изображения, был создан каталог с именами «train» и «test» и добавлено около 70% изображений и соответствующий им XML в каталог обучения, а оставшиеся 30% — в каталог теста.

*Создание TFRECORD*

После разделения набора данных изображения и их XML были преобразованы в формат, читаемый TensorFlow.

Этот формат называется «tfrecord», и для его создания необходимо выполнить два шага. Во-первых, из обоих наборов XML (обучение и тестирование) были преобразованы в два файла CSV с использованием модифицированной версии кода, xml\_to\_csv.py представленного на GitHub ([https:// github.com/datitran/raccoon\_dataset](https://github.com/datitran/raccoon_dataset)). Затем, используя файлы CSV, с помощью сценария создается набор данных tfrecord generate\_tfrecord.py (также из файла datitran). Перед запуском скрипта необходимо указать в функции класс объектов class\_text\_to\_int.

*Создание карты меток*

Требуется карта «labels», указывающая метки и их индексы (Рисунок 21).

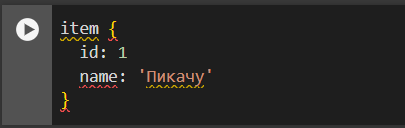


Рисунок 20. Карта меток

### 4.3 Обучение модели

Весь процесс обучения управляется файлом конфигурации, известным как «pipeline». Указанный конвейер разделен на пять основных структур, которые отвечают за определение модели, параметры процесса обучения и оценки, а также входные данные набора данных обучения и оценки. Скелет (Рисунок 21).

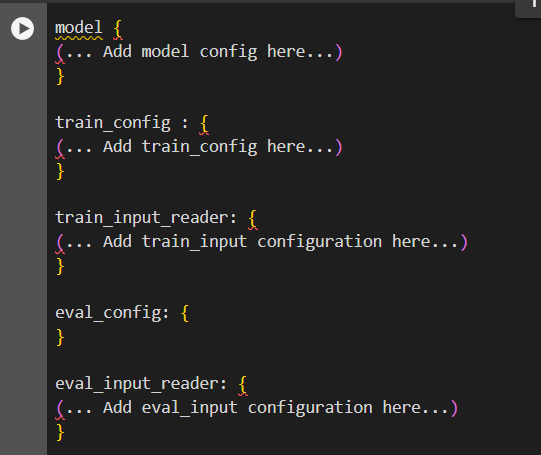


Рисунок 21. Скелет файла pipeline

Использовалась модель ssd\_mobilenet\_v1\_coco\_11\_06\_2017 (http://download.tensorflow.org/models/object\_detection/ssd\_mobilenet\_v1\_coco\_11\_06\_2017.tar.gz ).

Для обучения использовался файл конфигурации ssd\_mobilenet\_v1\_pets.config в качестве отправной точки. Были внесены изменения, которые, касались переменной, num\_classes поскольку в данном проекте всего один класс, num\_steps - чтобы остановить обучение раньше, fine\_tune\_checkpoint - указать местоположение загруженной модели, а также переменные и, input\_path - чтобы указать на набор обучающих и тестовых данных, а также карта меток - label\_map\_pathtrain\_input\_readereval\_input\_reader.

Когда pipeline был написан, файл был добавлен в каталог «training». Затем было запущено обучение (Рисунок 22).

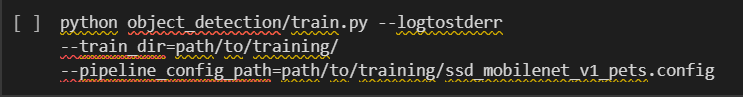


Рисунок 22. Запуск обучения

Каждый раз, когда в ходе обучения создается новая контрольная точка, инструмент оценки выполняет прогнозы, используя изображения, доступные в заданном каталоге. Оценка (Рисунок 23).

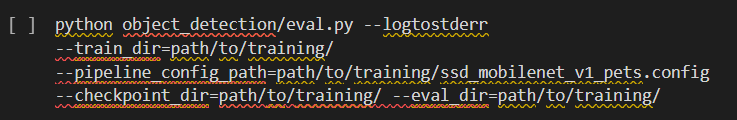


Рисунок 23. Запуск оценки

### 4.4 **Эк**спорт модели

После завершения обучения был произведен экспорт модели для ее дальнейшего использования. Библиотека предоставляет сценарий с именем export\_inference\_graph.py (<https://github.com/19VaNeK99/coco/blob/main/coco_v3/object_detection/export_inference_graph.py>), необходимый для выполнения этого шага.

Прежде чем экспортировать надо убедиться, что в каталоге обучения имеются следующие файлы:

model.ckpt-${CHECKPOINT\_NUMBER}.data-00000-of-00001,

model.ckpt-${CHECKPOINT\_NUMBER}.index

model.ckpt-${CHECKPOINT\_NUMBER}.meta

Далее следует выбрать нужную контрольную точку и выполнить сохранение модели (Рисунок 24). Результатом будет файл с именем frozen\_inference\_graph.pb.

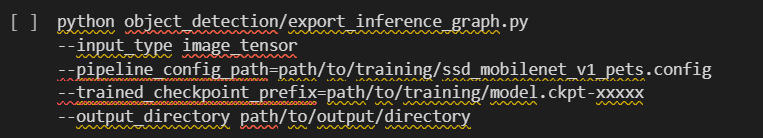


Рисунок 24. Сохранение модели

### 4.5 Запуск

Клонирование репозитория TensorFlow (Рисунок 25).

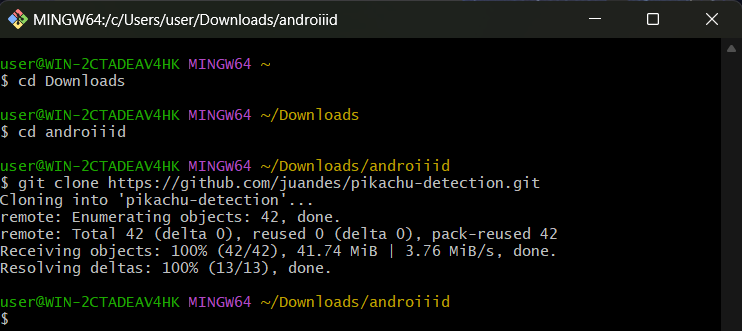


Рисунок 25. Клонирование репозитория

Далее было запущено приложение Android Studio (Рисунок 26).

Далее был импортирован новый проект при использовании каталога из клонированного репозитория TensorFlow, под названием «Android» (Рисунок 27).

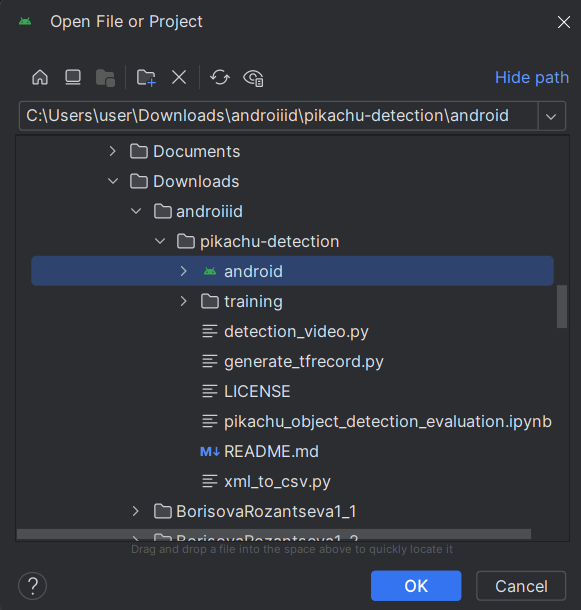


Рисунок 26. Модуль android

### 4.6 **Ошибки**

Запуск приложения не удался.

Были раскомментированы строки в модуле использования NDK.



> org/gradle/initialization/BuildCompletionListener" [android - HOW TO SOLVE THIS ERROR FLUTTER A.S. PROJECT:- " A problem occurred evaluating project ':app'. > org/gradle/initialization/BuildCompletionListener" - Stack Overflow](https://stackoverflow.com/questions/76199124/how-to-solve-this-error-flutter-a-s-project-a-problem-occurred-evaluating-p)

apply plugin: 'com.android.application' [Android Studio - Failed to apply plugin [id 'com.android.application'] - Stack Overflow](https://stackoverflow.com/questions/37086806/android-studio-failed-to-apply-plugin-id-com-android-application)

NDK [Install and configure the NDK and CMake | Android Studio | Android Developers](https://developer.android.com/studio/projects/install-ndk)

[examples/lite/examples/digit\_classifier/android/app/download\_models.gradle at master · tensorflow/examples · GitHub](https://github.com/tensorflow/examples/blob/master/lite/examples/digit_classifier/android/app/download_models.gradle) lite/examples/digit\_classifier/android/app/download\_models.gradle

## 5 SSD MobileNet с TensorFlow

### 5.1 **Описание**

SSD (детектор одиночных снимков) – это алгоритм, используемый для обнаружения объектов в реальном времени. Он состоит из двух частей:

1. извлечение карты объектов;
2. применение сверточного фильтра для обнаружения объекта.

В первой части он извлекает функции, представленные на изображении (строит карту функций изображения). Карта функций — это выходные данные CNN, которые извлекают некоторую важную часть изображения, например, руки, глаза. Во второй части он классифицирует объекты, присутствующие на изображении, и построит вокруг них ограничивающие рамки.

Был рассмотрен код:

https://github.com/kamlesh11/Object-Detection-Using-SSD-MobileNet.git

Этот репозиторий содержит реализацию Mobilenet-SSD300 и его развертывание на ПК, а также на Android. Были использованы весовые коэффициенты модели зоопарка, обученные на наборе данных обнаружения coco объектов. Далее была осуществлена развёртка этой модели с помощью opencv.

### 5.2 **Методы** архитектуры

Ниже приведено описание всех методов, используемых в архитектуре MobileNet.

Глубокие свертки

Вычислительные затраты на стандартные свертки с ядрами размера Dk\*Dk, каждая из которых вычисляет весь ввод размера Df\*Df\*M, тогда вычислительные затраты будут



в то время как в глубокой свертке для каждого входа существуют разные ядра, т.е. для M входных каналов есть M ядер, что означает, что существует M Dk \* Dk ядер. Теперь вычислительные затраты будут



Поточечная свертка

Поточечная свертка выполняет свертку 1x1. Она аналогична стандартной свертке, за исключением того, что размер ядра равен 1x1.

Стоимость вычислений составит



мобильный использует еще два метода, которые

Множитель ширины

Второй метод снижения вычислительных затрат — это «множитель ширины», который представляет собой гиперпараметр, находящийся в диапазоне [0, 1], обозначенном здесь как 𝛼α. 𝛼α пропорционально уменьшает количество входных и выходных каналов:



Множитель разрешения

Третий метод снижения вычислительных затрат — это «множитель разрешения», который представляет собой гиперпараметр, находящийся в диапазоне [0, 1], обозначенном здесь как 𝜌ρ. 𝜌ρ уменьшает размер входной карты объектов:



Объединение множителей ширины и разрешения приводит к вычислительным затратам:



Здесь каждый блок MobileNet будет иметь структуру (Рисунок ).



Рисунок 36. Структура MobileNet

### 5.3 Нейронная сеть и обучение

В качестве обучающих данных был выбран набор данных COCO, который состоит из 80 категорий и 300 тысяч изображений, из которых более 200 тысяч были помечены.

MobileNetV1 (Рисунок 37).



Рисунок 37. MobileNetV1

Object Detection Using SSD+Mobilenet (Рисунок 38-39).

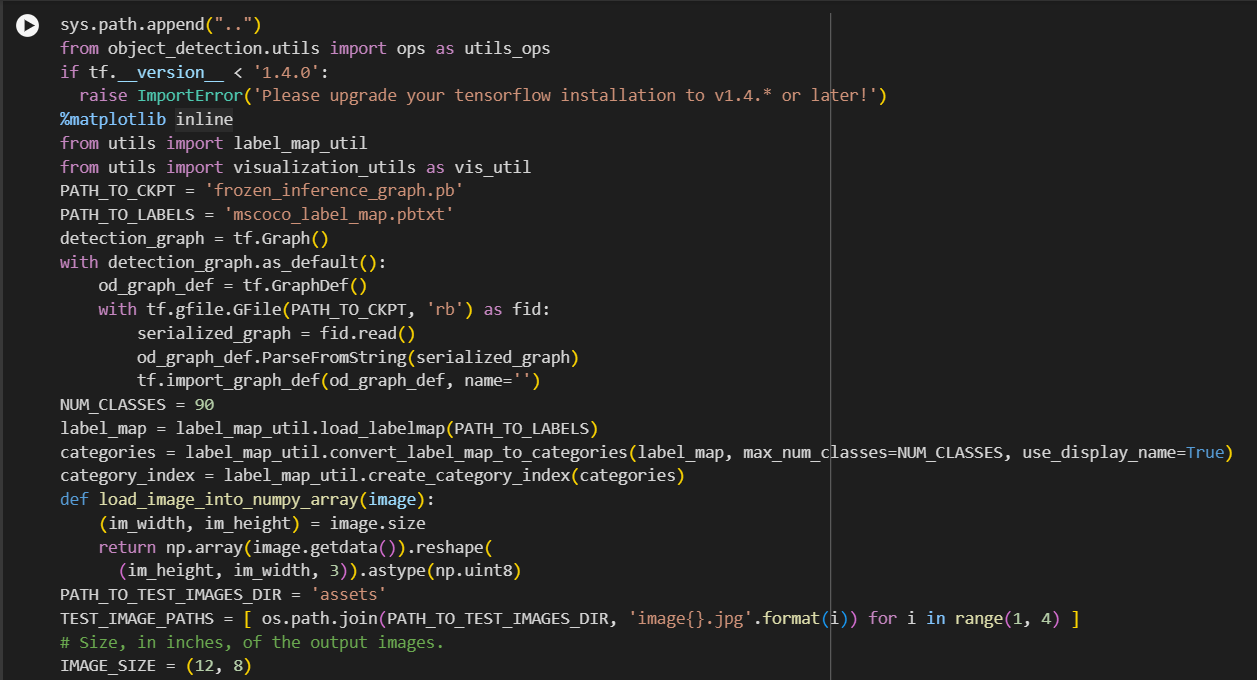


Рисунок 38. Object Detection Using SSD+Mobilenet, 1

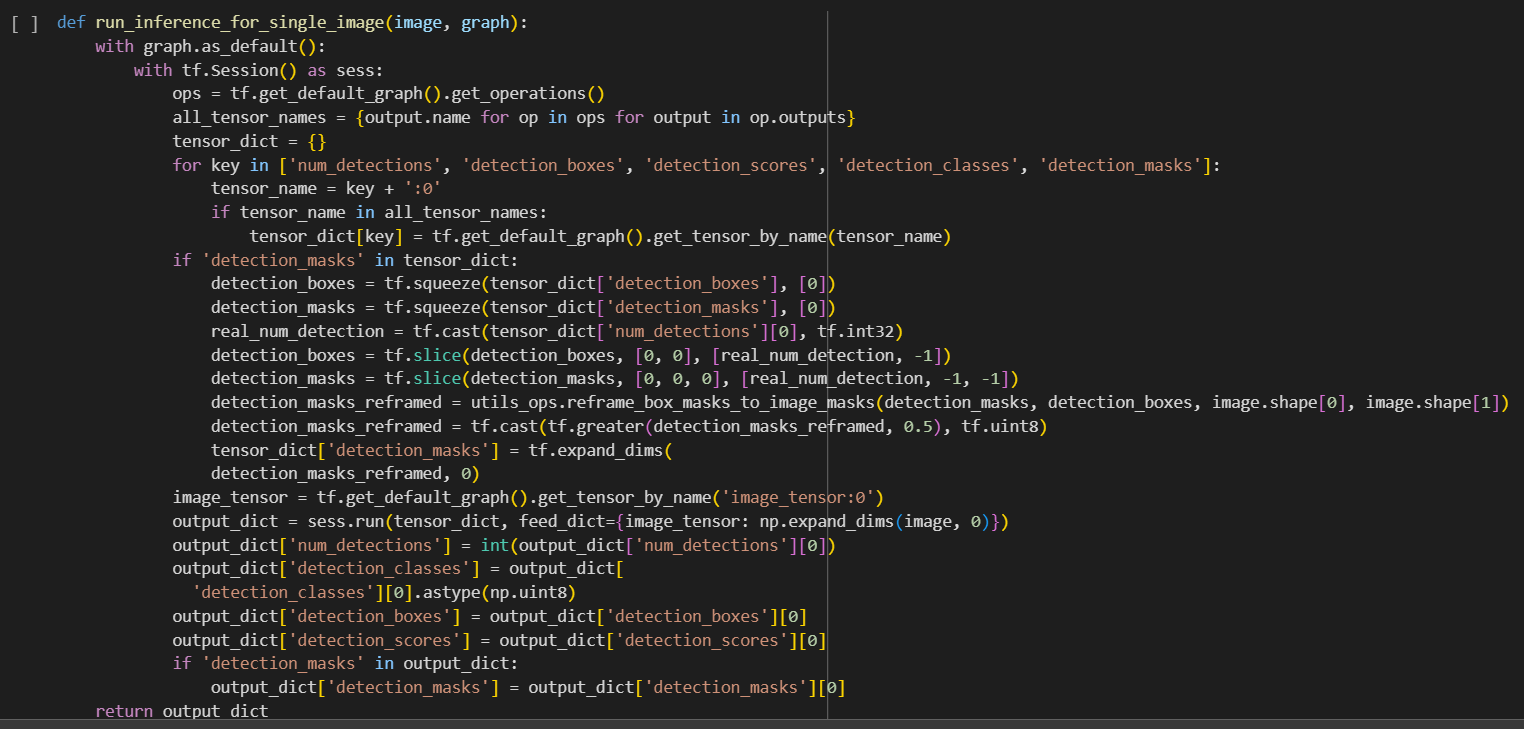


Рисунок 38. Object Detection Using SSD+Mobilenet, 2

### 5.4 **Запуск** приложения

Запуск в Android Studio (Рисунок 37).

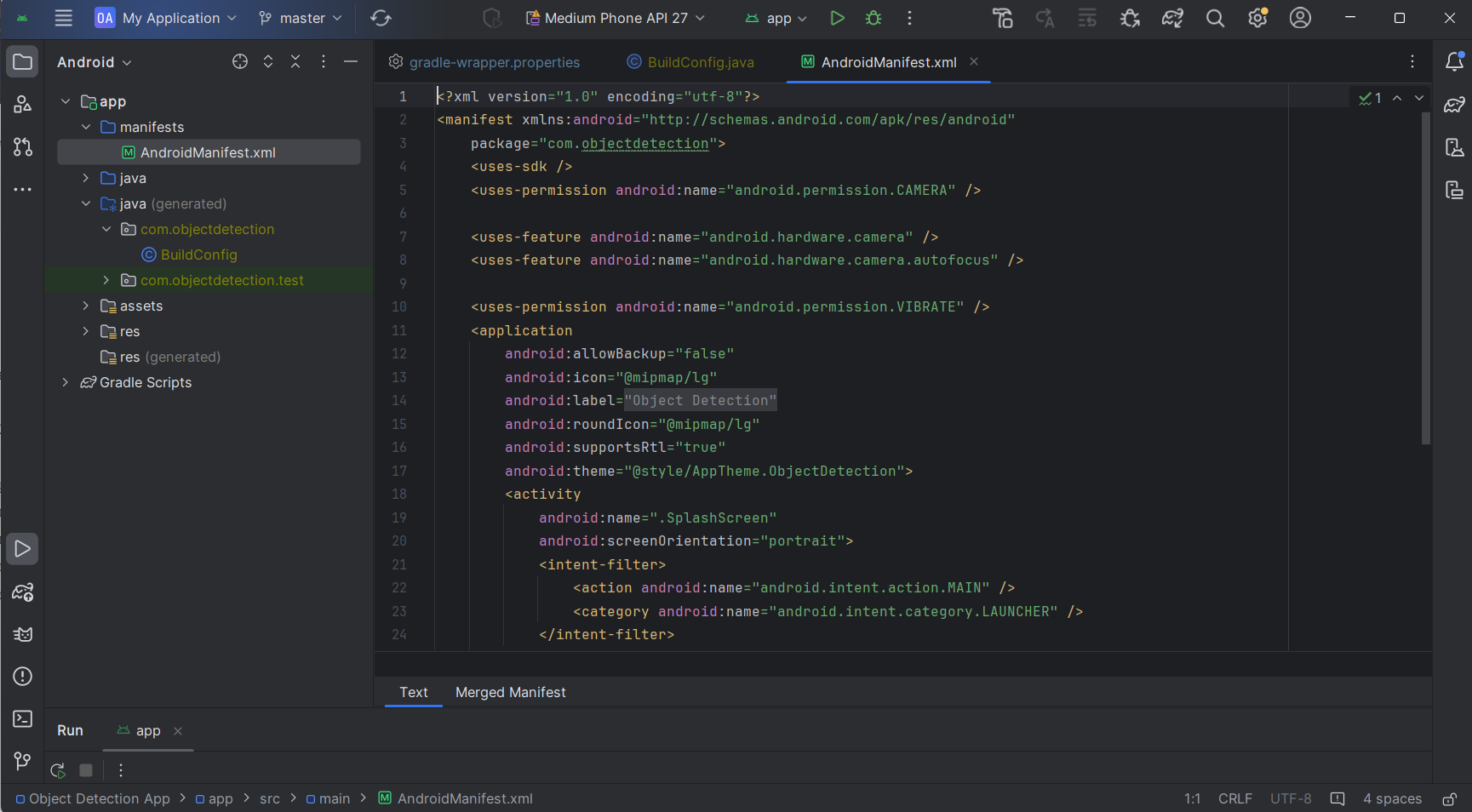


Рисунок 37. Android Studio

Обнаружение объектов (Рисунок 38-40).

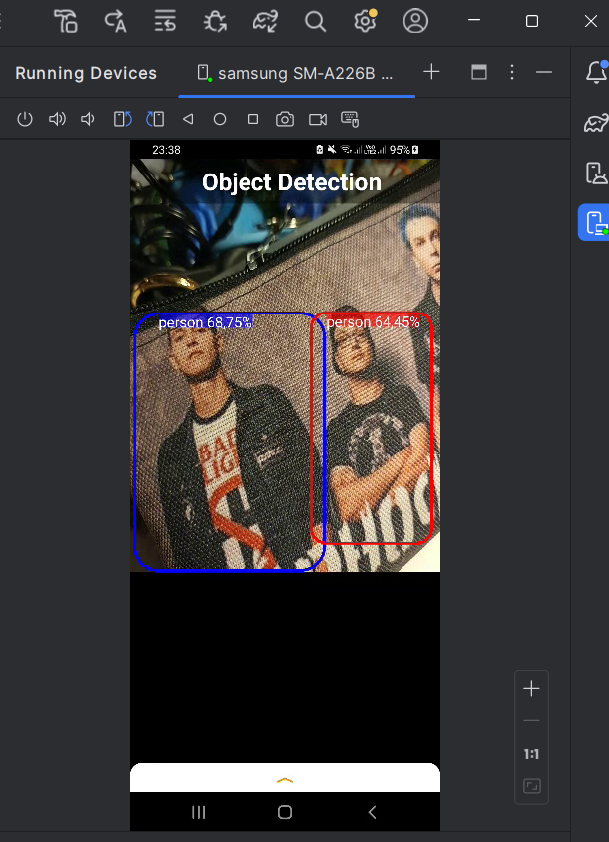


Рисунок 38. Обнаружение человека

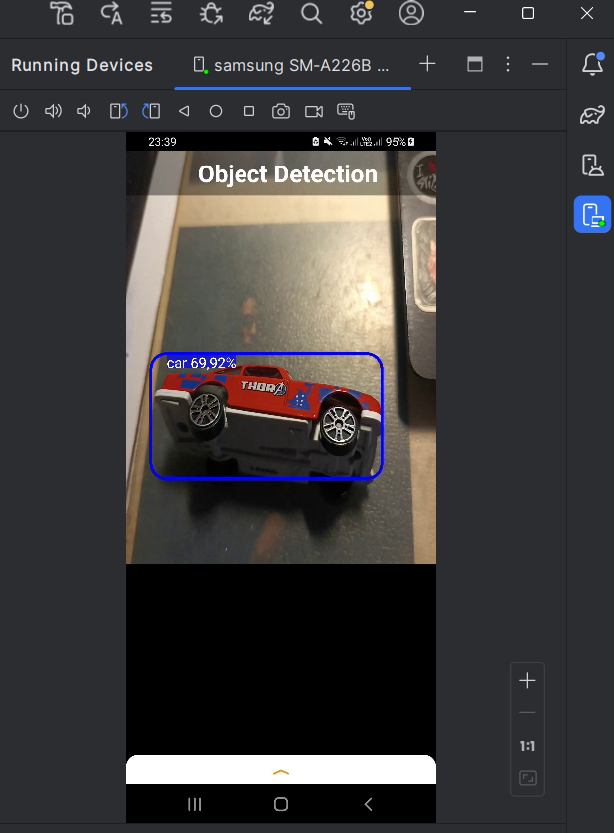


Рисунок 39. Обнаружение машины

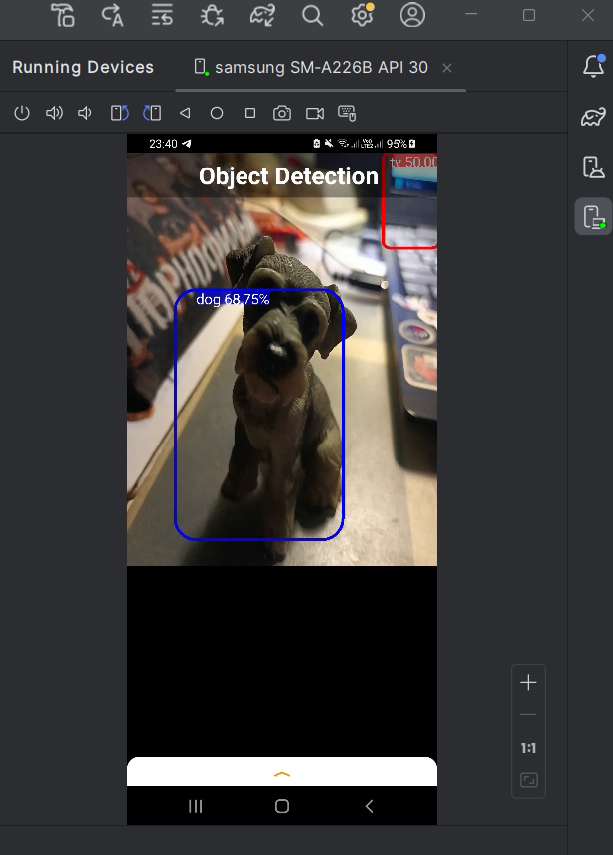


Рисунок 40. Обнаружение собаки

# 3 Выводы

В ходе выполнения были запущены приложения с помощью Android Studio, проверена их работоспособность. Обучена модель для детекции животных. Неуспешный запуск пикачу-детектора. Далее предполагается объединение с другими моделями для детекции других видов животных, проверка в приложении путем добавления моделей туда.

# 

# 4 Список литературы

1. Распознавание изображений на Python с помощью TensorFlow и Keras [Электронный ресурс]. Режим доступа: <https://evileg.com/ru/post/619/> (дата обращения 28.05.2024).
2. GitHub// tensorflow [Электронный ресурс]. Режим доступа: <https://github.com/tensorflow/> (дата обращения 28.05.2024).
3. TensorFlow// Обнаружение объектов с помощью Android [Электронный ресурс]. Режим доступа: <https://www.tensorflow.org/lite/android/tutorials/object_detection?hl=ru> (дата обращения 28.05.2024).
4. GitHub// juandes [Электронный ресурс]. Режим доступа: <https://github.com/juandes/pikachu-detection> (дата обращения 28.05.2024).
5. Protobuf [Электронный ресурс]. Режим доступа:

<https://developers.google.com/protocol-buffers/docs/downloads> (дата обращения 28.05.2024).

1. GitHub// PSofya [Электронный ресурс]. Режим доступа:

<https://github.com/PSofya/dz_chip/tree/main> (дата обращения 10.06.2024).