МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Технология программирования
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Применение нейронной сети для распознавания деталей оборудования с помощью мобильного телефона»

Студент Д.Д. Юрченко (И.О. Фамилия) _____ (личная подпись)

Руководитель к.т.н., доцент, Э.В. Егорова
(ученая степень, звание, И.О. Фамилия)

Консультант М.А. Дайнеко
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

Аннотация

Название бакалаврской работы – «Применение свёрточной нейронной сети для распознавания деталей оборудования с помощью мобильного телефона».

Объект исследования – процесс функционирования свёрточных нейронных сетей, архитектуры выбранной нами сети и использования её в мобильном приложении.

Предмет исследования – процесс взаимодействия свёрточных нейронных сетей с мобильным устройством. Предмет исследования – модель сверточной нейронной сети.

В первой главе затрагиваются вопросы процесса работы сверточных нейронных сетей, способы и сферы их применения и на основании полученных данных формулируется новая задача и требования к её непосредственному решению.

Во второй главе описывается архитектура выбранной для решения свёрточной нейронной сети и причины, по которым она была выбрана. Также проводятся исследования для выбора средств обучения модели.

В третьей главе предоставляются основные аспекты разработки.

В заключении подводятся итоги исследования, формируются окончательные выводы и результаты проделанной работы.

Бакалаврская работа выполнена по заказу НОЦ «Математические модели, распределенные вычисления и системы» на 47 страницах, состоит из введения, трёх глав, включающих 23 изображений, 8 листингов кода и 7 формул, заключения, списка используемых источников, включающего 18 источников, из них 8 на иностранном языке, и 1 приложения.

Abstract

The present graduation work is dedicated to the use of convolution neural networks for recognizing equipment parts via mobile phone.

The graduation work consists of 3 parts, 23 figures, 7 formulae, a list of 18 references including 8 foreign sources and 1 appendix.

The purpose of the research is developing a mobile phone application that will use the machine teaching model to recognize objects.

The object of the research is the functioning process of the convolution neural networks, the architecture of the network selected and its use in the mobile phone application.

The subject of the research is the process of interacting the convolution neural networks with a mobile device, as well as the convolution neural network model.

To reach the purpose of the research, the following objectives have to be achieved: to analyze the existing methods of applying the convolution neural networks; to study MobileNetV2 architecture and its advantage over other models; to ensure that the neural network recognizes objects via a mobile phone; to develop a mobile application for it to interact with the machine teaching model.

The first part of the graduation work touches upon the convolution neural networks operation process, their methods and areas of using.

The second part deals with the architecture of the convolution neural network chosen and describes the reasons why it has been chosen. The research is also conducted to select the training tools for the model to be explained.

The third part of the graduation work is devoted to developing and implementing mobile phone applications with the convolution neural network.

It can be concluded that the investigation conducted provides a new way of using the convolution neural networks, as well as allows us to develop new technology and opens up new opportunities for its subsequent application.

Содержание

Введение

В современных реалиях, когда происходит быстрое развитие технологий, а автоматизация упрощает практически любой процесс жизни человека, неудивительно, что когда-то описываемое писателями-фантастами компьютерное зрение, развилось и достигло той точки, когда это уже не будущее, а реальность.

Бурное развитие теорий компьютерного зрения привело к появлению технологий машинного зрения. Машинное зрение – это технологии, которые помогают оборудованию увидеть процесс производства чего-либо, проанализировать данные и принять информированное решение. И все это за секунды.

Сверточная нейронная сеть – это одна из разновидностей нейронных сетей прямого распространения. Прямое распространение – это когда переменные нейроны разбиты на группы. Такие группы называют слоями. Когда же такая нейросеть используется для применения к данным, то активация слоев (значение этих переменных) подсчитывается последовательно: сначала значение активации первого слоя, потом значение активации второго слоя, и так до последнего слоя. Результаты активации последнего слоя служат конечной точкой выхода результатов свёрточной нейронной сети.

Актуальность данной работы состоит в применении алгоритмов машинного обучения в приложении, предназначенном для мобильного устройства.

Новизна бакалаврской работы заключается в новом способе использования свёрточной нейронной сети.

Цель исследования – разработка мобильного приложения, которое будет использовать модель машинного обучения для распознавания объектов.

Для достижения цели решаются следующие задачи:

- проанализировать существующие способы применения свёрточных нейронных сетей;
- изучить архитектуру MobileNetV2 и её преимущество над остальными моделями;
- обучить нейронную сеть для распознавания объектов с помощью мобильного телефона.
- разработать мобильное приложение для взаимодействия с моделью машинного обучения

Практическая ценность состоит в разработке приложения с использованием свёрточной нейронной сети для распознавания моделей.

Объект исследования – процесс взаимодействия свёрточных нейронных сетей с мобильным устройством. Предмет исследования – модель сверточной нейронной сети.

Бакалаврская работа состоит из введения, трёх глав, заключения, списка литературы и приложения.

В первой главе затрагиваются вопросы процесса работы сверточных нейронных сетей, способы и сферы их применения и на основании полученных данных формулируется новая задача и требования к её непосредственному решению. Во второй главе описывается архитектура выбранной для решения свёрточной нейронной сети и причины, по которым она была выбрана. Также проводятся исследования для выбора средств обучения модели. В третьей главе предоставляются основные аспекты разработки. В заключении подводятся итоги исследования, формируются окончательные выводы и результаты проделанной работы.

1 Анализ состояния вопроса применения свёрточных нейронных сетей в мобильных устройствах

1.1 Описание работы свёрточных нейросетей

Нейронная сеть представляет из себя математическую модель схожую по своим качествам на мозг человека, таким как:

- возможностью обрабатывать поступающую в него информацию из окружающей среды, которая в последствии используется для обучения;
- накоплением знаний путем настройки синоптических весов на связях между нейронами.

Пример простой схемы нейронной сети представлен на рисунке 1.1.

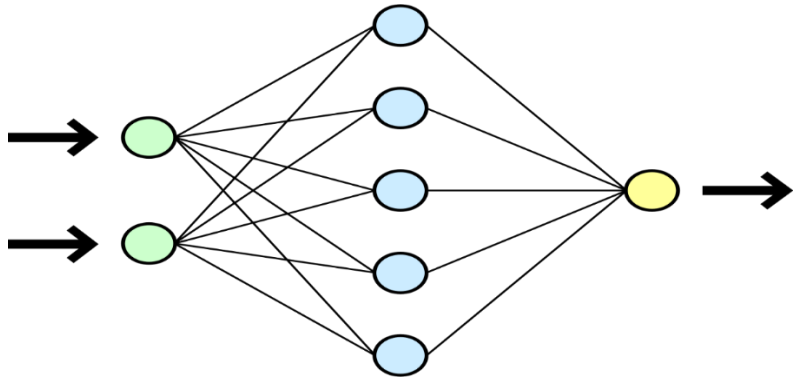

Рисунок 1.1 – Пример схемы простой нейронной сети

Принято, что нейронные сети состоят из трёх значимых слоёв:

- слой входных нейронов, обозначенный на схеме зелёными нейронами;
- n-ое количество слоёв скрытых нейронов, обозначенный на схеме одни слоем голубых нейронов;
- выходные нейроны, обозначенные на схеме лишь одним жёлтым нейроном.

Алгоритмом обучения называют процесс, который используется для обучения нейросети. Работа данного алгоритма состоит в выстраивании синаптических весов нейронной сети в определенном порядке. Это необходимо для получения необходимой структуры взаимосвязей нейронов.

Сверточные нейронные сети, СНС (англ. convolutional neural network, CNN) являются биологически вдохновленными вариантами многослойных персептронов (multilayer perceptron) [12]. Из ранних работ Хьюбел и Визела по зрительной коре животного, было выяснено, что зрительная кора содержит сложное расположение клеток. Данные клетки чувствительны к небольшим подобластям поля зрения, называемым рецептивным полем. Субрегионы представлены сеткой, которая покрывает всё поле зрения. Данные ячейки действуют как локальные фильтры в пространстве ввода зрительной информации и хорошо подходят для использования сильной пространственной локальной корреляции, присутствующей в естественных изображениях [22].

Если говорить простым языком, то сверточные нейронные сети представляют из себя нейронную сеть, которая умеет определять контуры и цветовое представление входного изображения. Отличительной чертой подобных сетей является сам процесс свёртки изображения. В ходе, которого мы “сворачиваем” матрицу признаков входного изображения и получаем еще одну матрицу признаков, но уже в уменьшенном варианте [18].

Данный процесс свёртки проходит благодаря ядру, состоящему из матрицы весов, и входному изображению, которое представляет из себя матрицу признаков, например, яркость пикселя в диапазоне от 0 до 255. Пример процесса операции свёртки представлен на рисунке 1.2. Ядро проходя по заданной ему области просчитывает выходную матрицу путем перемножения каждого параметра на вес с последующим их суммированием.

В приведенном выше примере у нас 5 * 5 = 25 знаков на входе и 3 * 3 = 9 знаков на выходе. Для стандартного слоя у нас будет весовая матрица 25 * 9 = 225 параметров. Каждый выходной объект будет взвешенной суммой всех объектов на входе. Свертка позволяет выполнить такую операцию только с 9 параметрами, потому что каждый знак на выходе получается путем анализа не каждого знака на входе, а только одного входа,

расположенного «примерно в одном месте». [17].

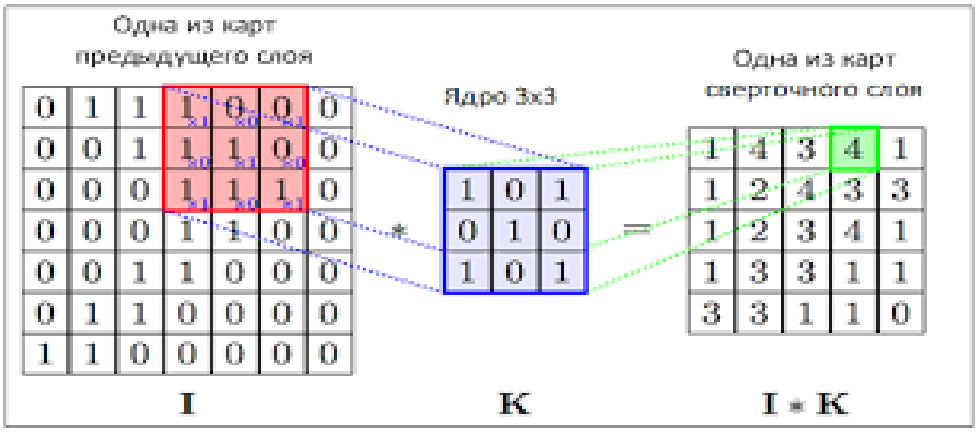

Рисунок 1.2 – Процесс операции свёртки

Приведенное выше объяснение касается только одноканальных изображений. На практике же мы обычно имеем дело с многоканальными изображениями, а в качестве примера для дальнейшего объяснения возьмем за основу трёхканальные, состоящие из красного, зеленого и синего каналов, известные как RGB изображения, так как они являются наиболее популярными.

Сам процесс свёртки любого многоканального изображения отличается не сильно и представлен на рисунке 1.3. Он заключается в том, что для каждого из каналов используется своё отдельное ядро со своими собственными весами нейронов [22].

При этом каждое подобное ядро может иметь свой собственный вес, изменяя значение, которого, можно регулировать, какому из каналов стоит уделить больше внимания. То есть чем выше вес конкретного ядра, допустим красного канала, тем сильнее будет реакция на различия в образах красного. После того как пройдет процесс свертки каждого ядра, мы суммируем полученные значения, добавляем скалярное смещение и получаем один общий выходной канал. Данная совокупность ядер всех каналов и одного выходного канала называется фильтром [38]. Процесс свёртки одного фильтра изображён на рисунке 1.3.

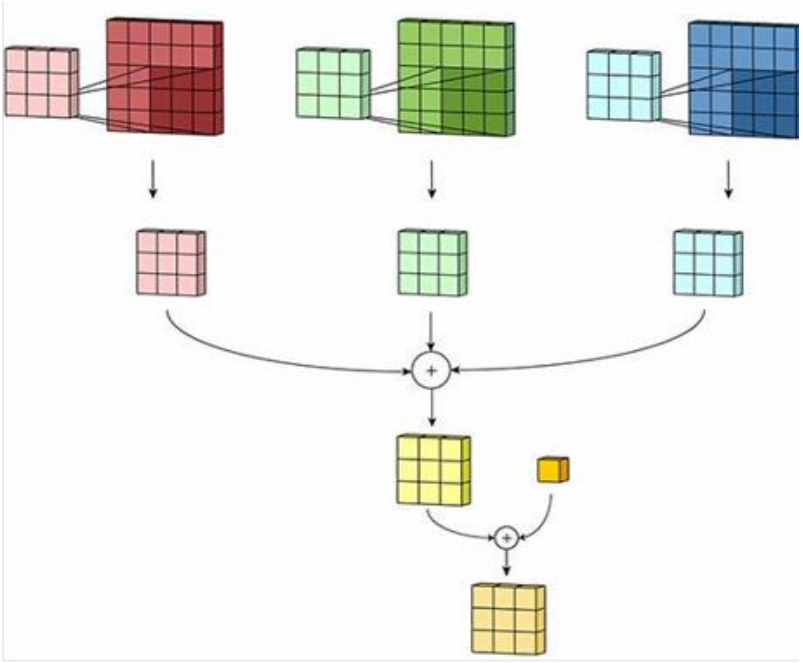

Рисунок 1.3 – Процесс свёртки одного фильтра

Результат для любого количества фильтров идентичен: каждый фильтр обрабатывает вход с собственным набором ядер и скалярным смещением в соответствии с процессом, описанным выше, создавая один выходной канал. Затем они объединяются, чтобы получить общий выходной сигнал с количеством выходных каналов, равным количеству фильтров. В этом случае нелинейность обычно применяется перед передачей ввода на другой слой свертки, который затем повторяет этот процесс.

1.2 Обзор применения алгоритмов свёрточных нейронных сетей

К классическим задачам, в ходе которых часто применяются CNN относятся [15]:

- определение границ;
- определение вектора нормали;
- определение объектов внимания
- семантическая сегментация;
- семантическое выделение границ;
- самая высокоуровневая задача – распознавание самих объектов.

Примеры выполнения данных задач приведены на рисунке 1.4

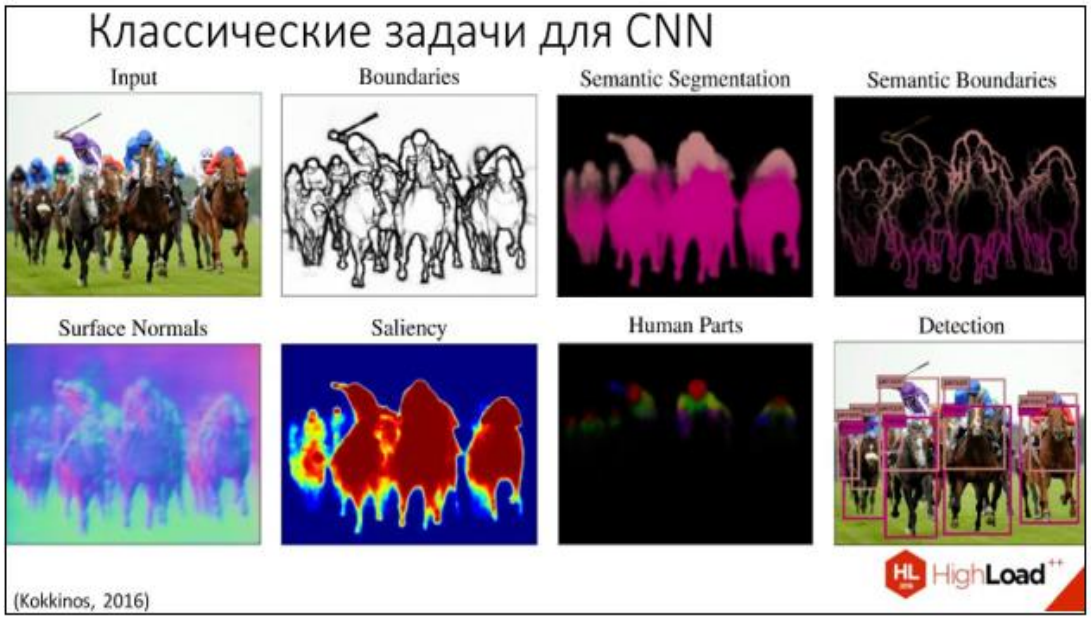

Рисунок 1.4 – Пример выполнения задач CNN

Определение границ (Boundaries) – самая низкоуровневая задача среди перечисленных. Свёрточные нейронные сети, выполняющие данную задачу, находят себя во множестве разнообразных сфер применения. Начиная от дорожного движения, где их используют для определения автомобильных номеров с камер дорожного наблюдения, заканчивая распознаванием всем известных captcha. Определение вектора нормали (Surface normals) – позволяет реконструировать трёхмерное изображение из двухмерного [21]. Благодаря нейронным сетям, выполняющим эту задачу, получают множество снимков нормалей и реконструируют объемную 3D модель чего либо, будь то ваза, здание или реконструированное трехмерное компьютерное представление куска поверхности, созданное с помощью снимков со спутника.

Определение объектов внимания или задача локализации (Saliency) – это нейронная сеть, которая занимается локализацией объекта внимания. Обученные на данную задачу свёрточные нейронные сети, способны отыскать как грибы в лесу, так и прогнозировать содержательность видео

[13].

Семантическая сегментация (Semantic segmentation) и семантическое выделение границ (Semantic boundaries) позволяют делить объекты на классы в соответствии с их структурой, ничего не зная об этих объектах. Пожалуй, одна из самых ключевых задач на сегодняшний день для нейронных сетей и компьютерного зрения в целом, так как именно решение этих задач позволяют компьютеру осознавать сцену, которую он видит, целиком. Сфера применения обширна, только на сегодняшний день данные сверточные нейронные сети используются в автопилотируемых транспортных средствах, при взаимодействии компьютера и человека или при использовании виртуальной реальности. Как можно понять, были перечислены довольно сложные задачи, а это только малая часть из тех, в которых может пригодиться семантическая сегментация.

Выделение частей тела человека (Human parts) – подвид задачи семантической сегментации, по которой особое внимание уделяется выделению человека и определению его частей тела. Практическое применение находит в медицине.

Задача распознавания (Detection) – самая высокоуровневая задача из всех перечисленных. Название говорит само за себя. Свёрточные нейронные сети, выполняющие данную задачу, находят и классифицируют объекты на изображении.

С помощью данных свёрточных нейронных сетей можно определять виды и подвиды животных на изображении, распознавать человека по фотографии и находить его профиль в социальных сетях.

1.3 Ограничения на условия работы разрабатываемой системы и используемым технологиям

Целевой аудиторией разрабатываемой системы являются физические лица. Основной сценарий использования системы – генерация и просмотр

контента на предприятиях. Таким образом, возникает естественное требование к системе – возможность работать на мобильных устройствах общего назначения:

- телефонах;
- планшетах;
- прочих устройствах, оснащённых видеокамерой, экранов и достаточными вычислительными мощностями.

Данное требование, в связи с технологическими особенностями мобильных устройств, накладывает ряд ограничений на используемые технологии, а также архитектуру системы [20].

Для достижения высоких характеристик по компактности устройства, проектировщики мобильных устройств отказываются от активных систем охлаждения. И даже пассивные системы охлаждения предельно минифицируются. Это приводит к значительному вынужденному снижению тактовых частот вычислительных ядер для предотвращения перегрева [17]. Таким образом даже при условии использования современных техпроцессов при производстве мобильных процессоров, получить значительные вычислительные мощности на мобильном устройстве невозможно.

Кроме того, в отличие от полноразмерных компьютеров, в мобильных устройствах (по тем же причинам) невозможно использовать мощные графические модули, которые позволяют распараллеливать вычисления по числам с плавающей запятой. Единственными доступными вариантами обработки графики являются компактные и холодные графические сопроцессоры, чьи вычислительные возможности ограничены.

Третье ограничение мобильных платформ - время автономной работы. В то же время важна не только продолжительность работы без подключения к сети, но и интенсивность энергопотребления. Если батарея разряжается слишком быстро, батарея нагревается и существует риск разрушения энергетических элементов и даже возгорания устройства.

Учитывая данную особенность, наше программное обеспечение не должно быть требовательно к ресурсам устройства и должно работать стабильно.

1.4 Постановка задачи на реализацию программного обеспечения

Отталкиваясь от обзора классических задач, для выполнения которых применяются свёрточные нейронные сети, можно сделать вывод, что применение CNN не ограничивается только ими, и можно найти ещё достаточное количество способов, как использовать данные нейронные сети в нестандартных для неё сферах применения.

Исходя из выше перечисленных аспектов, необходимо реализовать программное обеспечение, которое будет из себя представлять мобильное приложение, которое должно будет:

- взаимодействовать с моделью машинного обучения;
- получать на вход изображение;
- распознавать объекты REAL-time на изображении;
- не быть требовательна к ресурсам устройства;
- работать на платформе Android.

Таким образом, было описано понятие свёрточной нейронной сети и её отличие от классической нейронной сети. Так же были проанализированы существующие классические задачи, выполняемые свёрточными нейронными сетями. Для каждой задачи описана сфера применения, для которой разрабатываются конкретные CNN.

Исходя из полученных данных была сформулирована задача, выполнение которой подразумевает разрабатываемое программное обеспечение.

2 Теоретическое обоснование роли нейронной сети для распознавания деталей оборудования

2.1 Обзор программных средств для обучения нейронной сети

1) TensorFlow - это библиотека программного обеспечения машинного обучения с открытым исходным кодом, разработанная Google. Данная библиотека позволяет создавать и обучать нейронные сети различной архитектуры для обнаружения и распознавания шаблонов и поиска взаимосвязей. TensorFlow также включает TensorBoard, который представляет собой инструмент визуализации на основе браузера для оценки эффективности обучения и сетевых параметров модели. TensorFlow достигает своей производительности путем распараллеливания задач между процессорами и графическими процессорами. Ядро каждой операции реализовано на C++ с использованием библиотек Eigen и cuDNN для повышения производительности [2].

Каждый расчет в TensorFlow представлен в виде графика потока данных, он также является графиком вычислений. График вычислений - это модель, которая описывает, как будут выполняться расчеты. Важно отметить, что составление графика вычислений и выполнение операций в данной структуре - это два разных процесса. Граф состоит из переменных и операций. Он рассчитывает тензоры - многомерные массивы, которые, однако, могут быть числом или вектором.

Подсчет выполняется в сессиях. Есть два типа сессий - регулярные и интерактивные. Интерактивный сеанс подходит для выполнения в консоли. Сеанс хранит состояние переменных и очередей. Явное создание сеансов и графиков обеспечивает правильное освобождение ресурсов памяти [6].

В графе каждая вершина имеет 0 или больше входов и 0 или больше выходов, и представляет собой реализацию операции. Тензоры представляют собой рёбра графа, а именно массивы произвольного размера (тип массива указывается во время построения графа). Особые вершины, управляющие

зависимости (control dependencies), также могут быть в графе: они указывают, что исходный узел для контрольной зависимости должен закончить выполнение до того, как узел получателя контрольной зависимости начнет выполняться.

Каждая операция имеет название и представляет собой абстрактное вычисление (например, суммирование). У операции могут быть атрибуты: например, возможность сделать операцию полиморфной для разных типов тензоров [6]. Ядро — специфическая реализация операции, которая может выполнена на определенном типе устройства (центральный или графический процессор).

Переменная — особый вид операции, возвращающий указатель на постоянно меняющийся тензор: такая переменная не исчезает после единичного использования графа. Указатели на подобные тензоры передаются многочисленным операциям, которые затем изменяют указанный тензор.

В задачах машинного обучения, параметры модели обычно хранят тензоры в переменных, которые обновляются на каждом шаге обучения.

Уникальность TensorFlow заключается в возможности проводить частичные подграфовые вычисления. Эта особенность позволяет сделать разбиение нейронной сети, а значит можно использовать распределенное обучение. Также:

- Tensorboard: визуализация модели и возможность исследовать порядок вычислений в графе;
- TensorFlow может использована как на мобильных, так и на более мощных устройствах.
- в TensorFlow производные задаются автоматически: этот процесс называется автоматическим дифференцированием.

TensorFlow использует обратный режим автоматического дифференцирования для операций градиентов и метода конечных разностей

для тестов, которые проверяют правильность работы градиента.

2) Scikit-learn - это библиотека для машинного обучения для языка программирования Python. Она имеет различные алгоритмы классификации, регрессии и кластеризации, случайные леса, повышение градиента, k-среднее и DBSCAN, предназначен для взаимодействия с численными библиотеками NumPy и SciPy.

Возможности:

- кластеризация (Clustering): для группировки неразмеченных данных, например, метод k-средних (k-means)
- перекрестная проверка (Cross Validation): для оценки эффективности работы модели на независимых данных
- наборы данных (Datasets): для тестовых наборов данных и для генерации наборов данных с определенными свойствами для исследования поведенческих свойств модели
- сокращение размерности (Dimensionality Reduction): для уменьшения количества атрибутов для визуализации и отбора признаков (Feature Selection), например, метод главных компонент (Principal Component Analysis)
- алгоритмические композиции (Ensemble Methods): для комбинирования предсказаний нескольких моделей
- извлечение признаков (Feature Extraction): определение атрибутов в изображениях и текстовых данных
- отбор признаков (Feature Selection): для выявления значимых атрибутов, на основе которых будет построена модель
- оптимизация параметров алгоритма (Parameter Tuning): для получения максимально эффективной отдачи от модели
- множественное обучение (Manifold Learning): для нелинейного сокращения размерности данных

Алгоритмы обучения с учителем: огромный набор методов не ограничивается обобщенными линейными моделями, дискриминантным анализом, наивным байесовским классификатором, нейронными сетями, методом опорных векторов и деревьями принятия решений.

TensorFlow оперирует статическим вычислительным графом. То есть вначале мы определяем граф, далее запускаем вычисления и, если необходимо внести изменения в архитектуру, заново обучаем модель. Такой подход выбран ради эффективности, но многие современные нейросетевые инструменты умеют учитывать уточнения в процессе обучения без существенной потери скорости обучения.

3) PyTorch. В отличие от TensorFlow, библиотека PyTorch работает с динамически обновляемым графиком. То есть он позволяет вносить изменения в архитектуру в процессе [14].

Платформа PyTorch была разработана для сервисов Facebook, но уже используется для ее собственных задач такими компаниями, как Twitter и Salesforce. В PyTorch вы можете использовать стандартные отладчики, например, pdb или PyCharm. Процесс обучения нейронной сети прост и понятен. В то же время PyTorch поддерживает модель параллелизма данных и распределенного обучения, а также содержит множество предварительно обученных моделей.

Но в отличие от TensorFlow, описанная среда глубокого обучения гораздо менее гибка в поддержке различных платформ. Также в PyTorch нет нативных инструментов для визуализации данных. Тем не менее, существует сторонний аналог под названием TensorboardX.

4) Keras. Наиболее минималистичный подход к использованию TensorFlow, Theano или CNTK дает высокоуровневая оболочка Keras. Прототипирование здесь облегчено до предела. Создание массивных моделей глубокого обучения в Keras сведены до однострoковых функций. Но такая

стратегия делает Keras менее конфигурируемой средой, чем низкоуровневые фреймворки.

5) Theano - это библиотека Python для эффективной обработки математических выражений с участием многомерных массивов (также известных как тензоры). Это общий выбор для реализации моделей нейронных сетей. Theano разрабатывается в университете Монреаля в группе под руководством Йошуа Бенжио с 2008 года.

Некоторые функции включают в себя:

- автоматическое дифференцирование - нужно только реализовать передовую (прогнозирующую) часть модели, и Theano автоматически выяснит, как рассчитать градиенты в различных точках, что позволит выполнить градиентный спуск для обучения модели.
- прозрачное использование графического процессора - можно написать один и тот же код и запустить его либо на CPU, либо на GPU. В частности, Theano выяснит, какие части вычислений следует перенести на графический процессор.
- оптимизация скорости и стабильности - Theano внутренне реорганизует и оптимизирует вычисления, чтобы они работали быстрее и были более численно устойчивы. Он также попытается скомпилировать некоторые операции в код на C, чтобы ускорить вычисления.

Технически, Theano на самом деле не является библиотекой машинного обучения, поскольку она не предоставляет готовые модели, которые вы можно обучать на своем наборе данных. Это математическая библиотека, которая предоставляет инструменты для построения собственных моделей машинного обучения.

В материалах конференций по искусственному интеллекту и в конкурсах Kaggle исследователи нередко отдают предпочтение PyTorch. Связано это с тем, что PyTorch гораздо лучше подходит для небольших проектов и прототипирования. Когда же речь заходит о кроссплатформенных

решениях, TensorFlow выглядит более подходящим выбором. Так как нам требуется обучить нейросеть распознаваию объектов с помощью мобильного приложения, то было принято решение остановить выбор на Tensorflow.

2.2 Обзор существующих мобильных платформ и выбор платформы для разработки мобильного приложения

Наличие операционной системы (ОС) является главной особенностью, которая отличает смартфон от обычного мобильного телефона. При выборе конкретной модели телефона или устройства операционная система часто является определяющим фактором. Наиболее распространенные операционные системы для смартфонов и мобильных платформ:

Android – портативная(сетевая) операционная система для смартфонов, планшетных ПК, электронные книг, цифровых плееров, часов и нетбуков на базе ядра Linux. Первоначально разработанная Android Inc., которую затем купил Google. Впоследствии Google инициировала создание альянса Open Handset Alliance (OHA), который сейчас занимается поддержкой и дальнейшим развитием платформы. Android позволяет создавать приложения на основе Java, который управляют устройством через разработанные Google библиотеки. Android Native Development Kit позволяет системе использовать библиотеки и компоненты приложений, написанных на Си других языках;

ОС IOS (до 24 июня 2010 года – iPhone OS) –это мобильная операционная система, разработанная и изготовленная американской компанией Apple. Она была выпущена в 2007 году; первоначально –для iPhone и Ipod Touch, а позже –для таких устройств, как Ipad и Apple TV. В отличие от Windows Phone и Google Android, доступна только для устройств, производимых Apple;

В настоящее время Android развивается в геометрической прогрессии: каждый год число пользователей этой операционной системы постоянно растет. Согласно последнему отчету компании Canalys, ведущего аналитика

индустрии высоких технологий, операционная система Android занимает 69,2% мирового рынка мобильных устройств. Конечно, этот факт привлекает внимание многих разработчиков создавать мобильные приложения, специально для Android. Возможно, на сегодняшний день она является самой популярной и интересной системой. Разработчики дают пользователям уникальную возможность установить набор свободного программного обеспечения, можно создать программы для системы и продавать их в специализированном интернет-магазине. Руководствуясь этим, было принято решение остановить свой выбор на платформе Android.

2.3 Архитектура и описание принципа работы нейронной сети MobileNetV2

Воспользовавшись уже приведенными исследованиями на рисунке 2.1 в выборе модели для обучения нейросети, было принято решение использовать нейронную сеть MobileNetV2, так как она имеет небольшой вес, что позволяет хранить обученную модель на мобильном устройстве. Даже при небольшом весе модель имеет очень высокую точность и время отклика при работе на 4 ядрах мобильного устройства [7].

При изучении исследования был осуществлен упор на поиск нейронной сети, подходящей для использования на мобильном устройстве, а также подходящей для работы с библиотекой TensorFlow. Если нейронную сеть сравнить с другими сетями для решения задачи классификации, то мы получим следующие результаты, которые отображены на рисунке 2.1:

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

Рисунок 2.1 – Сравнение показателей нейросетей для решения задачи классификации

MobileNetV2 превосходит MobileNetV1 и ShuffleNet с сопоставимым размером модели и вычислительными затратами. С множителем ширины 1.4 MobileNetV2 превосходит ShuffleNet (× 2) и NASNet, имея меньшее кол-во времени на распознавание.

В предыдущей версии нейронной сети MobileNetV1 была представлена глубинная свёртка, которая значительно снижает размер модели сети, которая подходит для мобильных устройств или любых устройств с низкой вычислительной мощностью. В MobileNetV2 представлен новый улучшенный модуль с инвертированной остаточной структурой. Нелинейности в узких слоях устраняются на этот раз. С MobileNetV2 в качестве основы для извлечения признаков, современные достижения также достигнуты для обнаружения объекта и семантической сегментации.

На рисунке 2.2 можно увидеть структуру сверточных блоков MobileNetV2.

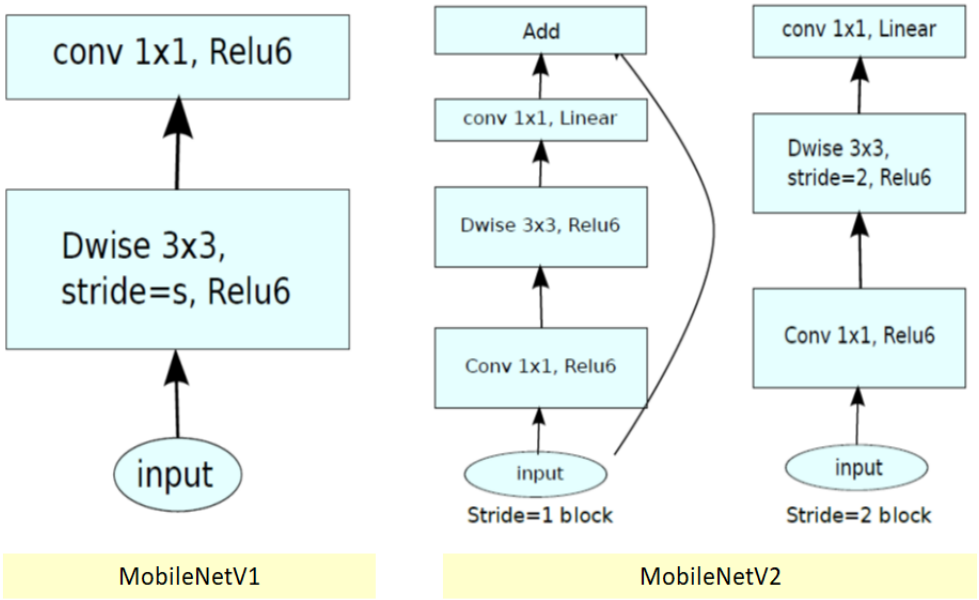

Рисунок 2.2 – Структура сверточных блоков MobileNetV2

В MobileNetV1 есть 2 слоя:

1. Первый слой называется глубинной сверткой, он выполняет фильтрацию путем применения одного сверточного фильтра на входной канал.

2. Второй уровень представляет собой свертку 1 × 1, называемую точечной сверткой, которая отвечает за создание новых функций посредством вычисления линейных комбинаций входных каналов.

ReLU6 используется здесь для сравнения. ReLU6 используется благодаря своей надежности при использовании с низкой точностью вычислений на основе MobileNetV1.

В MobileNetV2 есть два типа блоков. Один - остаточный блок с шагом, другой - блок с шагом 2 для уменьшения.

Есть 3 слоя для обоих типов блоков:

1. В новой версии первый уровень имеет свертку 1 × 1 с ReLU6.

2. Второй слой - это глубокая свертка.

3. Третий слой - это еще одна свертка 1 × 1, но без какой-либо нелинейности. Утверждается, что, если снова использовать ReLU, глубокие сети будут иметь мощность линейного классификатора только в ненулевой

части объема выходной области.

На рисунке 2.3 изображена общая архитектура MobileNetV2.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Рисунок 2.3 – Общая архитектура нейронной сети MobileNetV2

MobileNetV2 превосходит MobileNetV1 и ShuffleNet с сопоставимым размером модели и вычислительными затратами.

Блок MobileNet состоит из трёх слоёв:

1. Сначала идёт pointwise convolution с большим количеством каналов, называемый expansion layer. На входе этот слой принимает тензор размерности (формула 1):

$$D\_f * D\_f * C\_\{in\} \qquad 1$$

На выходе данный слой выдает следующий тензор (формула 1.1),

$$D\_f * D\_f * (t * C\_\{in\}) \qquad 1.1$$

где t - новый гиперпараметр, названный уровнем расширения (в оригинале expansion factor). Этот слой создает отображение входного тензора в пространстве большой размерности.

2. Затем идёт depthwise convolution с ReLU6-активацией. Этот слой вместе с предыдущим по сути образует уже знакомый нам строительный блок MobileNetV1.На входе этот слой принимает тензор размерности (формула 1.2):

$$D\_f * D\_f * (t * C\_\{in\}) \qquad 1.2$$

На выходе данный слой выдает следующий тензор (формула 1.3),

$$(D\_f/s) * (D\_f/s) * (t * C\_\{in\}) \qquad 1.3$$

где s - шаг свертки (stride), ведь, depthwise convolution не меняет число каналов.

3. В конце идёт 1x1-свертка с линейной функцией активации, понижающая число каналов. «Целевое многообразие» высокой размерности, полученное после предыдущих шагов, можно «уложить» в подпространство меньшей размерности без потери полезной информации, что, собственно и делается на этом шаге. На входе такой слой принимает тензор размерности (формула 1.4),

$$(D\_f/s) * (D\_f/s) * (t * C\_\{in\}) \qquad 1.4$$

На выходе данный слой выдает следующий тензор (формула 1.5),

$$(D\_f/s) * (D\_f/s) * C\_\{out\} \qquad 1.5$$

где C - количество каналов на выходе блока.

Структуру слоев MobileNevV2 вы можете видеть на рисунке 2.4/

Именно третий слой в этом блоке, называемый bottleneck layer, и определеляется основное отличие второго поколения MobileNet от первого.

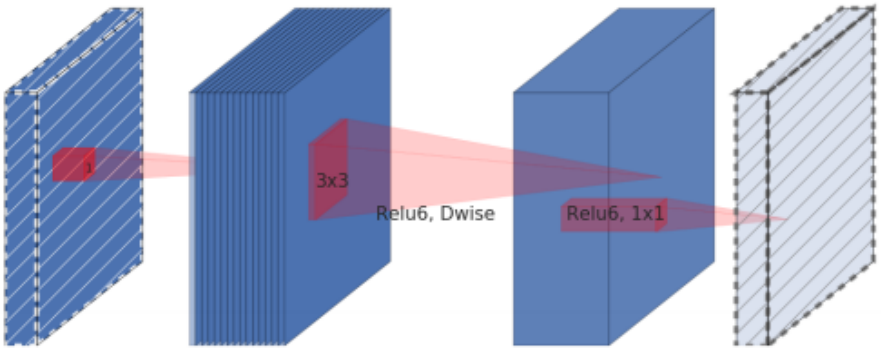

Рисунок 2.4 – Структура слоев MobileNetV2

Таким образом, в нашем приложении будет использоваться модель нейросети MobileNetV2.

2.4 Анализ и выбор программного стека для обучения нейронной сети и разработки мобильного приложения

Перед тем как переходить к реализации, определимся с программным стеком. Разобьем процесс анализа и выбора на три шага:

1. Выбор объекта. Модуль классификации изображений на основе сверточных нейронных сетей.

2. Выбор основных характеристик объекта, которые выражаются отвлеченным понятием: Фреймворк; Язык программирования; Среда разработки.

3. Указание всевозможных вариантов реализации характеристик, выбранных на шаге 2.

1. Фреймворк: TensorFlow, PyTorch
2. Язык программирования: Python, C#
3. Среда разработки: PyCharm, Visual Studio

Критерии оценки:

1. Связанность компонентов;
2. Сложность реализации;
3. Временные затраты на разработку.

Шаг 4. Рассмотрение различных полученных комбинаций.

Вариант 1: Tensor Flow + Python + PyCharm. Вариант 1 соответствует выбранным нами критериям. Среда разработки PyCharm используется исключительно с языком программирования Python, а среда TensorFlow реализована на языке программирования Python. Сложность реализации этой опции относительно невелика, поскольку язык программирования Python известен, среда разработки PyCharm также изучается, а инфраструктура TensorFlow имеет прозрачную модульную архитектуру с множеством внешних интерфейсов, благодаря чему довольно просто воплощать проекты в жизнь. Таким образом, временные затраты будут соответственно незначительными.

Вариант 2: Tensor Flow + Python + Visual Studio. Вариант 2 соответствует выбранным нами критериям. Среду разработки Visual Studio

можно использовать с языком программирования Python, а инфраструктура TensorFlow реализована на языке программирования Python. Сложность реализации этой опции относительно невелика, поскольку язык программирования Python известен, среда разработки Visual Studio также была изучена, а инфраструктура TensorFlow имеет прозрачную модульную архитектуру со многими интерфейсами, благодаря чему она довольно легко реализована, однако, разработка в этой среде на выбранном языке программирования неудобна. Затраты времени будут средними.

Вариант 3: Tensor Flow + C# + PyCharm. Вариант 3 не соответствует выбранным нами критериям. Среда разработки PyCharm используется исключительно с языком программирования Python и не может использоваться с языком программирования C#. Отсюда следует, что этот вариант нам не подходит, и дальнейший анализ этого варианта не имеет смысла.

Вариант 4: Tensor Flow + C# + Visual Studio. Вариант 4 отвечает выбранному нами критерию связанности компонентов. Среда разработки Visual Studio используется с языком программирования C#, однако, фреймворк TensorFlow не может быть реализован на данном языке программирования. Из этого следует, что данный вариант нам не подходит и дальнейший анализ данного варианта не имеет смысла.

Вариант 5: PyTorch + Python + PyCharm. Вариант 5 соответствует критерию связности компонентов. Среда разработки PyCharm используется исключительно с языком программирования Python, а среда PyTorch реализована на языке программирования Python. Сложность реализации этой опции относительно высока, так как язык программирования Python известен, среда разработки PyCharm также была изучена, но в коде платформы PyTorch трудно ориентироваться, отладку и рефакторинг нелегко, затрудняя реализацию. Таким образом, временные затраты будут умеренными.

Вариант 6: PyTorch + Python + Visual Studio. Вариант 6 соответствует критерию связности компонентов, который мы выбрали. Среду разработки Visual Studio можно использовать с языком программирования Python, а среда PyTorch реализована на языке программирования Python. Сложность реализации этой опции относительно высока, так как язык программирования Python известен, среда разработки Visual Studio также была изучена, но в коде платформы PyTorch трудно ориентироваться, отладку нелегко и рефакторинг, что затрудняет реализацию в этой среде, а разработка в этой среде на выбранном нами языке программирования неудобна. Таким образом, временные затраты будут значительными.

Шаг 5. Выбор оптимального варианта по обобщенным критериям Оценки по частным критериям:

1. Отлично = 1,0
2. Очень хорошо = 0,75
3. Хорошо = 0,625
4. Удовлетворительно = 0,5
5. Посредственно = 0,25
6. Неудовлетворительно = 0

Результаты анализа вы можете видеть в таблице 1.

Таблица 1 – Результаты исследования

Критерий №	Связанность компонентов	Сложность реализации	Временные затраты на разработку	Итог
Вариант 1 • TensorFlow • Python • PyCharm	1,0	0,75	0,75	**2,5**
Вариант 2 • Tensorflow • Python	0,625	0,625	0,5	**1,75**

• Visual Studio				
Вариант 3 • PyTorch • Python • PyCharm	1,0	0,5	0,5	**2,0**

После проведения анализа вариант номер 1 оказался оптимальным для обучения сверточной нейронной сети. Мобильное приложение будет разрабатываться на платформе Android с использованием программного обеспечения Android Studio, так как это наиболее популярное и подходящее программное обеспечение для разработки мобильного приложения.

Итак, были рассмотрены средства разработки для обучения моделей машинного обучения. Выбор был остановлен на платформе TensorFlow, так как это наиболее подходящий инструмент для кроссплатформенной разработки. Была изучена архитектура нейронной сети MobileNet, а также выбран программный стек для реализации поставленной задачи.

3 Обучение нейронной сети и разработка мобильного приложения

3.1 Обучение нейронной сети

Для обучения нейросети распознаванию образов было принято решение воспользоваться сервисом Google Colaboratory.

Нашей целью будет обучить нейросеть распознавать детали компьютера с точностью около 90%. Для начала, соберем набор данных для обучения (пока остановимся на ~200 фото). На рисунке 3.1 представлен пример датасета, составленный из видеокарточек.

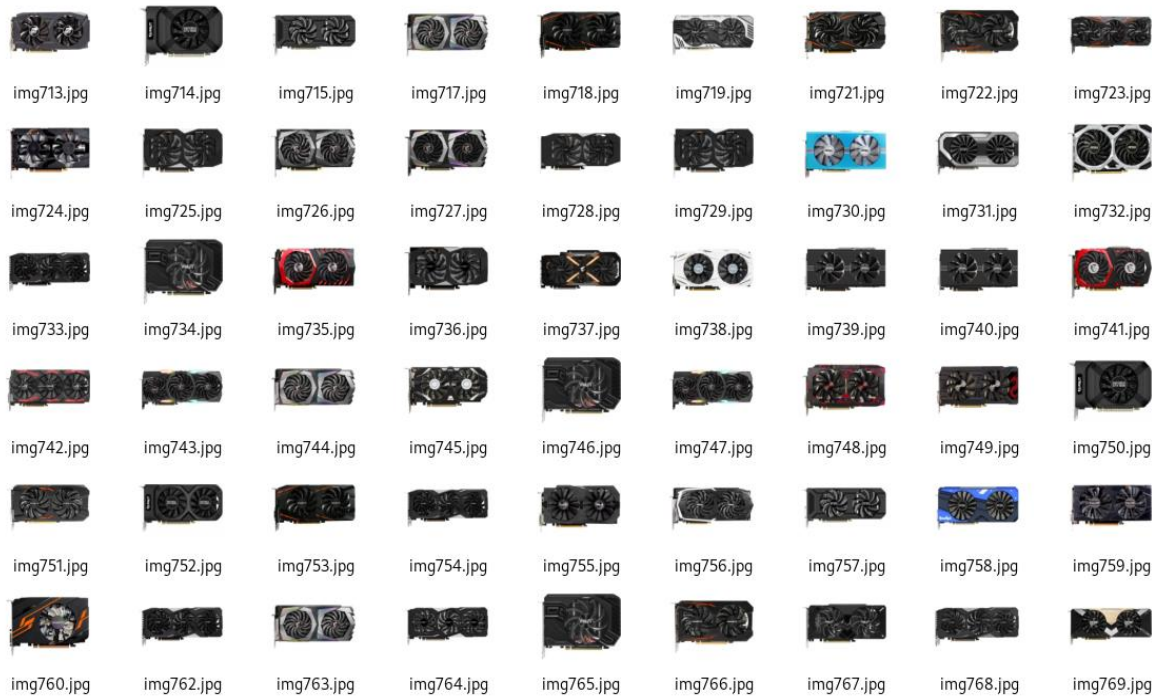

Рисунок 3.1 – Неподготовленные данные для тренировки нейронной сети

Для обучения нам потребуется Tensorflow Object Detection API, который мы будем использовать для подготовки данных для обучения. Для комфортной работы нам потребуется программное обеспечение Anaconda.

Напишем скрипт для подготовки рабочего окружения (рисунок 3.2):

```
#!/bin/bash

#Создадим окружение для работы:
conda create -n object_detection_prepare pip python=3.6

#И активируем его:
conda activate object_detection_prepare

#Установим зависимости, которые нам понадобятся:
pip install --ignore-installed --upgrade tensorflow==1.14
pip install --ignore-installed pandas
pip install --ignore-installed Pillow
pip install lxml
conda install pyqt=5
```

Рисунок 3.2 – Скрипт для подготовки рабочего окружения

Теперь подготовим изображения для обучения. Первым делом нам нужно привести их к одинаковому размеру. В Object Detection API есть инструмент для этого, им и воспользуемся. Скрипт для работы с инструментом представлен на рисунке 3.3.

```
#!/bin/bash

python ./object_detection/preprocessing/image_resize.py -i /
        ./object_detection/images --imageWidth=800 --imageHeight=600
```

Рисунок 3.3 – Скрипт для приведения изображений к одинаковому размеру

Этот скрипт изменит размер изображений до 800x600. После этого можно приступить к разметке данных, для этого воспользуемся утилитой LabelImg, которая присутствует в пакетном менеджере языка Python. Пример работы с данной утилитой вы можете увидеть на рисунке 3.4.

Рисунок 3.4 – Утилита LabelImg

С помощью данной утилиты выделяем объекты для распознавания и указываем их класс. В конкретном примере, это комплектующие компьютера. Сохраним метаданные (файлы *.xml) в той же папке. На рисунке 3.5 можно увидеть подготовленный датасет.

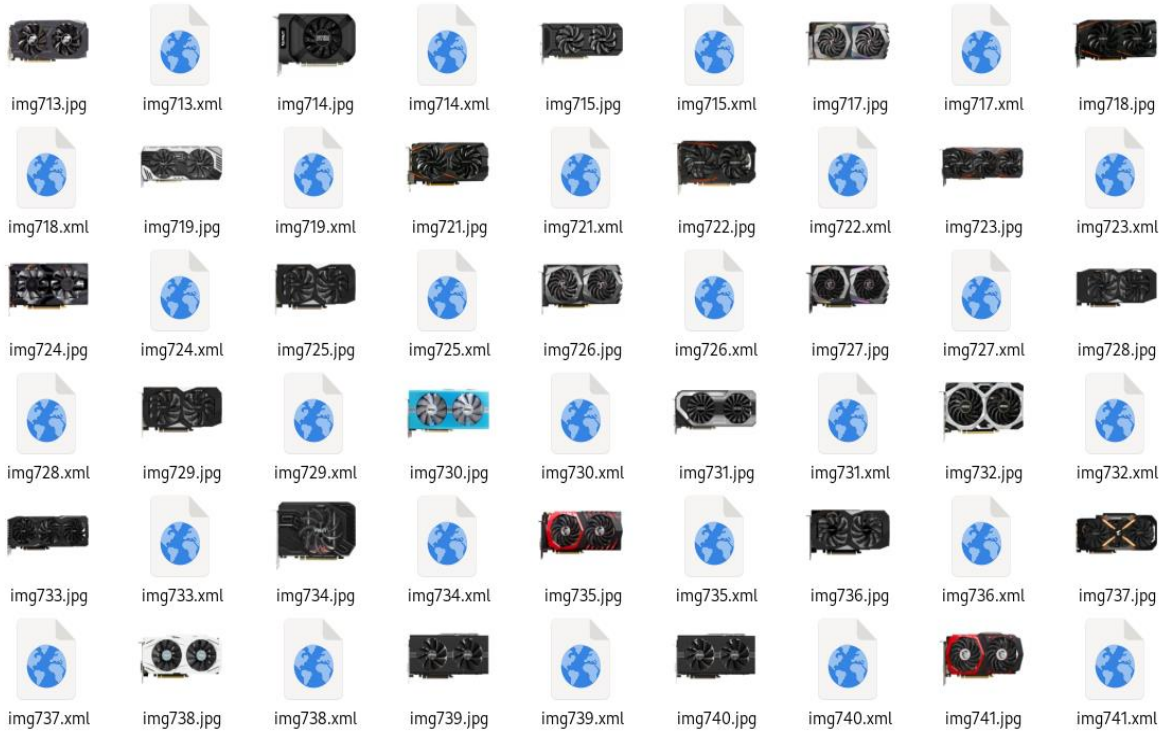

Рисунок 3.5 – Метаданные изображений

Следующим шагом отсортируем наши изображения на тренировочные и тестовые в следующем соотношении: 80/20. Теперь нужно их перенести в папки train и test, которые будут использоваться при обучении.

Теперь нужно создать папку annotations. В ней мы будем хранить файлы с метаданными необходимыми для обучения. Первым из них файл с метками - label_map.pbtxt. В нем мы укажем отношение класса объекта и целочисленного значения.

После этого для дальнейшей работы необходимо конвертировать метаданные в формат TFRecord. Для конвертации воспользуемся следующими скриптами. На рисунке 3.6 изображена реализация метода конвертации Xml файлов в Csv.

```
def xmlToCsv(path):
    xmlList = []
    for xmlFile in glob.glob(path + '/*.xml'):
        tree = ET.parse(xmlFile)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xmlList.append(value)
    columnName = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xmlList, columns=columnName)
    return xml_df


def main():

    parser = argparse.ArgumentParser()
    parser.add_argument('-in', '--input_xml', help='Файл для обработки', type=str, required=True)
    parser.add_argument('-out', '--output_csv', help='Файл после обработки ', type=str, required=True)
    args = parser.parse_args()

    xml_df = xmlToCsv(args.input_xml)
    xml_df.to_csv(args.output_csv, index=None)
    print('XML успешно конвертирован в CSV')
```

Рисунок 3.6 – Конвертация из XML в CSV

Запустим скрипт для подготовки наших данных. Скрипт для подготовки данных можно видеть на рисунке 3.7.

```
#!/bin/bash

#Из xml в csv

python xmlToCsv.py -i [FULL_PATH]/object_detection/training_demo/images/train \
                   -o [FULL_PATH]/object_detection/training_demo/annotations/train_labels.csv

python xmlToCsv.py -i [FULL_PATH]/object_detection/training_demo/images/test \
                   -o [FULL_PATH]/object_detection/training_demo/annotations/test_labels.csv

#Из csv в record

python generate_tfrecord.py --label_map_path=[FULL_PATH]\object_detection\training_demo\annotations\label_map.pbtxt \
                            --csv_input=[FULL_PATH]\object_detection\training_demo\annotations\train_labels.csv \
                            --output_path=[FULL_PATH]\object_detection\training_demo\annotations\train.record \
                            --img_path=[FULL_PATH]\object_detection\training_demo\images\train

python generate_tfrecord.py --label_map_path=[FULL_PATH]\object_detection\training_demo\annotations\label_map.pbtxt \
                            --csv_input=[FULL_PATH]\object_detection\training_demo\annotations\test_labels.csv \
                            --output_path=[FULL_PATH]\object_detection\training_demo\annotations\test.record \
                            --img_path=[FULL_PATH]\object_detection\training_demo\images\test
```

Рисунок 3.7 – Скрипт для подготовки стабовых данных

На этом мы закончили подготовку данных, теперь настроим модель, которую будем обучать.

Для этого немного изменим конфиг файла модели под наши нужды. Фрагмент конфига модели представлен на рисунке 3.8:

```
#Укажем количество классов:

model.ssd.num_classes: 9

#Укажем размер пакета (количество данных для обучения за одну итерацию), количество итераций и путь к сохраненной модели:

train_config.batch_size: 18
train_config.num_steps: 20000
train_config.fine_tune_checkpoint:"./training_demo/pre-trained-model/ssdlite_mobilenet_v2_coco/model.ckpt"

#Укажем количество фото в тренировочном наборе:

object_detection/training_demo/images/traineval_config.num_examples: 200

#Укажем путь к набору данных для тренировки:

train_input_reader.label_map_path: "./training_demo/annotations/label_map.pbtxt"
train_input_reader.tf_record_input_reader.input_path:"./training_demo/annotations/train.record"

#Укажем путь к тестовому набору данных:

eval_input_reader.label_map_path: "./training_demo/annotations/label_map.pbtxt"
```

Рисунок 3.8 – Файл настройки обучения

На этом подготовка данных закончена, перейдем к обучению.

Запускаем процесс обучения. Скрипт для запуска обучения представлен на рисунке 3.9.

```
#!/bin/bash

python ./models/research/object_detection/legacy/train.py --logtostderr /
                                                           --train_dir=./training_demo/training /
                                                           --pipeline_config_path=./training_demo/training/ssdlite_mobilenet_v2_coco.config
```

Рисунок 3.9 – Скрипт для запуска обучения

После обучения нужно конвертировать полученные результаты в frozen graph, который затем можно будет использовать в мобильном приложении. Конвертация обученной модели представлена на рисунке 3.10.

```
#!/bin/bash

python /content/models/research/object_detection/export_inference_graph.py --input_type image_tensor \
                                                                            --pipeline_config_path /content/training_demo/training/ssdlite_mobilenet_v2_coco.config \
                                                                            --trained_checkpoint_prefix /content/training_demo/training/model.ckpt-07070 \
                                                                             --output_directory /content/training_demo/training/output_inference_graph_v1.pb
```

Рисунок 3.10 – Конвертация полученного результата

Для использования полученных результатов в мобильном приложении нам конвертируем обученную модель в формат tflite, необходимый для запуска модели на мобильном устройстве. Сначала конвертируем полученный результат обучения в frozen graph который поддерживает конвертацию в tflite. Затем уже конвертируем в tflite. Конвертация модели в модель tflite представлена на рисунке 3.11.

```
#!/bin/bash

python /content/models/research/object_detection/export_tflite_ssd_graph.py --pipeline_config_path /content/training_demo/training/ssdlite_mobilenet_v2_coco.config \
                                                                             --trained_checkpoint_prefix /content/training_demo/training/model.ckpt-07070 \
                                                                             --output_directory /content/training_demo/training/output_inference_graph_tf_lite.pb
```

Рисунок 3.11 – Создание tflite

Чтобы запустить нашу модель в приложении, нужно заменить в assets файл с моделью на training_demo/training/model_q.tflite (переименовав её в detect.tflite) и файл с метками labelmap.txt.

После проделанных шагов и обучения модели был произведен запуск в утилите TensorBoard (рисунок 3.12):

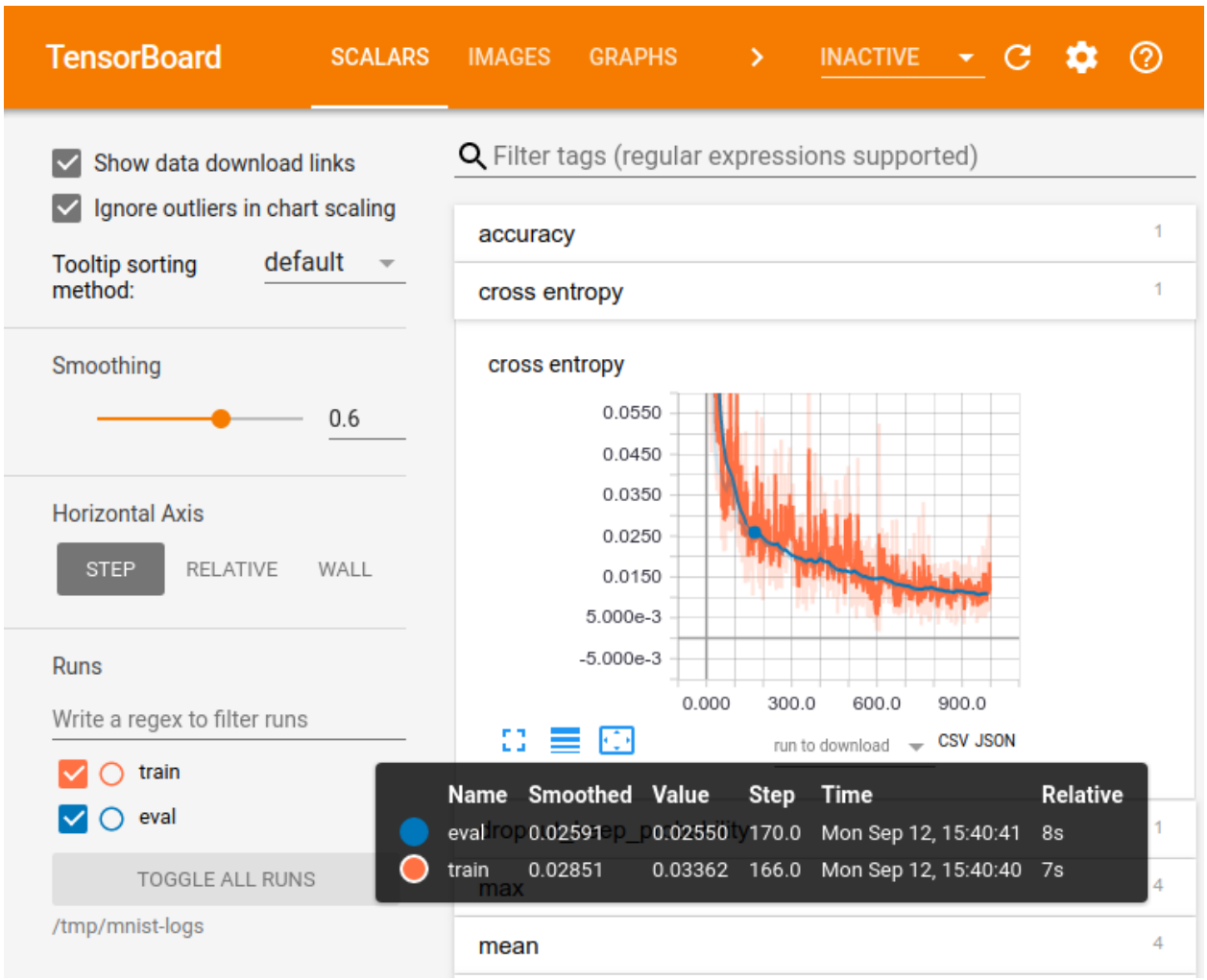

Рисунок 3.12 – Запуск модели в Tensoboard

В процессе тестирования модели было выявлено что точность модели машинного обучения составляет около 92%, что соответствует поставленной задаче. Следующим шагом будет разработка приложения для использования модели машинного обучения.

3.2 Разработка мобильного приложения для распознавания деталей оборудования с использованием обученной нейросети

Для разработки приложения была выбрана ранее платформа Android Studio и язык программирования Java. Разработчики мобильных приложений обычно взаимодействуют с типизированными объектами, такими как растровые изображения, или примитивами, такими как целые числа. Однако

интерпретатор TensorFlow Lite, который запускает модель машинного обучения на устройстве, использует тензоры в форме ByteBuffer, которые могут быть трудны для отладки. Библиотека поддержки Android TensorFlow Lite разработана для облегчения обработки ввода и вывода моделей TensorFlow Lite и упрощения использования интерпретатора TensorFlow Lite.

Перед переходом к разработке приложения, импортируем зависимости сборщика Gradle и других настроек. После того, как мы скопировали файл модели .tflite, нам нужно указать путь в каталоге ресурсов до нашей модели. Также укажем, что файл не должен быть сжат, и добавим библиотеку TensorFlow Lite в файл build.gradle модуля(рисунок 3.13):

```
android {
    aaptOptions {
        noCompress "tflite"
    }

}

dependencies {
    implementation 'org.tensorflow:tensorflow-lite:0.0.0-nightly'
    implementation 'org.tensorflow:tensorflow-lite-gpu:0.0.0-nightly'
    implementation 'org.tensorflow:tensorflow-lite-support:0.0.0-nightly'
}
```

Рисунок 3.13 – Импорт зависимостей и настроек

Для начала нам нужно получить данные с камеры. Наше мобильное приложение должно будет получать данные с камеры, используя функции, определенные в файле CameraActivity.java (Рисунок 3.14). Этот файл зависит от AndroidManifest.xml, чтобы установить ориентацию камеры.

CameraActivity также содержит код для захвата пользовательских настроек из пользовательского интерфейса и предоставления их другим классам с помощью удобных методов.

```
model = Model.valueOf(modelSpinner.getSelectedItem().toString().toUpperCase());
device = Device.valueOf(deviceSpinner.getSelectedItem().toString());
numThreads = Integer.parseInt(threadsTextView.getText().toString().trim());
```

Рисунок 3.14 – Класс CameraActivity.java

Основная часть логики будет реализована в классе Classifier. Файл Classifier.java содержит большую часть обработки ввода с камеры и выполнения логического вывода.

Существуют два подкласса файла, в ClassifierFloatMobileNet.java и ClassifierQuantizedMobileNet.java, чтобы продемонстрировать использование квантованных моделей. После появления библиотеки поддержки Android TensorFlow Lite эти подклассы в основном содержат настройки, а не логику обработки.

Класс Classifier реализует статический метод create, который используется для создания экземпляра соответствующего подкласса на основе предоставленного типа модели (квантуемой).

Для выполнения вывода нам нужно загрузить файл модели и создать экземпляр интерпретатора. Это происходит в конструкторе класса Classifier (рисунок 3.15) вместе с загрузкой списка меток класса. Информация о типе устройства и количестве потоков используется для настройки интерпретатора с помощью экземпляра Interpreter.Options, передаваемого в его конструктор.

```
protected Classifier(Activity activity, Device device, int numThreads) throws
    IOException {
  tfliteModel = FileUtil.loadMappedFile(activity, getModelPath());
  switch (device) {
    case NNAPI:
      nnApiDelegate = new NnApiDelegate();
      tfliteOptions.addDelegate(nnApiDelegate);
      break;
    case GPU:
      gpuDelegate = new GpuDelegate();
      tfliteOptions.addDelegate(gpuDelegate);
      break;
    case CPU:
      break;
  }
  tfliteOptions.setNumThreads(numThreads);
  tflite = new Interpreter(tfliteModel, tfliteOptions);
  labels = FileUtil.loadLabels(activity, getLabelPath());
```

Рисунок 3.15 – Реализация класса Classifier

Метод FileUtil.loadMappedFile выполняет предварительную загрузку и

сопоставление памяти файла модели, чтобы ускорить загрузку, возвращая MappedByteBuffer, содержащую модель.

MappedByteBuffer передается в конструктор Interpreter вместе с объектом Interpreter.Options. Этот объект можно использовать для настройки интерпретатора, например, путем установки количества потоков (.setNumThreads (1)) или включения NNAPI (.addDelegate (nnApiDelegate)).

Следующим шагом нам нужно осуществить предварительную обработку растрового изображения: в конструкторе Classifier мы получаем растровое изображение входной камеры, преобразуем его в формат TensorImage для эффективной обработки и предварительно обрабатываем его. Шаги показаны в приватном методе loadImage (рисунок 3.16).

```
private TensorImage loadImage(final Bitmap bitmap, int sensorOrientation) {
  image.load(bitmap);

  int cropSize = Math.min(bitmap.getWidth(), bitmap.getHeight());
  int numRoration = sensorOrientation / 90;
  ImageProcessor imageProcessor =
      new ImageProcessor.Builder()
          .add(new ResizeWithCropOrPadOp(cropSize, cropSize))
          .add(new ResizeOp(imageSizeX, imageSizeY, ResizeMethod.BILINEAR))
          .add(new Rot90Op(numRoration))
          .add(getPreprocessNormalizeOp())
          .build();
  return imageProcessor.process(inputImageBuffer);
}
```

Рисунок 3.16 – Реализация метода loadImage

Предварительная обработка в основном одинакова для квантованных и плавающих моделей с одним исключением: нормализация.

В ClassifierQuantizedMobileNet нормализация не требуется. Таким образом, параметры обозначения плавающей модели определяются на рисунке 3.17:

```
private static final float IMAGE_MEAN = 127.5f;
private static final float IMAGE_STD = 127.5f;
```

Рисунок 3.17 – Константы для плавающей модели

Теперь нам нужно реализовать вывод результатов нашей модели. Инициируем вывод TensorBuffer для вывода модели (рисунок 3.18):

```
private final TensorBuffer outputProbabilityBuffer;

int probabilityTensorIndex = 0;
int[] probabilityShape =
    tflite.getOutputTensor(probabilityTensorIndex).shape(); // {1, 1001}
DataType probabilityDataType =
    tflite.getOutputTensor(probabilityTensorIndex).dataType();

outputProbabilityBuffer =
    TensorBuffer.createFixedSize(probabilityShape, probabilityDataType);

probabilityProcessor =
    new TensorProcessor.Builder().add(getPostprocessNormalizeOp()).build();
```

Рисунок 3.18 – Реализация вывода модели

Вывод выполняется с использованием следующего в классе классификатора.

Вместо непосредственного вызова, используется метод Image. Он принимает растровое изображение и ориентацию сенсора, выполняет вывод и возвращает отсортированный список List of Recognition, каждый из которых соответствует метке. Метод вернет количество результатов, ограниченных значением MAX_RESULTS, которое по умолчанию равно 3.

Распознавание - это простой класс, который содержит информацию о конкретном результате распознавания, включая его заголовок и достоверность. Используя указанный метод нормализации постобработки, достоверность преобразуется в значение от 0 до 1 для данного класса, представленного изображением. Реализация данного метода показана на рисунке 3.19.

```
Map<String, Float> labeledProbability =
    new TensorLabel(labels,
        probabilityProcessor.process(outputProbabilityBuffer))
        .getMapWithFloatValue();

private static List<Recognition> getTopKProbability(
    Map<String, Float> labelProb) {

  PriorityQueue<Recognition> pq =
      new PriorityQueue<>(
          MAX_RESULTS,
          new Comparator<Recognition>() {
            @Override
            public int compare(Recognition lhs, Recognition rhs) {
              return Float.compare(rhs.getConfidence(), lhs.getConfidence());
            }
          });

  for (Map.Entry<String, Float> entry : labelProb.entrySet()) {
    pq.add(new Recognition("" + entry.getKey(), entry.getKey(),
                entry.getValue(), null));
  }

  final ArrayList<Recognition> recognitions = new ArrayList<>();
  int recognitionsSize = Math.min(pq.size(), MAX_RESULTS);
  for (int i = 0; i < recognitionsSize; ++i) {
    recognitions.add(pq.poll());
  }
  return recognitions;
}
```

Рисунок 3.19– Класс, отвечающий за распознавание

Классификатор вызывается, и результаты вывода отображаются функцией processImage () в ClassifierActivity.java.

ClassifierActivity - это подкласс CameraActivity, который содержит реализации методов, которые визуализируют изображение с камеры, выполняют классификацию и отображают результаты. Метод processImage () выполняет классификацию в фоновом потоке как можно быстрее, отображая информацию в потоке пользовательского интерфейса, чтобы избежать блокирования и создания задержки. Реализация данного подкласса находится на рисунке 3.20.

```
@Override
protected void processImage() {
  rgbFrameBitmap.setPixels(getRgbBytes(), 0, previewWidth, 0, 0, previewWidth,
      previewHeight);
  final int imageSizeX = classifier.getImageSizeX();
  final int imageSizeY = classifier.getImageSizeY();

  runInBackground(
      new Runnable() {
        @Override
        public void run() {
          if (classifier != null) {
            final long startTime = SystemClock.uptimeMillis();
            final List<Classifier.Recognition> results =
                classifier.recognizeImage(rgbFrameBitmap, sensorOrientation);
            lastProcessingTimeMs = SystemClock.uptimeMillis() - startTime;
            LOGGER.v("Detect: %s", results);

            runOnUiThread(
                new Runnable() {
                  @Override
                  public void run() {
                    showResultsInBottomSheet(results);
                    showFrameInfo(previewWidth + "x" + previewHeight);
                    showCropInfo(imageSizeX + "x" + imageSizeY);
                    showCameraResolution(imageSizeX + "x" + imageSizeY);
                    showRotationInfo(String.valueOf(sensorOrientation));
                    showInference(lastProcessingTimeMs + "ms");
                  }
                });
          }
          readyForNextImage();
        }
      });
}
```

Рисунок 3.20 – Вывод результатов

Другая важная роль ClassifierActivity заключается в определении пользовательских предпочтений и создании экземпляра соответствующим образом настроенного подкласса Classifier. Это происходит, когда начинается подача видео (через onPreviewSizeChosen()) и когда параметры изменяются в пользовательском интерфейсе. Создание классификатора представлено на рисунке 3.21.

```
private void recreateClassifier(Model model, Device device, int numThreads) {
  if (classifier != null) {
    LOGGER.d("Closing classifier.");
    classifier.close();
    classifier = null;
  }
  if (device == Device.GPU && model == Model.QUANTIZED) {
    LOGGER.d("Not creating classifier: GPU doesn't support quantized models.");
    runOnUiThread(
        () -> {
          Toast.makeText(this, "GPU does not yet supported quantized models.",
              Toast.LENGTH_LONG)
              .show();
        });
    return;
  }
  try {
    LOGGER.d(
        "Creating classifier (model=%s, device=%s, numThreads=%d)", model,
        device, numThreads);
    classifier = Classifier.create(this, model, device, numThreads);
  } catch (IOException e) {
    LOGGER.e(e, "Failed to create classifier.");
  }
}
```

Рисунок 3.22 – Создание экземпляра класса

После выполнения всех шагов и реализации всех классов было разработано мобильное приложение, которое может запускаться на мобильных устройствах под управлением операционной системы Android. Теперь нам провести тестирование и определить, насколько качественно было произведено обучение, а также разработка приложения.

3.3 Тестирование мобильного приложения и нейронной сети

После обучения модели и реализации приложения был произведен запуск на нескольких устройствах системы Android. В процессе тестирования было выявлено, что ошибка распознавания не превышала 10%, но на устройствах, которые имеют меньше четырех логических ядер распознавание занимает около 300мс, против 20мс на устройствах с четырьмя логическими ядрами. Скриншот тестирования приложения представлен на рисунке 3.23х.

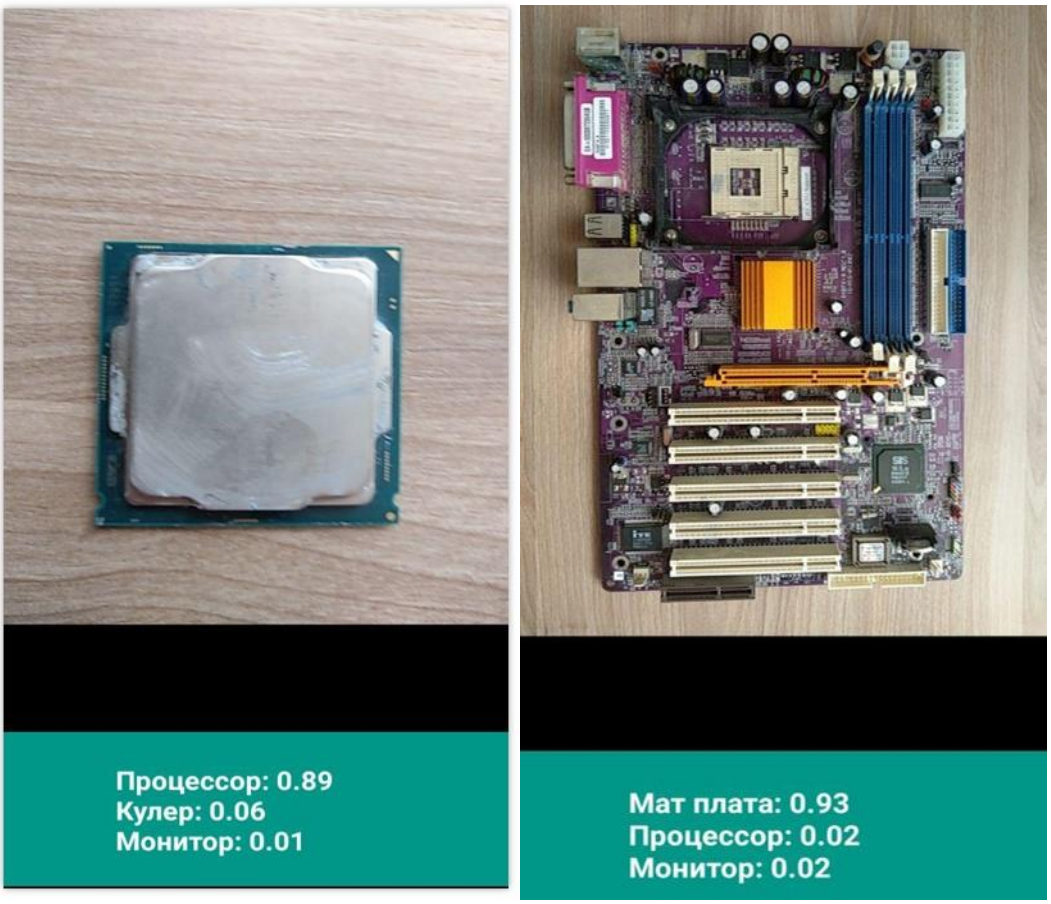

Рисунок 3.23 – Скриншот тестирования приложения

Как можно видеть на данном скриншоте, наше приложение смогло с высокой точностью определить, что находится на изображении.

Таким образом, можно сделать вывод что наше приложение было реализована успешно и удовлетворяет требованиям. Данное приложение показывает как с использованием нейронных сетей можно создавать приложения, которые будут реализованы и работать на мобильных приложениях.

Заключение

Выполненная работа посвящена исследованию свёрточных нейронных сетей и использованию их в мобильном приложении. Цель работы определила её основное направление – обучение модели распознаванию деталей оборудования, а также разработка приложения для комфортного использования обученной модели.

Для этого были исследованы модели машинного обучения, которые бы удовлетворяли нашим требованиям, а точнее: иметь небольшой вес; высокую точность; небольшие требования к вычислительной мощности.

В процессе исследований было выявлена, что на данный момент наиболее эффективно подходит нейронная сеть MobileNetV2, так как распознавание происходит даже с 5 FPS, а вес составляет всего пару мегабайт, что позволяет хранить обученную модель на мобильном устройстве.

Обучение проводилось с использованием языка Python и платформы TensorFlow. Для тренировки был подготовлен небольшой датасет в размере около 200 изображений. После обучения, полученную модель нужно было конвертировать в формат tflite для использования модели в мобильном приложении. Было разработано приложения на языке Java. Благодаря хорошо задокументированному API от TensorFlow, приложение получилось простым в реализации, а также без специальных технических параметров, необходимых для запуска. Оно состоит из нескольких классов, которые отвечают за считывание изображения с камеры, передача изображения в классификатор и вывода классификатором результата.

Протестированное приложение с обученной моделью показало неплохие данные. Приложение запускалось на трёх разных устройствах. В целом, приложение работало довольно плавно, ошибка не превышала 10%, но время на распознавание объекта составило порядка 20мс на устройствах с 4 логическими ядрами, а на устройствах с двумя ядрами распознавание

составило 300мс, без превышения процента ошибки.

Список используемых источников

1. Бурков А. Машинное обучение без лишних слов / Андрей Бурков - Питер СПб, 2020. – 192 с.
2. Вьюгин В. Математические основы машинного обучения и прогнозирования / Владимир Вьюгин. - МЦНМО, 2014. - 304 с
3. Документация по библиотеке машинного обучения TensorFlow 2.0 // TensorFlow [Электронный ресурс]: официальный сайт TensorFlow. URL: https://www.tensorflow.org/versions/r2.0/api_docs/python/ (дата обращения 17.03.2020).
4. Документация по языку программирования Python // Python [Электронный ресурс]: официальный сайт Python. URL: https://www.python.org/doc/ (дата обращения 12.03.2020)
5. Abu-Mostafa Y. Learning From Data / Yaser S. Abu-Mostafa, Malik Magdon-Ismail, Hsuan-Tien Lin – AMLBook. – 2012.-Jan. -C. 213.
6. Al-Azawi M. A. N. Neural Network Based Automatic Traffic Signs Recognition //International Journal of Digital Information and Wireless Communications (IJDIWC). - 2011. - Т. 1. - №. 4. - С. 753-766.
7. Bahlmann C. et al. A system for traffic sign detection, tracking, and recognition using color, shape, and motion information //Intelligent Vehicles Symposium, 2005. Proceedings. IEEE. - IEEE, 2005. - С. 255-260.
8. Baldi P. Autoencoders, Unsupervised Learning, and Deep Architectures //ICML Unsupervised and Transfer Learning. - 2012. - Т. 27. - С. 37-50.
9. Bastien F. et al. Theano: new features and speed improvements //arXiv preprint arXiv:1211.5590. - 2012.
10. Bergstra J. et al. Theano: A CPU and GPU math compiler in Python //Proc. 9th Python in Science Conf. - 2010. - С. 1-7.
11. Bishop C. Pattern Recognition and Machine Learning (Information Science and Statistics) / Christopher M. Bishop - Springer-Verlag New York Inc. - 2007.-Feb. -C. 738.

12. Broggi A. et al. Real time road signs recognition //Intelligent Vehicles Symposium, 2007 IEEE. - IEEE, 2007. - С. 981-986.

13. Canny J. A computational approach to edge detection //Pattern Analysis and Machine Intelligence, IEEE Transactions on. - 1986. - №. 6. - С. 679-698.

14. Chollet F. Deep Learning with Python / Francois Chollet - Manning Pubns Co. – 2017.-Nov. -C. 361.

15. Christian Szegedy. Scalable, High-Quality Object Detection, 2015 // arXiv [Электронный ресурс]: открытый архив научных статей. URL: https://arxiv.org/abs/1412.1441 (дата обращения 10.04.2020).

16. Cireşan D. C. et al. Deep big multilayer perceptrons for digit recognition //Neural Networks: Tricks of the Trade. - Springer Berlin Heidelberg, 2012. - С. 581-598.

17. Ciresan D. et al. A committee of neural networks for traffic sign classification //Neural Networks (IJCNN), The 2011 International Joint Conference on. - IEEE, 2011. - С. 1918-1921.

18. Ciresan D., Meier U., Schmidhuber J. Multi-column deep neural networks for image classification //Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. - IEEE, 2012. - С. 3642-3649.

19. Daugman J. G. Complete discrete 2-D Gabor transforms by neural networks for image analysis and compression //Acoustics, Speech and Signal Processing, IEEE Transactions on. - 1988. - Т. 36. - №. 7. - С. 1169-1179.

20. Eddison L. Python Machine Learning: A Technical Approach To Python Machine Learning For Beginners / Leonard Eddison - CreateSpace Independent Publishing Platform. – 2018.-Mar. -C. 292.

21. Gao X. W. et al. Recognition of traffic signs based on their colour and shape features extracted using human vision models //Journal of Visual Communication and Image Representation. - 2006. - Т. 17. - №. 4. - С. 675-685.

22. Geron A. Hands-On Machine Learning with Scikit-Learn and TensorFlow / Aurelien Geron – O`Reilly Media. – 2017.-Mar. -C. 566.

23. Goodfellow I. Deep Learning (Adaptive Computation and Machine Learning series) / Ian Goodfellow, Yoshua Bengio, Aaron Courville - The MIT Press. -2016.-Nov. -C. 800.

24. Goodfellow I. J. et al. Pylearn2: a machine learning research library //arXiv preprint arXiv:1308.4214. - 2013.

25. Guido S. Introduction to Machine Learning with Python: A Guide for Data Scientists / Sarah Guido, August Mueller - O`Reilly Media. – 2016.-May. -C. 400.

26. Mitchell T. Machine Learning / Tom Mitchell – Mc Graw Hill India. - 2017.– Mar. – C. 432.

27. Raschka S. Python Machine Learning / Sebastian Raschka - Packt Publishing. – 2015.-Sep. -C. 454.

28. Rashid T. Make Your Own Neural Network / Tariq Rashid - CreateSpace Independent Publishing Platform. – 2016. – C. 222.

29. Rojas R. Neural Networks: A Systematic Introduction / Raul Rojas, Peter Varga - Springer Berlin Heidelberg. – 1996.-Jul. -C. 522.

30. Russell S. Artificial Intelligence: Pearson New International Edition: A Modern Approach / Stuart Russel, Norvig Peter – Pearson. – 2013.-Aug. -C.1104.

31. Ryo Takahashi. Data Augmentation using Random Image Cropping and Patching for Deep CNNs, 2019 // arXiv [Электронный ресурс]: открытый архив научных статей. URL: https://arxiv.org/abs/1811.09030 (дата обращения 10.04.2020).

32. Shalev-Shwarthz S. Understanding Machine Learning: From Theory to Algorithms / Shai Shalev-Shwartz, Shai Ben-David - Cambridge University Press. -2014.-May. -C. 415.

33. Shukla N. Machine Learning with TensorFlow / Nishant Shukla – Manning Publication. – 2018.-Jan. -C. 272.

34. Wei Liu. SSD: Single Shot MultiBox Detector, 2016 // arXiv [Электронный ресурс]: открытый архив научных статей. URL:

https://arxiv.org/abs/1512.02325 (дата обращения 10.04.2020).