МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное   
учреждение высшего образования   
«Южный федеральный университет»

Институт математики, механики   
и компьютерных наук им. И. И. Воровича

Кондратьева Аэлина Максимовна

РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ РАСПОЗНАВАНИЯ ДОСТОПРИМЕЧАТЕЛЬНОСТЕЙ НА ИЗОБРАЖЕНИИ СРЕДСТВАМИ МАШИННОГО ОБУЧЕНИЯ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА  
по направлению подготовки  
02.04.02 – Фундаментальная информатика и информационные технологии,   
направленность программы  
«Разработка мобильных приложений и компьютерных игр»

**Научный руководитель** –   
доц., к. т. н. Демяненко Яна Михайловна

**Рецензент** –   
доц., к. ф.-м. н. Абрамян Анна Владимировна

Допущено к защите:  
руководитель   
образовательной программы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Демяненко Я. М.

Ростов-на-Дону – 2024

Оглавление

[Введение 3](#_Toc168247989)

[1. Постановка задачи 5](#_Toc168247990)

[2. Анализ существующих решений 6](#_Toc168247991)

[2.1. Модель от Mail.ru Group 6](#_Toc168247992)

[2.2. Модель Supporting large-scale image recognition with out-of-domain samples 9](#_Toc168247993)

[2.3. Модель 2nd Place Solution to Google Landmark Recognition Competition 2020 10](#_Toc168247994)

[2.4. Модель Dolg: Single-stage image retrieval with deep orthogonal fusion of local and global features 12](#_Toc168247995)

[3. Данные Google Landmarks Dataset 14](#_Toc168247996)

[4. Создание RndLandmarks 19](#_Toc168247997)

[5. Обучение c помощью фреймворка PyTorch 22](#_Toc168247998)

[6. Экспортирование в TFLite 25](#_Toc168247999)

[7. Библиотека TensorFlow Lite Model Maker 27](#_Toc168248000)

[8. Обучение и эксперименты TensorFlow Lite Model Maker 33](#_Toc168248001)

[9. Разработка мобильного приложения 41](#_Toc168248002)

[9.1. Используемые инструменты и технологии 41](#_Toc168248003)

[9.2. Работа с данными RnDLandmarks 44](#_Toc168248004)

[9.3. Главный экран 46](#_Toc168248005)

[9.4. Экран демонстрации 48](#_Toc168248006)

[9.5. Экран карты 50](#_Toc168248007)

[Заключение 52](#_Toc168248008)

[Литература 53](#_Toc168248009)

# Введение

В наши дни индустрия туризма является крупным экономическим сектором. Ни одно путешествие не обходится без списка интересных мест, посещения достопримечательностей и создания не маленького количества фотографий. На каждом из этих этапов возникает потребность в распознавании достопримечательности по фотографии. Не редко мы восхищаемся кадрами из социальных сетей и хотим увидеть это своими глазами. К сожалению, не всегда удается узнать, где именно был отснят материал, чтобы добавить новое место в свой чек-лист. Гуляя по новому городу без гида нужно искать или спрашивать название каждого памятника архитектуры, который тебя заинтересовал. Таким образом распознавание достопримечательности на фотографиях является актуальной проблемой.

Существует несколько подходов разработки системы распознавания достопримечательностей. Один из них – создать базу, содержащую эталонное изображение для каждой достопримечательности и решать задачу поиска изображения, похожего на данное. Однако, фотографии могут иметь различный ракурс (рис. 0.1), погодные условия и время суток, следовательно быть совершенно не похожими. Другой подход – поставить задачу классификации изображений.

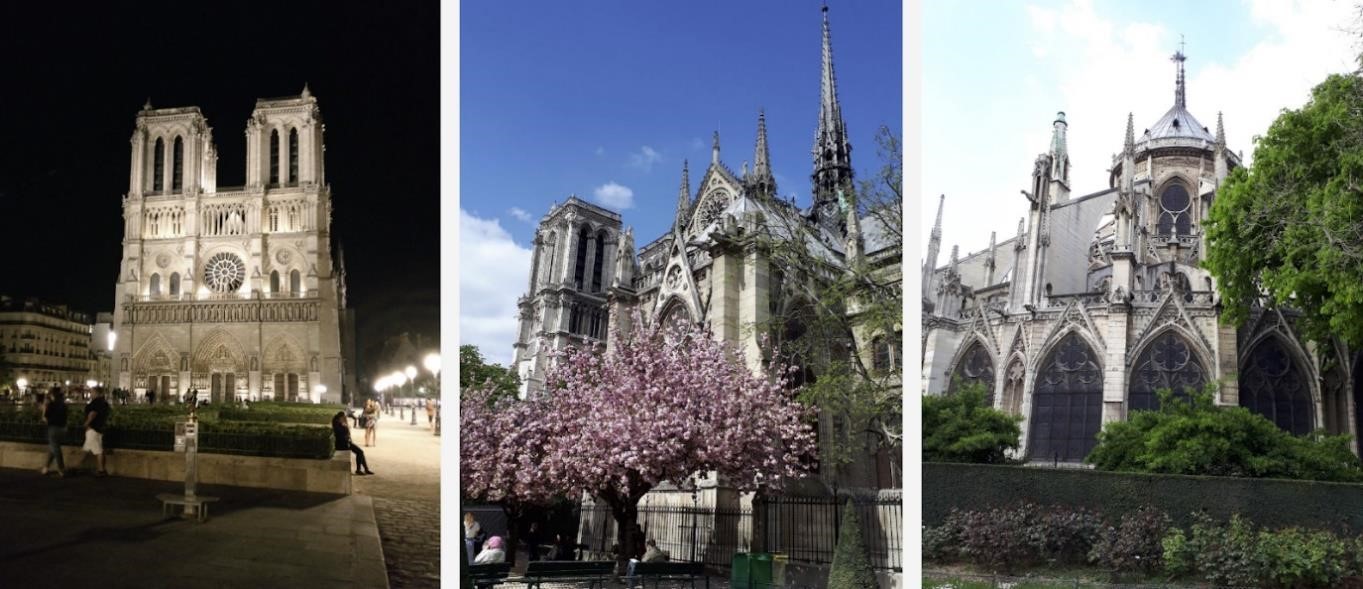


Рис. 0.1. Нотр-Дам-де-Пари с разных ракурсов

Задача является непростой по следующим причинам. Во-первых, достопримечательности бывают весьма разнообразными: здания, памятники, набережные, парки озера (рис. 0.2). Во-вторых, многие пользователи отключают функцию определения геоданных на своих смартфонах, поэтому нужно уметь обходиться только изображением.



Рис. 0.2. Кейсария в Израиле, Английский парк в Мюнхене.

Существует множество алгоритмов компьютерного зрения для распознавания объектов. Примитивы Хаара, метод Виолы Джонса, алгоритм Хафа и его модификации и так далее. Однако современный решения все больше используют машинное обучение и сверточные нейронные сети.

# Постановка задачи

Цель: создание мобильного приложения под платформу Android, распознающего достопримечательность на фото, средствами машинного обучения. Приложение должно работать без доступа к геологическим данным и сети интернет.

Для этого необходимо решить следующие задачи:

* проанализировать существующие методы решения задачи классификации достопримечательностей, основанные на машинном обучении;
* выбрать оптимальную архитектуру нейросети;
* обучить и протестировать модель;
* разработать мобильное приложение.

# Анализ существующих решений

## Модель от Mail.ru Group

В 2019 году решение этой задачи представили исследователи Mail.ru Group (рис. 2.1). В качестве архитектуры сверточной нейронной сети для извлечения признаков из изображения они взяли ResNet50 и удвоили количество сверток во внутреннем bottleneck-блоке. Технология residual blocks позволяет сети самой решать, на сколько глубокой она должна быть. Был применен необычный подход предварительного обучения – сначала сеть обучали на базе для распознавания сцен, а затем на главном наборе данных, обобщая низкоуровневые признаки первого этапа. Весь набор данных разделен на 4 части по географическому признаку. Обучение происходит в 3 этапа с разными learning rate. Полученные признаки перерабатываются в эмбеддинги с помощью полносвязного слоя и Batch norm. Эмбеддинги поступают на вход классификатору (полносвязный слой), а затем в функцию потерь. Для N достопримечательностей используется CenterLoss, а для класса «не достопримечательность» SoftmaxLoss, так как сюда входит огромное разнообразие изображений и нет смысла их центрировать.

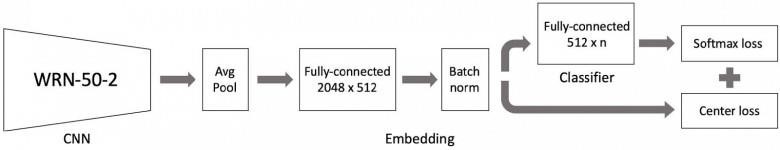


Рис. 2.1. Архитектура модели от Mail.ru Group

Для каждого класса достопримечательностей вычисляется центроид – среднее эмбеддингов изображений данного класса. Вычислять только один центроид неэффективно, так как изображения одной и той же достопримечательности могут существенно отличаться из-за ракурса, крупного плана или времени суток. Например, для класса Зимний дворец в базе есть фото с Дворцовой площадью и фото с аркой Главного штаба. А для класса колокольня Белфорт в база есть разноплановые изображения. Поэтому необходимо вычислить несколько центроидов для каждого класса (рис. 2.2).

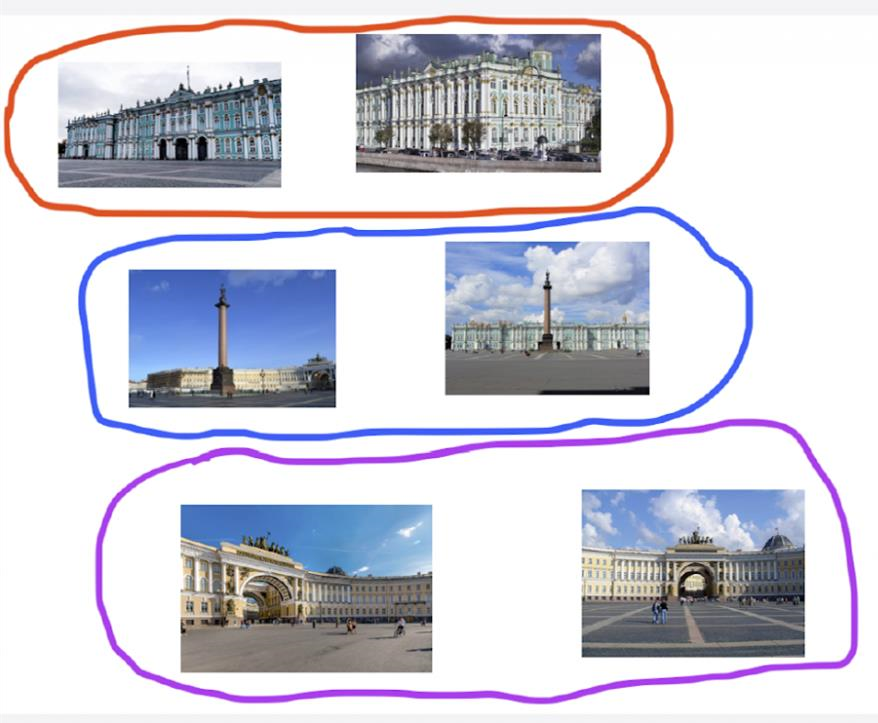


Рис. 2.2. Примеры различных центроидов для одного класса

Для определения количества центроидов в кластере и множеств изображений, для вычисления каждого центроида, изображения из каждого класса подвергаются кластеризации. Затем отбрасываются кластеры, содержащие менее чем 50 изображений. По оставшимся кластерам вычисляются центроиды.

Для распознавания достопримечательности на фото исходное изображение прогоняется через нейронную сеть. Полученный эмбеддинг сравнивается с центроидами с помощью скалярного произведения. Если разница меньше некоторого порога, то на фото нет достопримечательности. В ином случае правильность предсказания проверяется геоданными. При их отсутствии происходит сравнение с эмбедингом эталонного изображения класса, определенным для каждой достопримечательности.

Первые тесты модели показали плохие результаты. Так как Mail.ru Group нацелены главным образом на поиск достопримечательностей в хранилищах своих клиентов, а люди зачастую фотографируют себя на фоне чего-либо, тестовые фотографии имели значительное отличие от обучающей выборки (рис. 2.3).

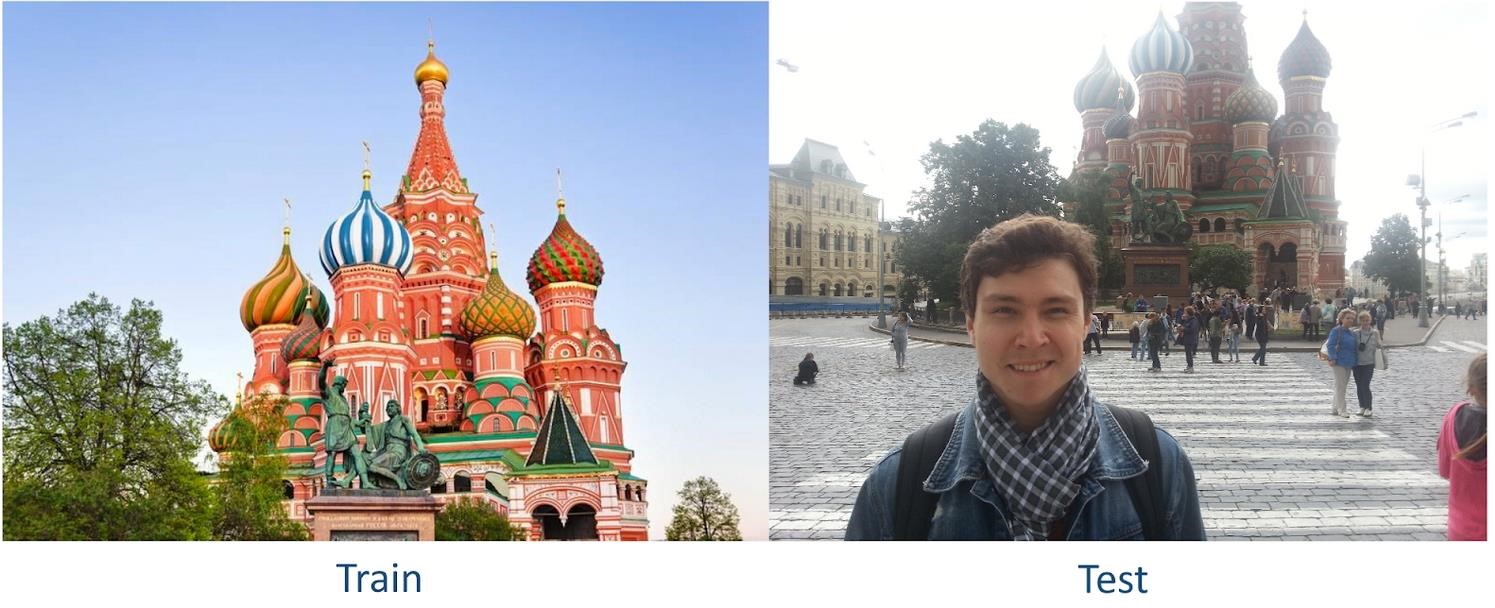


Рис. 2.3. Принципиальное отличает тестовой выборки от обучающей

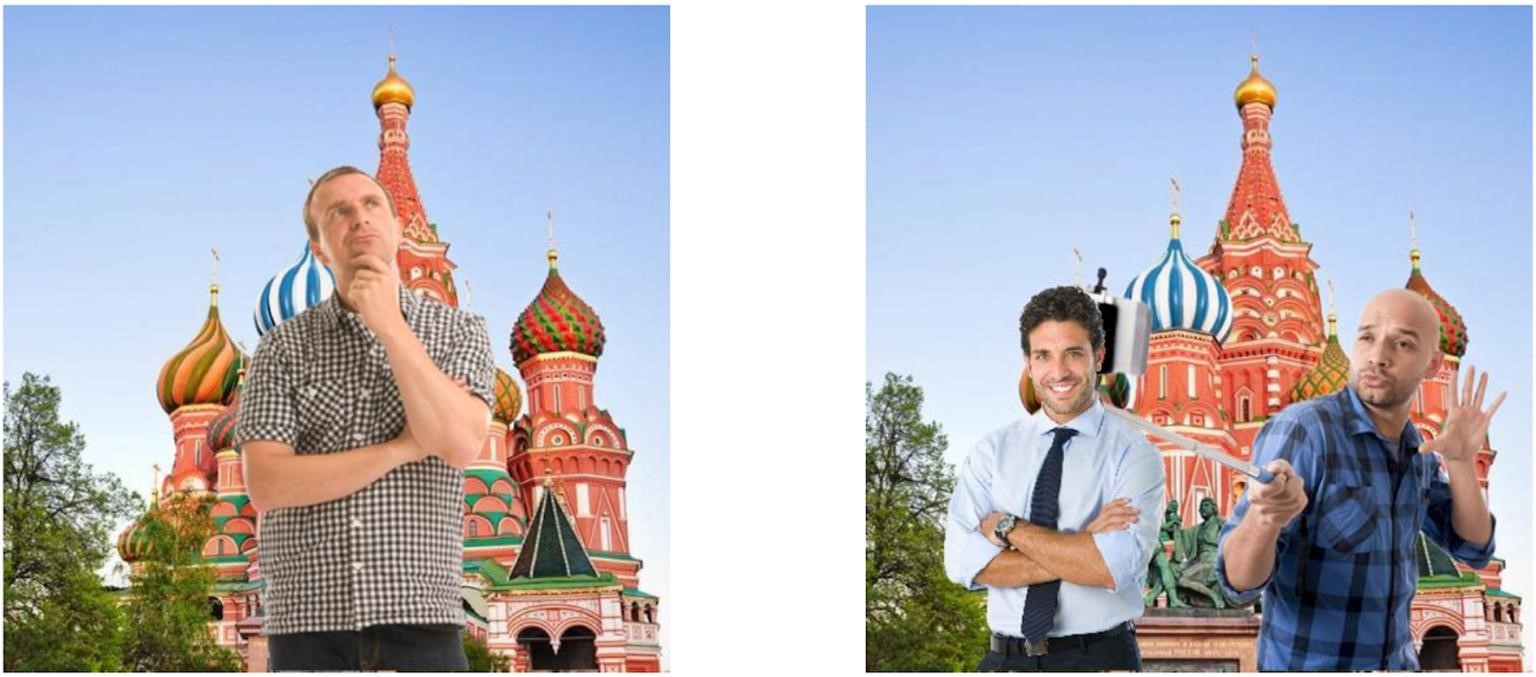


Рис. 2.4. Новая обучающая база

Тогда исследователи модифицировали свою базу данных, добавив туда людей, в прямом смысле этого выражения (рис. 2.4). Это помогло повысить качество предсказания на тестовой выборке.

## Модель Supporting large-scale image recognition with out-of-domain samples

В 2020 году на сайте Kaggle проходило соревнование, посвященное распознаванию достопримечательности по фотографии. В качестве датасета был использован GLDv2. В ходе данной научно-исследовательской работы были проанализированы решения победителей. Кандидаты для подробного изучения отбирались по принципу непохожести, для того чтобы рассмотреть различные подходы к решению данной задачи.

Первым претендентом на рассмотрение является решение победителей 2020 года. Supporting large-scale image recognition with out-of-domain samples полностью отказались от локальных дескрипторов. Их предсказательная система использует лишь глобальные дескрипторы. Во-первых, расчет ключевых точек достаточно затратная процедура, как по времени, так и по вычислительной мощности. Во-вторых, после повышения эффективности глобальных дескрипторов преимущества локальных стали не столь велики. Таким образом, необходимость в дорогостоящем расчете локальных дескрипторов отпала.

Все входные изображения подвергаются предобработке. Для более эффективного обучения изображения масштабируются и обрезаются, расширяя обучающий датасет. При предсказании изображение нормализуется с помощью «mean and standard devation of imagenet dataset». В работе упоминают, что обучение на более полном датасете 2019 года не дает улчшения качества предсказаний. Эта информация позволяет нам выбрать для своих исследований только последнюю версию GLD.

Основной магистралью для извлечения фитч (глобальных дескрипторов) из изображения выступает ансамбль из нескольких сверточных нейронных сетей, обрабатывающих входное изображение независимо. Ансамбль состоит из 7 моделей: три сети SE-ResNeXt101, две сети EfficientNet B3, ResNet152 и Res2Net101 по одной сети.

Предварительно каждая из нейросетей подвергается L2-нормализации, а в конце каждый разряд эмбедингов подвергается квантильной трасформации. Затем идет слой GeM-pooling и перешеек, состоящий из Linear(512) + Batch-Norm + PReLU слоев. На выходе получается вектор в 512-мерном пространстве, описывающий изображение.

Полученные эмбединги используются в двух способах предсказания класса. Первый, это использовать arc margin head (найти среднее описание класса в 512 мерном пространстве и сравнивать близость описания входного изображения с этим центроидом, но все это происходит на поверхности многомерной сферы). Второй способ – стандартный метод k ближайших соседей на основе косинусной схожести. Для оценки достоверности предсказаний на тестовом наборе используется метрика GAP (Global Average Precision).

Обучение длится в течение 10 эпох с помощью косинусного планировщика отжига с одной эпохой прогрева. Используется SGD оптимизатор со следующими глобальными параметрами: скорость обучения - 0.05, функция потерь – arcface loss, вес каждой модели – 0.0001.

## Модель 2nd Place Solution to Google Landmark Recognition Competition 2020

Следующим в подробное рассмотрение попало решение, занявшее в 2020 году первое место на публичной доске лидеров, но уступившее одну позицию в приватной доске.

Публичная доска лидеров составляется на основе предоставленных соперникам данных и изменяется в ходе предоставления новых решений и результатов. Приватная доска публикуется после закрытия соревнований на основе измерения точности модели на новых данных.

Итак, что же мы имеем. Модель состоит из четырех сверточных нейронных сетей: EfficientNet B5, EfficientNet B6, EfficientNet B7, ResNet 152. За ними следует GeM pooling слой и два полносвязных слоя. На вход последнему слою подаются 512-ти размерные эмбединги, а на выходе получаются вероятности принадлежности к 81 000 классов. Данная архитектура тоже отказалась от локальных дескрипторов. В качестве функции потерь используется только arcmargin с масштабом 30 и отступом 0.3. При этом авторы утверждают, что 200 000 изображений вполне достаточно для валидации, вовсе не обязательно использовать весь тестовый набор данных.

Обучение происходило в 5 этапов, используя изображения различных размеров в тренировочном датасете, с фиксированными скоростью обучения – 0.01 и снижением весов – 0.00001.

1. 6 эпох, изображения из тестового датасета размером 448x448
2. 4 эпох, все изображения из тестового датасета – 3.2 млн
3. 6 эпох, изображения из тестового датасета размером 512x512
4. 3 эпох, изображения из тестового датасета размером 640x640
5. 3 эпох, изображения из тестового датасета размером 736x736

Такой подход позволяет модели быть устойчивой к разномасштабным изображениям. Также для борьбы с внутриклассовой изменчивость было создано расширения датасета на основе следующих параметров: случайного масштабирования, изменения яркости, контрастности и цветопередачи, обрезка, повороты и зеркальные отображения.

Решение второго места 2020 года еще раз подтверждает эффективность использования ансамбля моделей. В статье подробно рассказывается о стратегии обучения, но остается множество вопросов по поводу самой модели.

## Модель Dolg: Single-stage image retrieval with deep orthogonal fusion of local and global features

Параллельно с соревнованиями по распознаванию достопримечательностей проходило соревнование по восстановлению или поиску достопримечательностей. Задачей на этом соревновании было найти в датасете как можно большей изображений того же класса, что и исходное изображение (его класс неизвестен). По факту это та же подзадача расчета схожести. А значит, из решений можно почерпнуть идеи для нашей задачи. Третьим кандидатом на подробное рассмотрение стала работа под названием DOLG: Single-Stage Image Retrieval with Deep Orthogonal Fusion of Local and Global Features. Ключевым отличаем данного подхода является использование и локальных и глобальных дескрипторов. Причем авторы отказываются от последовательной обработки: сначала выбрать из базы кандидатов на основе глобального сходства, а затем переранжировать изображения с помощью локальных дескрипторов. Вместо этого используется одноэтапная модель, объединяющая оба вида фитч в комактный эмбединг.

Глобальные фитчи извлекаются так же, как и вышеупомянутых работах – глубокая сверточная нейронная сеть с ArcFace функцией потерь. Локальные фитчи извлекаются методологией, похожей на поиск SIFT дескрипторов, с помощью многоуровневых сверточных слоев. Также используются карты внимания. Затем оба дескриптора поступаю в ортогональную модель объединения.

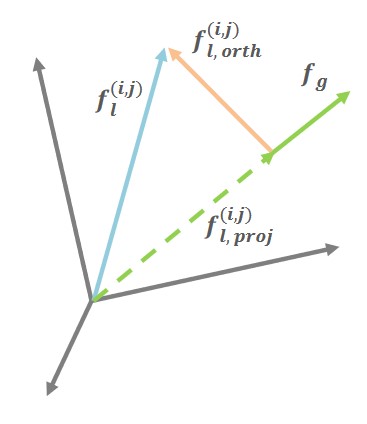


Рис. 2.5. Ортогональная модель объединения фитч

Обозначим вектор, описывающий локальные признаки через 𝑓𝑙, а вектор, описывающий глобальные признаки через 𝑓𝑔. 𝑓𝑝𝑟𝑜𝑗 – это проекция вектор 𝑓𝑙 На 𝑓𝑔, а 𝑓𝑜𝑟𝑡ℎ – это перпендикуляр, опущенный из конца 𝑓𝑙 на 𝑓𝑔. В двумерном пространстве это бы выглядело следующим образом. Математически же, эти величины можно рассчитать и в многомерном пространстве. Затем происходит объединение вектора глобальных признаков 𝑓𝑔 и ортогонального вектора 𝑓𝑜𝑟𝑡ℎ. Последним шагом является полносвязный сверточный слой, приводящий итоговый эмбединиг к размерности 512.

# Данные Google Landmarks Dataset

Качественный датасет является важнейшим шагом на пути к успешному обучению модели. Поэтому необходимо тщательно выбирать набор данных, с которым ты будешь работать.

Задача распознавания достопримечательностей имеет несколько особенностей, которые необходимо учитывать. Во-первых, это задача распознавания крупномасштабных объектов на изображении. Во-вторых, изображения могут быть сильно зашумлены ввиду погодных условий. В-третьих, у достопримечательностей рекордно высокая внутриклассовая изменчивость. Это обусловлено тем, что к одному и тому же классу должны быть отнесены объекты, изображенные крупным планом, расположенные на заднем плане и даже виды с достопримечательностей. Также на разнообразие изображений влияет освещение, время суток и угол обзора. Ну и конечно, модель не должна называть Пизанской башней все светлые цилиндры с таким же углом наклона.

Отличной базой специфических наборов данных для машинного обучения и компьютерного зрения является Kaggle. Датасетом, учитывающим все эти особенности, является Google Landmarks Dataset v2. В нем содержится более 5 млн изображений. Изображения взяты из Wikimedia Commons – крупнейшей в мире коллекции достопримечательностей. Google Landmark Recognition 2020 старается собрать как можно больше изображений, что может привести к искажению данных. В отличай от своего предшественника, Google Landmark Recognition 2021 выбирает более востребованные достопримечательности, ориентируясь на статистику Google Maps.

Набор данных состоит из трех частей: Train, Index, Test. Обучающий набор включает в себя 4 млн изображений, индексный набор – 761 сотню изображений, а тестовый набор – 117 сотен изображений. Index можно использовать для обучения. Его главное отличие от набора Train – отсутствие авторских прав, фотографии являются общественным достоянием. Изображения из набора данных Index и Test имеют лицензию CC0. Это означает, что произведение распространяется свободно, и каждый может делать с ним что угодно. Ему не нужно спрашивать разрешение у первоначального создателя, он может использовать продукт в коммерческих целях и может создавать производные работы. Изображения из набора данных Train, самого большого, имеют лицензию CC-BY. Лицензия CC-BY требует, чтобы первоначальный владелец авторских прав был указан в качестве автора, но в остальном работа доступна для всех, кто её использует. Любой может использовать её в коммерческих целях или изменять.

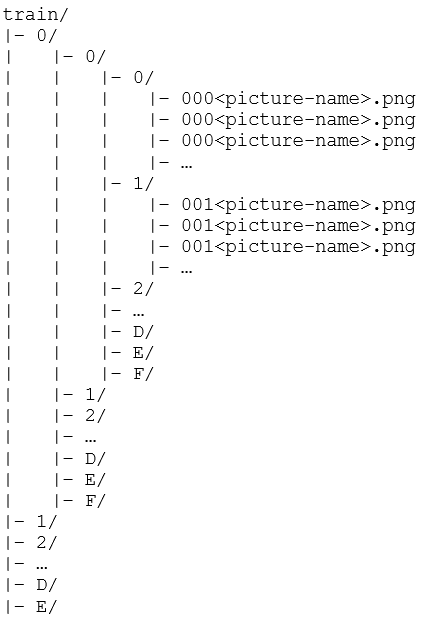


Рис 3.1 Структура файлов Google Landmarks Dataset v2

Набор данных Google Landmarks Dataset v2 имеет нестандартную для задачи классификации структуру файлов (рис). Вместо папок с классами, директория train содержит в себе семнадцать директорий (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f ). Каждая из которых содержит семнадцать поддиректорий (второй уровень вложенности), с еще семнадцатью поддиректориями. И только на третьем уровне вложенности хранятся RGB изображения. Принадлежность картинок к классам описана в train.csv таблице. Первые три знака в имени изображения описывают путь до изображения по трем уровням вложенности папок. Аналогично устроены папки test и index.

Для работы с Google Landmarks Dataset v2 был создан класс LandmarkDataset (листинг 3.1). Входными данными для конструктора класса являются путь до папки train, test или index и путь до соответствующей scv таблицы. Метод \_\_len\_\_(self) возвращает количество уникальных классов, в датасете их боле 81 тысячи. Методы transform\_\*() производят различную предварительную обработку изображений (обрезание, изменение размера, нормализация). Метод \_\_getitem\_\_(self,index) предоставляет возможность итерации по датафрейму, последовательно возвращая пары изображение — класс, что позволяет классу работать со стандартной библиотекой torch.utils.DataLoader.

Листинг 3.1 Класс десериализации данных Google Landmarks Dataset v2

data class LandmarkEntity(  
 val id: Int,  
 val name: String?,  
 val geo: LatLng?,  
 val url: String?,  
 val description: String?,  
):Parcelable {  
 constructor(parcel: Parcel) : this(  
 parcel.readInt(),  
 parcel.readString(),  
 parcel.readParcelable(LatLng::class.*java*.*classLoader*),  
 parcel.readString(),  
 parcel.readString()  
 ) {  
 }

Изображения будут подаваться на конвейер обучения в том порядке, в котором они перечислены в csv таблице. Так как в таблице классы не упорядочены, в один батч размера 32 могут попасть картинки из 32 различных классов, что негативно сказывается на процессе обучения. Появилась необходимость упорядочить scv файл, чтобы у нейронных сетей была возможность последовательно изучать и запоминать особенности каждого класса.

С целью получения доступа к большим вычислительным ресурсам, чем личный пк, процесс обучения и валидации происходит в облачной среде разработки и исполнения Kaggle Notebook. Это позволяет не загружать огромный набор данных на гугл диск, а просто подключить его как библиотеку в интерактивном меню. Время непрерывной сессии в Kaggle Notebook ограничено. К сожалению, даже одна эпоха обучения на таком большом датасете (81 тысяча классов) не вписывается в 24 часа. Было принято решение уменьшить количество классов.

При более близком знакомстве с Google Landmarks Dataset v2 в процессе переработки проявились его качественные изъяны. В классах находилось огромное количество некорректных данных — фотографий, по которым даже осведомленный человек не смог предсказать достопримечательность.



Рис. 3.2. Пример не информативного изображения Эльбруса

Например, фото мужчины по колено в снегу с абсолютно белым фоном из-за метели относится к классу Эльбрус (рис. 3.2). Подобный кадр можно сделать в непогоду в любом северном уголке. Он не несет никакой полезной в определении достопримечательности информации ни для человека, ни для искусственного интеллекта. Также встречались неверно классифицированные изображения (брелок в форме Эйфелевой башни) и устаревшие фотографии (здания до ремонта, существенно изменившего их внешний вид), что несомненно запутывает нейронную сеть. Существенным изъяном является то, что некоторые классы содержат всего 3 фотографии.

# Создание RndLandmarks

Так как Google Landmarks Dataset v2 потребовал больших вычислительных мощностей, при этом предоставляя данные сомнительного качества, было принято решение отказаться от работы с ним. Для распознавания достопримечательностей был создан свой качественный набор данных по городу Ростов-на-Дону — RnDLandmarks.

RnDLandmarks состоит из сорока шести классов с различными достопримечательностями города. Это монументы, памятники архитектуры, парки, исторические места и необычные локации. Каждый класс содержит в себе разное количество изображений, в зависимости от сложности достопримечательности. Минимальное количество снимков на достопримечательность — тридцать, максимальное — более сотни. Например, монумент Тачанка-ростовчанка, который расположен на южном въезде в Ростов-на-Дону, описан только тридцать одной фотографией. Набор изображений включает в себя виды на памятник с различных ракурсов, в темное и светлое время суток. Так же есть снимки в зимнее время года, где монумент слегка припорошило снегом. А вот класс Городской парк имени Максима Горького описан сто двадцатью снимками. Эта достопримечательность является более сложной, так как включает в себя сразу несколько локаций: парк аттракционов, аллея с гирляндами, фонтан, скульптурный монумент, несколько памятников, арка под мостом, аллея с характерной клумбой и статуями.

Для составления датасетов использовались различные источники фотографий:

* поисковая система изображений Google;
* поисковая система изображений Яндекс;
* поисково-информационная картографическая служба Google;
* публичные страницы социальных сетей (с разрешения владельцев страниц);
* создание новых снимков.

Один из возможные сценариев применения системы распознавания достопримечательностей — это узнать, где подруга сделал понравившееся мне фото, чтобы повторить его. Таким образом, на изображении могут присутствовать помехи в виде людей. При создании набора данных была учтена эта особенность. Часть снимков содержит толпу горожан, торопящихся по своим делам, а часть снимков являются селфи либо личными фото, созданными с помощью штатива.

В RnDLandmarks 60% изображений было снято лично на несколько мобильных устройств:

* samsung galaxy S9
* samsung galaxy S22+
* iphone 15
* iphone 9
* samsung J6+
* honor 5

Особенно сложно было найти фотографии в зимнее время года, так как в Ростове-на-Дону снег является не частым явлением. Большим преимуществом при составлении датасета является личное знакомство с городом. Это позволяет добавить на карту не очевидные достопримечательности, о которых не прочитать в интернете, что делает приложение-гид особенным. Например, торговый центр под открытым небом в одном из внутренних дворов в центре города — Инсайд Спейс.

RnDLandmarks имеет стандартную файловую систему. Каждый класс хранится в отдельной директории. Директории пронумерованы от 0 до 45 — это уникальный айди класса достопримечательности. Сопоставление айди с названием достопримечательности хранится в rndlandmarks.csv файле. Всего в таблице 5 полей:

* id - уникальный айди достопримечательности;
* name - название достопримечательности;
* description - краткое описание достопримечательности;
* url - ссылка на вебсайт с более полной информацией
* geopos — широта и долгота.

Деление датасета на тренировочный, валидационный и тестовый набор происходит программно. Поэтому все директории-классы хранятся в одной папке RndLandmarks.

# Обучение c помощью фреймворка PyTorch

На сегодняшний день существует огромное множество архитектур сверточных нейронных сетей, которые решают задачу классификации изображений. Сравнение частоты ошибок (в процентах) при валидации наиболее известных архитектур на наборе данных ImageNet представлено в таблице 5.1.

Таблица 5.1. Частота ошибок (%) в наборе валидации ImageNet

|  |  |
| --- | --- |
| метод | Частота ошибок (%) |
| VGG | 7.1 |
| PReLU-net | 5.71 |
| BN-inception | 5.81 |
| ResNet-34 | 5.71 |
| ResNet-50 | 5.25 |
| ResNet-101 | 4.6 |
| ResNet-152 | 4.49 |

Лучшую точность предсказания класса демонстрирует семейство моделей Residual Neural Network (подробнее об архитектуре будет рассказано в следующих главах). ResNet обладает несколькими преимуществами, которые сделали ее очень популярной и успешной в области компьютерного зрения.

Преимущества Residual Neural Network:

* решена проблема затухания градиента, которая часто наблюдается в глубоких нейронных сетях;
* большое количество слоев позволяет извлекать более сложные признаки из изображений;
* в сложных задачах классификации архитектура демонстрирует лучшую производительность;
* данная архитектура проста в использовании, так как исходных код находится в открытом доступе и модель поддерживается популярными фреймворками.

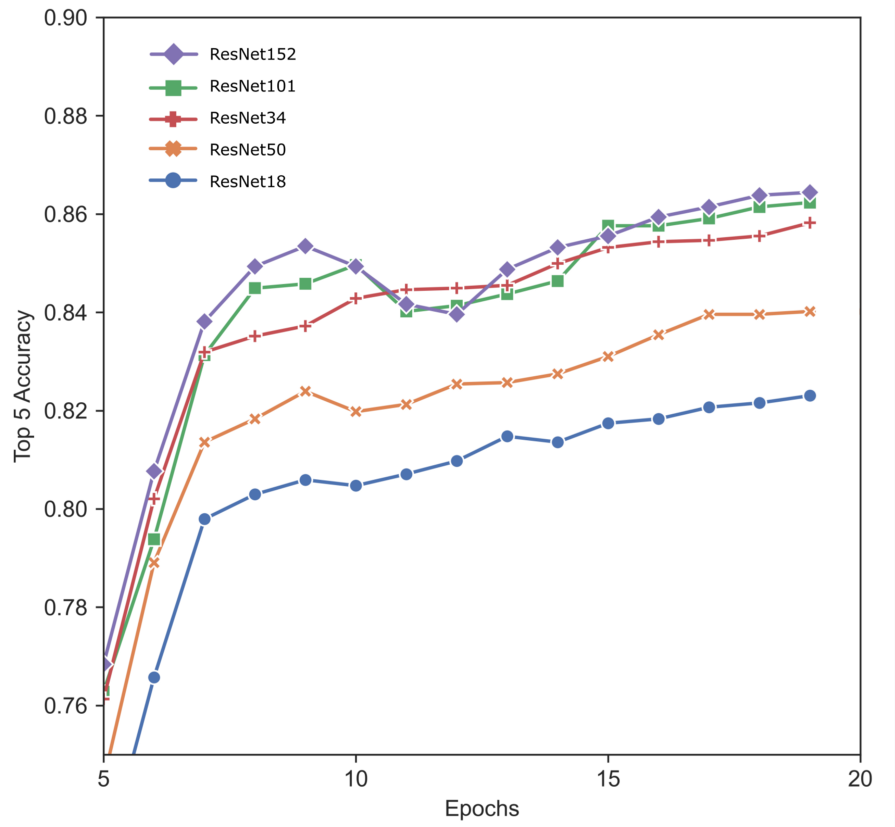


Рис. 5.1. Частота ошибок (%) в наборе валидации ImageNet

Семейство ResNet состоит из нескольких моделей разной глубины (рис. 5.1). Для решения задачи классификации достопримечательностей на изображении была выбрана архитектура ResNet101, так как она демонстрирует высокую точность предсказания. Время, необходимое для обучения нейронной сети, прямо пропорционально количеству ее слоев. Архитектура ResNet-152 является избыточной, потому что при добавлении пятидесяти одного слоя точность предсказания увеличивается не значительно.

Для загрузки предварительно обученной модели ResNet-101 из репозитория PyTorch Hub используется функция torch.hub.load() с указанием репозитория pytorch/vision:v0.10.0. Данная сверточная сеть позволяет извлечь признаки из изображения. Для определения принадлежности полученных эмбедингов к конкретному классу модель дополняется простым полносвязанным слоем torch.nn.Lianer(). На вход слой принимает такое же количество сигналов, какое будет на выходе у загруженной модели. Количество выходных сигналов полносвязного слоя равно количеству классов. Таким образом получаются оценки для каждого класса. Для того чтобы преобразовать их в вероятности принадлежности, к модели добавляется слой torch.nn.Softmax. Для получения предсказания остается лишь применить функцию Argmax, возвращающую индекс максимального элемента в массиве, и преобразовать id класса в его название с помощью csv таблицы.

Обучение происходит в течении 50 эпох на изображениях с разрешение 512х512. В качестве функции потерь используется torch.nn.CrossEntropyLoss(), а в качестве оптимизатора torch.optim.Adam() со скоростью обучения 0.001. Размер батча был определен на этапе загрузки датасета и раен 32.

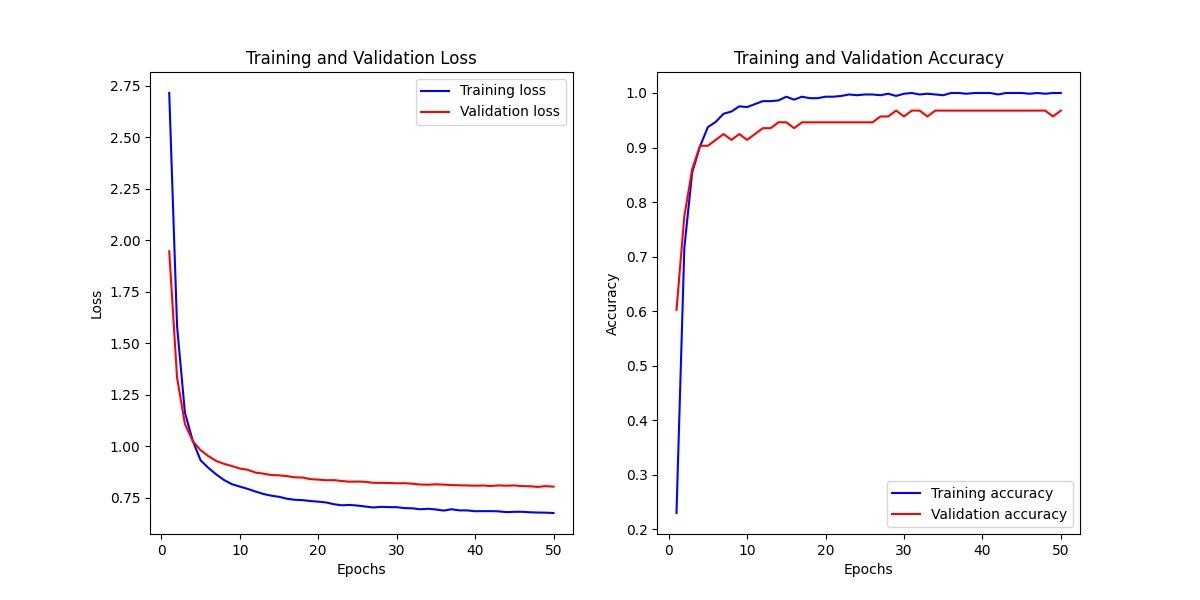


Рис. 5.2. Изменения точности и функции потерь

в течении эпох обучения модели ResNet101

Обучение происходило в бесплатной интерактивной облачной среде для работы с кодом на языке Python от Google в браузере — Google Colab, с использованием GPU ускорителя и заняло около 14 часов. График изменения точности и функции потерь в течении эпох представлен на рисунке 5.2. По окончании процесса, состояние модели (только ее обучаемые параметры) сохраняются в pth файле.

# Экспортирование в TFLite

TensorFlow Lite (TFLite) является открытой библиотекой Google. Она представляет собой легковесную версию фреймворка TensorFlow, оптимизированную для работы на устройствах с вычислительными ограничениями и низким энергопотреблением. Например, на микроконтроллерах или мобильных устройствах. Легковесные нейронные сети позволяют выполнять предсказание (инференс) прямо на телефоне, не прибегая к подключению облачных серверов. Таким образом, приложение сможет работать без подключения к сети интернет.

Преобразование модели PyTorch в формат TensorFlow Lite происходит в несколько этапов.

1. Для начала модель PyTorch экспортируется в промежуточный формат — ONNX. Open Neural Network Exchange позволяет обмениваться моделями между различными фреймворками. Для этого используется функция torch.onnx.export() с указанием следующих парметров:

* torch\_model — модель, предварительно загруженная из pth файла состояния с помощью функции torch.load();
* dummy\_input — простой пример входных данных, для понимания размерности; ◦ model\_name — имя файла для сохранения модели в формате ONNX.

1. Затем ONNX-модель преобразуется в TensorFlow граф состояния, который хранится в pb файле. Для этого используется функция onnx-tf.convert\_onnx\_to\_tf() с указанием одного парметра:

* model — модель, предварительно загруженная из onnx файла состояния с помощью функции onnx.load().

Для сохранения графа состояния используется функция tensorflow.io.write\_graph().

1. И наконец, модель TensorFlow преобразуется в TensorFlow Lite. Для этого необходимо загрузить модель с помощью tensorflow.TFLiteConverter.from\_saved\_model() и вызвать метод convert(). Конвертер позволяет оптимизировать модель для мобильных устройств, уменьшая размер модели и повышая ее производительность. Модель TensorFlow Lite сохраняется в файл tflite.

Экспортировав модель в формат TensorFlow Lite, ее легко применить в мобильном приложении. Библиотека предоставляет простой API для интеграции моделей TensorFlow Lite в мобильные приложения на Android и iOS. После всех преобразований размер модели и уменьшился более чем в 4 раза и составил 34 мегабайта.

# Библиотека TensorFlow Lite Model Maker

Модель машинного обучения, созданная с помощью фреймворка PyTorch, после экспортирования в TFLite оказалась слишком тяжеловесной для работы на мобильном устройстве с ограниченными ресурсами. Поэтому, было принято решение упростить архитектуру нейронной сети.

Хорошие готовые решения для задачи распознавания изображения предоставляет сервис ML Kit. Но возможности настройки модели из данной библиотеки сильно ограничены.

Библиотека TensorFlow Lite Model Maker предоставляет возможность трансферного обучения самых популярных архитектур нейронных сетей для решения задачи классификации изображений. Плюс, фреймворк TensorFlow Lite был разработан специально для работы на мобильных устройствах, сохраняя производительность на компактных моделях. Для успешного запуска обученной модели на мобильном устройстве необходимо оптимизировать ее, а точнее:

* уменьшить веса моделей (кластеризация (рис 7.1), квантизация, обрезка, объединение операций);
* понизить энергопотребление (уменьшить время ввода и вывода модели);
* оптимизировать использование оперативной памяти.

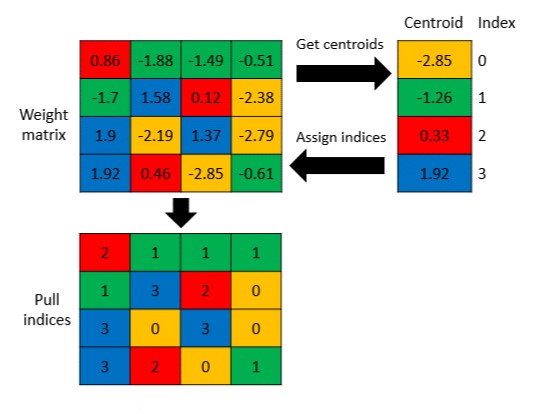


Рис. 7.1. Кластеризация весов модели

Завершив процесс обучения, TensorFlow Lite Model Maker автоматически оптимизирует полученную модель, что уменьшает ее размер и повышает скорость выполнения на мобильном устройстве. TensorFlow Lite Model Maker предоставляет возможность трансферного обучения семи моделей машинного обучения:

* Resnet50;
* EfficientNetLite0;
* EfficientNetLite1;
* EfficientNetLite2;
* EfficientNetLite3;
* EfficientNetLite4;
* MobileNetV2.

У глубоких сверточных нейронных сетей существует проблема деградации, которая заключается в том, что при добавлении слоев в сеть в какой-то момент качество достигает своего предела и начинает падать. Resnet50, что является сокращением от Residual Network, состоит из 50 остаточных блоков (Residual Blocks [6] (рис. 7.3), которые решают проблему деградации. Пример блока представлен на рисунке 7.2.

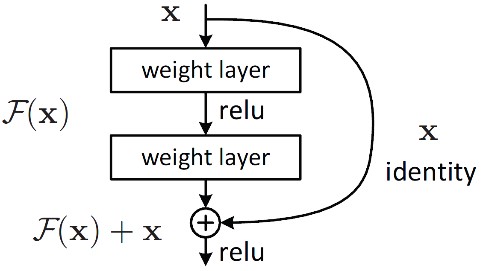


Рис. 7.2. Остаточный блок

Пусть F(x) — сложная нелинейная функция, полученная объединением нескольких слоев. Проблема деградации подразумевает, что если на предыдущих слоях был достигнут предел качества, то сеть должна выучить тождественное преобразование. Однако это оказалось сложной задачей для оптимизатора. Поэтому добавляется shortcut-соединение. Оптимизатору будет легче сделать все веса в основной ветке близкими к нулю, нежели создавать тождественное преобразование. Получается, что нейросеть сама решает на сколько глубокой нужно быть, выкидывая некоторые слои. Данная манипуляция обоснована, так как нейронная сеть может аппроксимировать почти любую функцию, например некоторую сложную функцию H(x). Тогда справедливо, что такая сеть легко выучит остаточную функцию 𝐹(𝑥)=𝐻(𝑥)−𝑥. Очевидно, что наша первоначальная целевая функция будет равна 𝐻(𝑥)=𝐹(𝑥)+𝑥. Residual Blocks позволяют понизить процент ошибки при обучении, который высок в глубоких сетях (рис. 7.3).

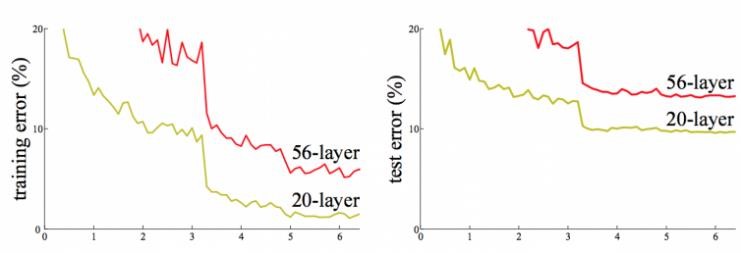


Рис. 7.3. Красный график демонстрирует изменение процента ошибки при обучении сети из 56 слоев, а зеленый – из 20 слоев. Правый график, в отличие от левого, имеет Residual Blocks.

Архитектуры Resnet101 и Resnet152 не рассматриваются, по причине своей тяжеловесности. Как уже было описано в предыдущих главах, разработка мобильного приложения требует легковесных моделей, для возможности инференса на устройствах с ограниченными ресурсами.

MobileNetV2 состоит из InvertedResBlock, наследующих идею остаточных блоков. Если внимательно посмотреть на количество входных и выходных сигналов в слоях ResBlock, параллельно shortcut-соединению можно заметить «узкое горлышко» — размерность вектора признаков уменьшается, а затем увеличивается (рис. 7.5). Сначала с помощью свертки 1х1 происходит объединение каналов в один признак для каждого пикселя (в случае работы с изображениями это объединение каналов R, G и B). Происходит обработка признаков сверткой 3х3 без изменения размерности. Затем число каналов увеличивается. Инвертированный остаточный блок использует обратный подход — количество фитч увеличивается с помощью ядра 1х1, также обрабатывается ядром 3х3, и затем восстанавливается исходная размерность ядром 1х1 (рис. 7.4). Некоторые инвертированные блоки MobileNetV2 не имеют остаточной связи.

а) б)

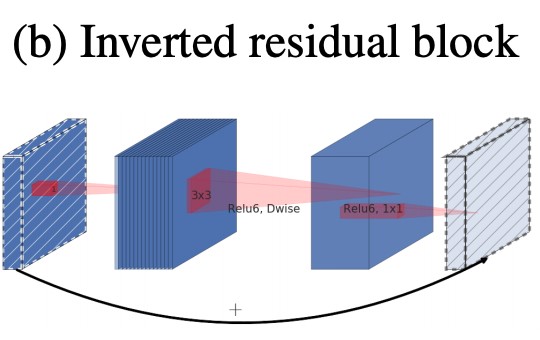
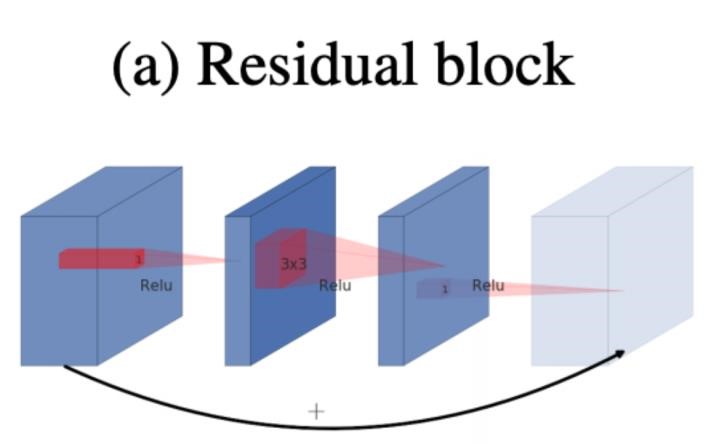


Рис. 7.4. (а) остаточный блок (б) инвертированный остаточный блок

EfficientNet состоит из InvertedResBlock, которые являются основным строительным материалом для архитектуры MobileNetV2. Четыре проиндексированных модели (EfficientNet1, EfficientNet2, EfficientNet3, EfficientNet4) были получены, улучшая способности предсказания EfficientNet0 с помощью масштабирования.

Помимо масштабирования в глубину (увеличение количества слоев), существует так же масштабирование в ширину. Оно заключается в увеличении количества нейронов в слое.

Если первое повышает сложность изучаемых признаков, то второе позволяет изучать более детализированные признаки. Расширение слоев можно увидеть в таких архитектурах нейронных сетей как MobileNet и WideResNet. Третий метод улучшения способности предсказания — это масштабирование разрешения. Это позволяет нейронной сети посмотреть на изображение вблизи и найти более мелкие отличительные структуры. К сожалению, каждый метод имеет свои ограничения. Например, 1000 остаточных блоков дают приблизительно тот же результат, что и сотня. Увеличение количества нейронов в слое требует больших вычислительных ресурсов. Масштабирование разрешения само по себе ограничено.

Обозначим глубину d (depth), ширину w (width), разрешение r (resolution). Для эффективного комбинированного масштабирования необходимо подобрать такие параметры d w r, чтобы максимальное повышение точности влекло минимальное повышение вычислительной мощности. Если строить графики зависимости, можно увидеть прямую пропорциональность для глубины и квадратичную пропорциональность для ширины и разрешения. То есть при утроении глубины вычислительная мощность тоже утроится, а при утроении ширины или разрешения вычислительная мощность увеличится в девять раз. Введя ограничение на комбинированное масштабирование можно гарантировать, что количество потребляемых ресурсов вырастет не более чем в два раза. При сетевом поиске оптимальных параметров масштабирования были получены следующие коэффициенты: depth = 1.2, width = 1.1 и resolution = 1.15. Применив их к сети из инвертированных остаточных блоков, была получена архитектура EfficientNet1.

Порядковый номер в названии архитектуры означает степень двойки прироста количества потребляемых ресурсов. Так, например, EfficientNet3 повышает потребление вычислительной мощности в восемь раз. В соответствии с формулой ограничения ее коэффициенты масштабирования будут равны depth = , width = и resolution = . Семейство моделей EfficientNetLite было выпущено в марте 2020 года и представляет собой облегченную архитектуру, способную работать на мобильных процессорах. Пять вариантов размера моделей дают возможность выбора точности предсказания в зависимости от количества вычислительных мощностей на мобильном устройстве.

Квантование является главным методом адаптации EfficientNet под периферийное устройство, по причине того, что последние сильно ограничены в вычислительных операциях над числами с плавающей запятой. Однако, квантование после обучения понизило точность предсказаний на наборе данных ImageNet с 75.1 до 46 процентов, что связано со слишком широким диапазоном значений выходного сигнала. Это можно исправить функцией активации. ReLuменее требовательно к вычислительным ресурсам, чем гиперболический тангенс или сигмоида, так как производит более простые математические операции. Поэтому она является оптимальной функцией активации при создании глубоких нейронных сетей. Все слои активации были заменены на функцию RELU6 (рис. 7.5), что вернуло точность предсказаний на наборе данных ImageNet к 74.4 процентам.

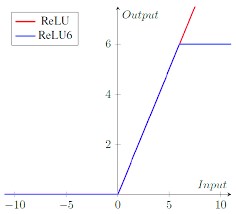


Рис. 7.5. Функция активации ReLu6

Различные мобильные устройства используют широкий спектр ускорителей (mobile GPU, EdgeTPU), которые достаточно хорошо работают на ограниченном наборе операций. По этой причине из EfficientNet были удалены некоторые слои. Таким образом, размер модели EfficientNetLite в четыре раза меньше своего родителя, а скорость вывода в 2 раза быстрее при незначительных потерях в точности (- 0.7%).

# Обучение и эксперименты TensorFlow Lite Model Maker

Итак, TensorFlow Lite Model Maker предоставляет возможность трансферного обучения семи моделей машинного обучения: Resnet50, пять моделей EfficientNetLite и MobileNetV2.

Для этого в библиотеке есть функция image\_classifier.create(), которая содержит в себе сразу несколько этапов:

1. Загрузка базовой модели из TensorFlow Hub, которая уже предварительно обучена извлекать признаки из изображения.
2. Добавление слоев преобразования полученных эмбедингов в вероятности принадлежности классам.
3. Предварительная обработка входных данных, а именно:

* попиксельная нормализация каждого канала RGB изображения от диапазона [0,225] к диапазону [0,1];
* изменение разрешения изображения в соответствии с размером входных данных модели.

1. Подача данных на конвейер обучения классификатора с гиперпараметрами по умолчанию.
2. Обучение классификатора с валидацией полученной модели.

Загрузка набора данных RndLandmark происходит с помощью функции TensorFlow Lite Model Maker image\_classifier.DataLoader.from\_folder(data\_dir), возвращающих класс DataLoader. Директория data\_dir должна содержать столько поддиректорий, сколько классов содержит в себе набор данных. Названия поддиректорий могут быть названиями классов или их id. Разделение данных на обучающую, валидационную и тестовую выборку происходит програмно. Метод split(proc) класса DataLoader разделяет данные на две части в процентном соотношении proc.

Чтобы повлиять на точность предсказания моели можно изменить значения гиперпараметров обучения сконфигурировав опциональные парметры функции image\_classifier.create():

* epochs — количество эпох обучения;
* batch\_size — количество пар изображение-лэйбл, которые будут использоваться в одном шаге обучения;
* validation\_data — данные для подсчета точности и функции потерь после каждой эпохи;
* train\_whole\_model — булевый параметр, отвечающий за отключение трансферного обучения;
* learning\_rate — скорость обучения.

Чтобы построить графики изменения функция потерь и точности предсказания в течение эпох используется библиотека matplotlib. После обучения числовые ряды хранятся внутри модели и могут быть извлечены методом model.history (листинг).

Листинг 8.1 Вырезка из метода model\_experiment()

...

# Получаем значения loss и accuracy из истории обучения

history = model.history

loss = history.history['loss']

accuracy = history.history['accuracy']

# Получаем значения loss и accuracy для валидационных данных val\_loss = history.history['val\_loss']

val\_accuracy = history.history['val\_accuracy']

# размерность оси абсицц и разрешение изображения

epochs = range(1, len(loss) + 1)

plt.figure(figsize=(12, 6))

...

# Сохранение графика в формате JPG

plt.savefig(os.path.join(save\_dir, 'my\_plot.jpg'))

…

# Экспортирование модели в формат tflite

config = QuantizationConfig.for\_float16()

model.export(export\_dir=save\_dir, tflite\_filename='model.tflite', quantization\_config=config)

Для измерения качества модели на новых данных вызывается метод model.evaluate на тестовом наборе данных. Полученные потери и точность, а так же время, затраченное на обучение модели, сохраняется в той же папке, что и графики с помощью библиотеки os.path. Библиотека TensorFlow Lite Model Maker предоставляет возможность экспортирования модели в формат tflite и сохранения. Для этого необходимо вызвать метод model.export(), указав директорию для сохранения файла, имя файла и размерность квантования. Описанный процесс обернут в функцию model\_experiment() для многократного вызова обучения моделей с различной конфигурацией и сохранения результатов, с целью дальнейшего сравнительного анализа.

Каждая из семи моделей, поддерживаемых библиотекой TensorFlow Lite Model Maker, была обучена трансферно в течении ста эпох. Входные изображения имели разрешение 224х224. В качестве функции потерь используется кросэнтропия, а в качестве оптимизатора Адам со скоростью обучения 0.001. Размер батча по умолчанию равен 256.

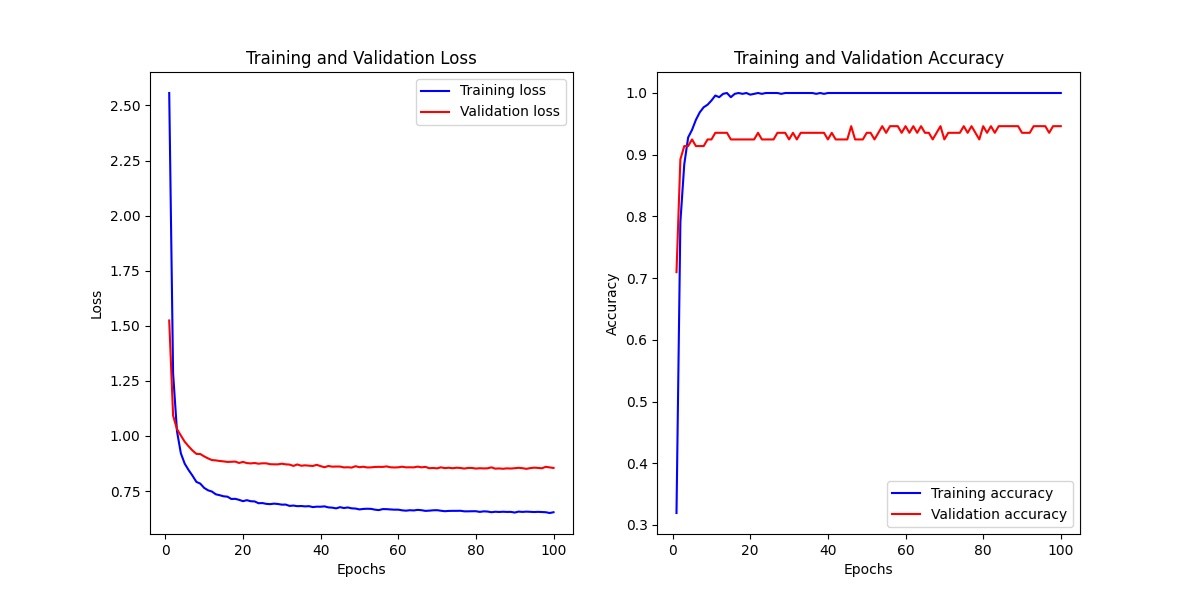


Рис. 8.1. а) Изменения точности и функции потерь

в течении эпох обучения модели MobileNetV2

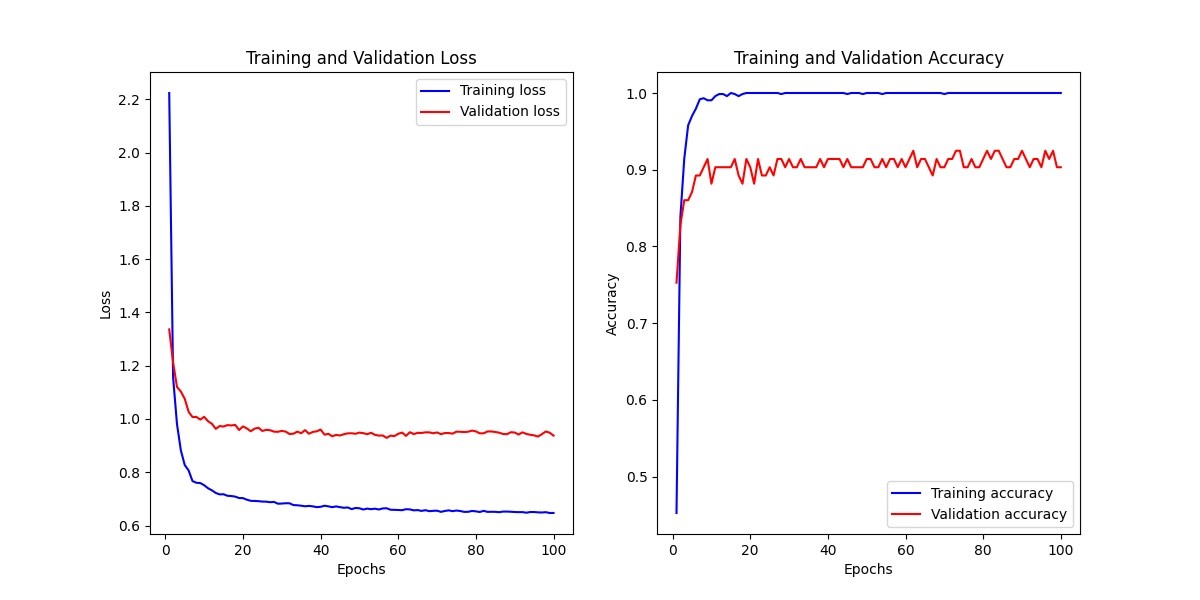


Рис. 8.1. б) Изменения точности и функции потерь

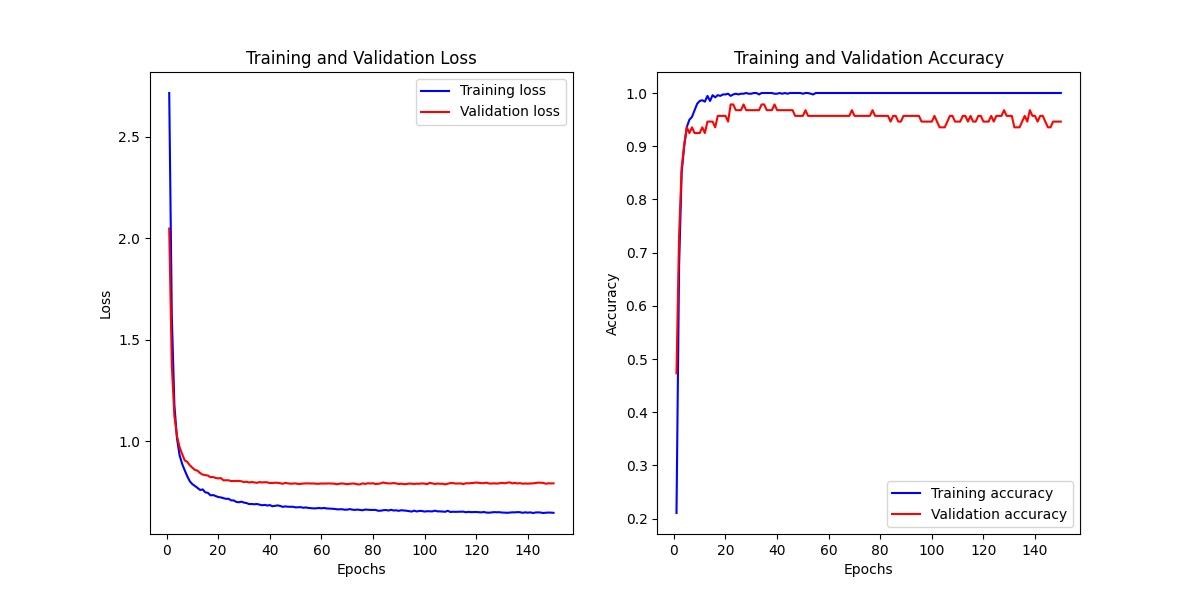
в течении эпох обучения модели Resnet50

Рис. 8.1. в) Изменения точности и функции потерь

в течении эпох обучения модели EfficientNetLite0

Проанализировав полученные графики изменения валидационной и тренировочной точности предсказания, сделаны выводы о необходимом количестве эпох для каждой модели. MobileNetV2 достигает максимальной точности на валидационной выборке данных на пятьдесят пятой эпохе. Дальнейшее обучение вызывает колебания точности предсказания, которое не превышает максимально достигнутого порога. Таким образом, шестидесяти эпох более чем достаточно. EfficientNetLite0, EfficientNetLite2, EfficientNetLite3 быстрее достигают пиковой точности — на сороковой эпохе. А EfficientNetLite1 и EfficientNetLite4 — на сорок пятой эпохе. Для семейства EfficientNetLite достаточно пятидесяти эпох обучения. Resnet50 обучается медленнее всех — необходимо восемьдесят эпох обучения.

Трансферное обучение происходит на основе моделей, которые уже предобучены на каком-то наборе данных. Основная идея заключается в том, чтобы не придумывать новых функций для извлечения признаков из RGB изображения, за что отвечает базовая модель, хранящаяся в TensorFlow Hub. Нужно лишь научить классификатор преобразовывать выходные эмбединги в оценки новых классов. Для этого веса базовой модели замораживаются и градиент корректирует только слой классификации. Точности трансферно обученных моделей на тестовой выборке представлены в таблице 8.1.

Таблица 8.1. Точность обученных моделей на тестовой выборке

|  |  |  |  |
| --- | --- | --- | --- |
| Архитектура | Трансферное  обучение | Обучение  всей модели | Изменение точности |
| resnet\_50 | 0.947 | 0,915 | -3.49% |
| efficientnet\_lite0 | 0.968 | 0,947 | -2.25% |
| efficientnet\_lite1 | 0,894 | 0,936 | 4.55% |
| efficientnet\_lite2 | 0,936 | 0,947 | 1.12% |
| efficientnet\_lite3 | 0,883 | 0,926 | 4.6% |
| efficientnet\_lite4 | 0,925 | 0,968 | 4,47% |
| mobilenet\_v2 | 0.883 | 0,936 | 5.68% |

Для того чтобы научить нейронную сеть извлекать из изображения признаки, специфичные для конкретной задачи распознавания достопримечательностей, нужно разморозить базовую модель, чтобы градиент скорректировал веса всей нейронной сети. Для этого в функция image\_classifier.create() передается параметр train\_whole\_model со значением True. Как видно из таблицы 1 большинство моделей улучшили свои показатели на 4-5 процентов. А resnet\_50 и EfficientNetLite0 потеряли в точности 2-3 процента. Трансферное подход сокращает время, затраченное на обучение, но не влияет на инференс. По этой причине можно смело использовать полностью обученные модели в мобильном приложении.

Таблица 8.2. Точность и потери моделей,

обученных с разными размерами батча

|  |  |  |
| --- | --- | --- |
| model | loss | accuracy |
| efficientnet\_lite0-80-whool | 0,809 | 0,947 |
| efficientnet\_lite0-80-whool-bs32 | 0,782 | 0,947 |
| efficientnet\_lite0-80-whool-bs64 | 0,782 | 0,947 |
| efficientnet\_lite4-80-whool | 0,737 | 0,968 |
| efficientnet\_lite4-80-whool-bs32 | 0,734 | 0,968 |
| efficientnet\_lite4-80-whool-bs64 | 0,734 | 0,968 |
| mobilenet\_v2-80-whool | 1,03 | 0,936 |
| mobilenet\_v2-80-whool-bs32 | 1,008 | 0,936 |
| mobilenet\_v2-80-whool-bs64 | 1,008 | 0,936 |
| resnet\_50-80-whool | 1,19 | 0,915 |
| resnet\_50-80-whool-bs32 | 1,189 | 0,915 |
| resnet\_50-80-whool-bs64 | 1,189 | 0,915 |

Было выполнено несколько экспериментов с изменением количества пар изображение — лэйбл класса в батче. Модели MobileNetV2, EfficientNetLite0, EfficientNetLite4 и Resnet50 были обучены в течении 80 эпох (обучение всей модели) с размерами батча 32 и 64 (таблица 8.2). Это никак не повлияло ни на точность предсказания, ни на функцию потерь. Поэтому было принято решение оставить размер батча значением по умолчанию.

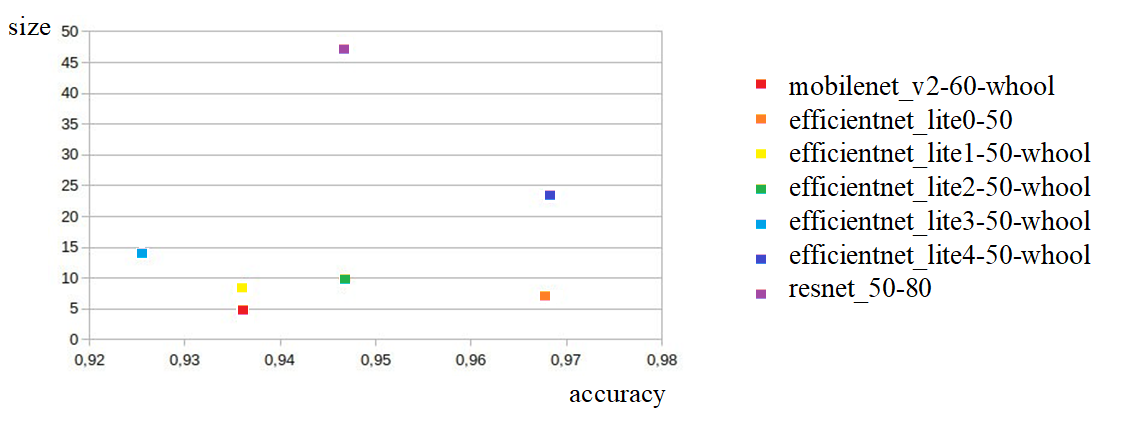


Рис. 8.2. Соотношение размера и точности моделей

Исходя из сравнительного анализа полученных данных, в конечном мобильном приложении для распознавания достопримечательности на RGB изображении используется ансамбль из 4 сверточных нейронных сетей:

* MobileNetV2 полностью обученный в течении 60 эпох;
* EfficientNetLite0 трансферно обученный в течении 50 эпох;
* EfficientNetLite1 полностью обученный в течении 50 эпох;
* EfficientNetLite2 полностью обученный в течении 50 эпох.

Модель Resnet50 не попадает в итоговое решения из-за своего размера — 47 мегабайт, а EfficientNetLite3 из-за низкой точности предсказания — 92 процента. EfficientNetLite4 тоже является тяжеловесной моделью — 23 мегабайта, при этом давая точность предсказания близкую к EfficientNetLite0 — 97 процентов.

Объединение предсказаний происходит с помощью голосования. Из выходных сигналов каждой модели определяется топ 5 наиболее вероятных классов достопримечательностей с их вероятностной оценкой принадлежности. В полученном списке из 20 пар id-probability для каждого класса вычисляется сумма предсказанных вероятностей, возводя каждое слагаемое в восьмую степень. Слагаемое, полученное из предсказания модели EfficientNetLite0, дополнительно увеличивается с помощью коэффициента 1.2, так как эта сеть самая точная в ансамбле. Результатом является id одного класса достопримечательностей с наибольшей суммарной вероятностью.

# Разработка мобильного приложения

Мобильное приложение было разработано под операционную систему Android. Его основной задачей является распознавание достопримечательностей города Ростов-на-Дону по фотографии. В качестве дополнения была поставлена задача отображения местоположения распознанной достопримечательности на карте. Приложение должно содержать 3 экрана:

* главный экран, на котором пользователь может загрузить изображение, которое содержит неопознанную достопримечательность, из галереи мобильного устройства или же из приложения камера;
* экран демонстрации, на котором отображена основная информация о результате работы нейронной сети;
* экран карты, на котором отображается местоположения распознанной достопримечательности на карте города.

## Используемые инструменты и технологии

В качестве интерактивной среды разработки использовалась AndroidStudio версии Jellyfish // 2023.3.1. Язык программирования – Kotlin. Для упрощения процесса разработки использовалось множество библиотек, вот некоторые из них.

Jetpack Navigation для упрощения навигации между экранами приложения. Он предоставляет инструмент визуального проектирования в AndroidStudio – Navigation Editor. Это позволяет создавать навигационные графы (рис. 9.1), добавлять экраны и определять действия для перехода между ними.

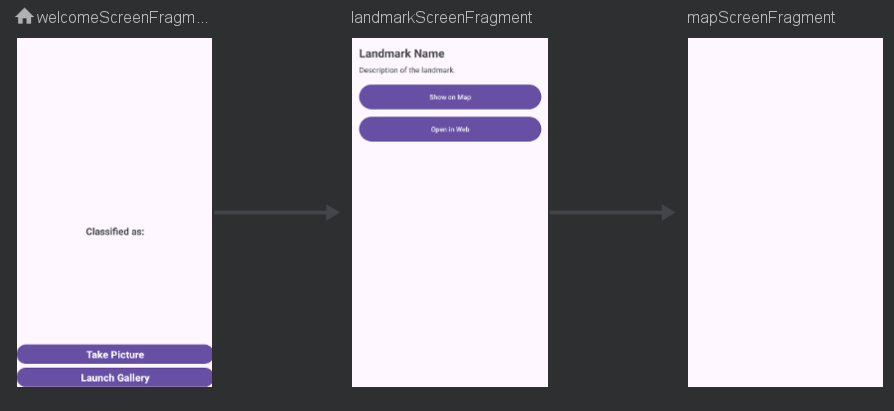


Рис. 9.1. Навигационный граф

TensorFlow Lite это фреймворк машинного обучения от Google, который оптимизирован для работы на мобильных устройствах. Мы используем три зависимости:

* tensorflow.lite.support – основная библиотека. Она содержит в себе инструменты для загрузки обученной модели из tflite файла, обработки входных изображений и выполнения прямого прохода модели.
* tensorflow.lite.metadata – библиотека для работы с метаданными моделей. Она позволяет получить тип и размерность входных и выходных данных моделей.
* tensorflow.lite.gpu – библиотека для использования графического процессора при вычислениях. Это ускоряет процесс и повышает производительность.

Android Studio версии медуза (2023.3.1) предоставляет возможность загрузки модели в формате tflite с помощью графического интерфейса (рис. 9.2). При этом среда разработки дает рекомендации, как лучше использовать модель.

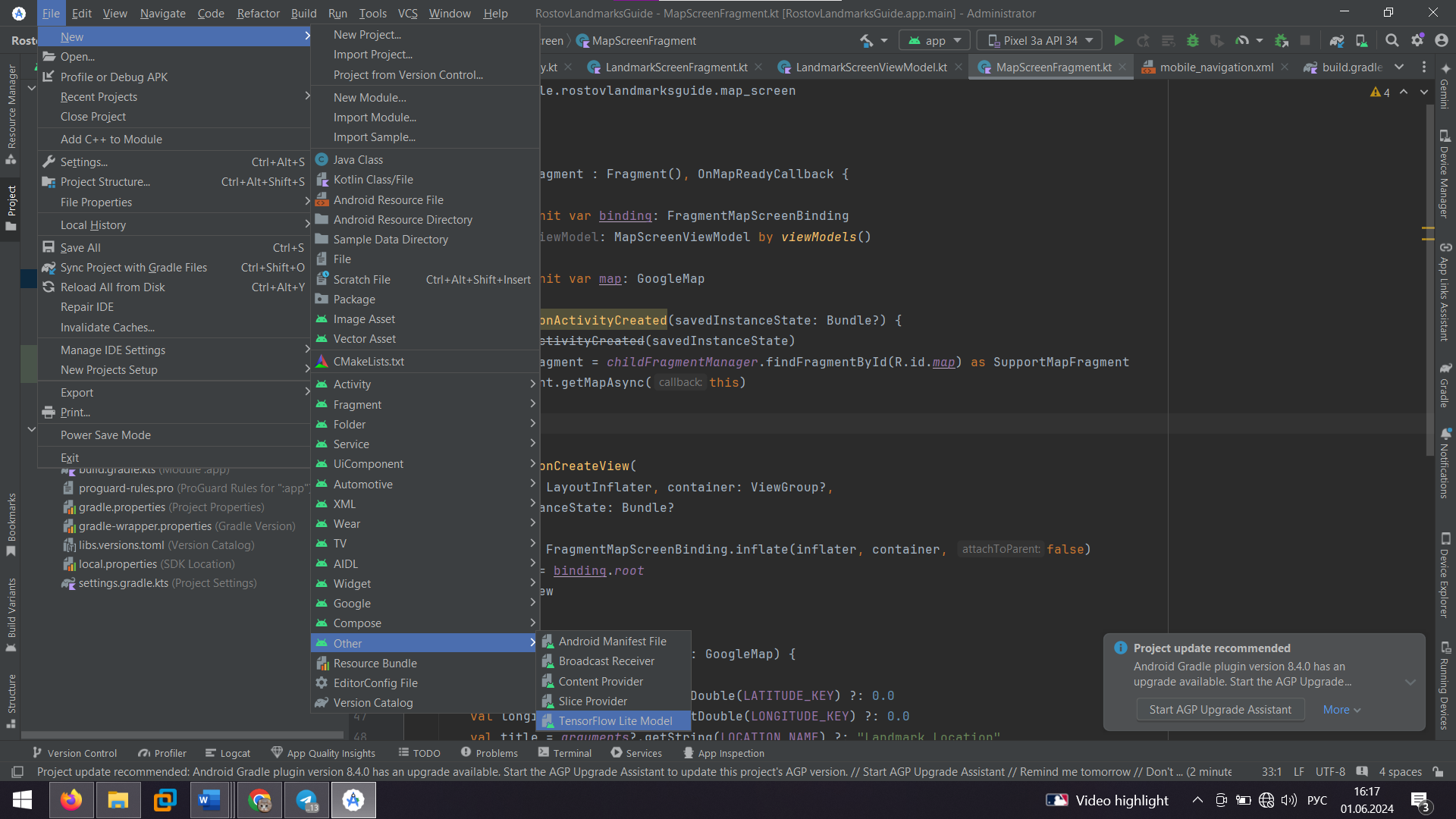


Рис. 9.2. Добавление tflite модели в Android Studio

GoogleMapsSDK (software development kit) для Android это набор инструментов для взаимодействия с GoogleMapsAPI, который упрощает интеграцию функционала карт в мобильное приложение. Например, управление данными о местоположении пользователя. возможность отображать карты и строить маршруты.

Чуть подробнее об интеграции карт в мобильное приложение. Первое, что необходимо сделать, это получить ключ к API карт. Для этого был создан новый проект в Google Cloud Console. Затем для проекта был активирован сервис Google Maps Platform и во вкладке учетные данные создан ключ API сервера. Данный ключ используется в мобильном приложении. Для этого в зависимости проекта добавляется библиотека libs.tensorflow.lite.gpu, а полученный ключ добавлен в файл AnsroidManifest.xml в тег application (листинг 9.1). Также Google Cloud Console предоставляет возможность настроить ограничение на использование ключа API по домену или ip-адресу.

Листинг 9.1. API ключ для интеграции карт

<application ...>

<meta-data

android:name="com.google.android.geo.API\_KEY"

android:value="YOUR\_API\_KEY"/>

</application>

## Работа с данными RnDLandmarks

Для работы с достопримечательностями был создан класс LandmarkEntity (листинг 9.2). Поля данного дата класса отражают информацию о классе достопримечательности, которая хранится в scv таблице набора данных RnDLandmarks. Это уникальный идентификатор класса, полное имя достопримечательности, геологическая позиция объекта, краткое описание и ссылка на полное описание или сайт объекта.

Листинг 9.2. Класс для десериализации данных

data class LandmarkEntity(  
 val id: Int,  
 val name: String?,  
 val geo: LatLng?,  
 val url: String?,  
 val description: String?,  
):Parcelable {  
 constructor(parcel: Parcel) : this(  
 parcel.readInt(),  
 parcel.readString(),  
 parcel.readParcelable(LatLng::class.*java*.*classLoader*),  
 parcel.readString(),  
 parcel.readString()  
 )

Сущность LandmarkEntity реализует интерфейс Parcelable. Данный интерфейс позволяет десериализовать каждую строку csv таблицы в объект сущности. Помимо данного интерфейса десериализации объектов существует интерфейс Serializable, который является основным для разработки Java приложений. Он прост в использовании, но может работать медленно и потреблять много памяти, так как использует механизм сериализации Java. Parcelable был разработан специально для Android, поэтому он гораздо лучше интегрируется в платформу, более стабилен в работе и значительно быстрее. Также этот интерфейс потребляет меньше памяти, чем Serializable, что является важным фактором при работе на мобильных устройствах с ограниченными ресурсами.

Для того чтобы работать с данными из таблицы, их необходимо хранить внутри приложения. Для этого csv таблицу, которая была подготовлена на этапе создания набора данных, загружается в ресурсы Android приложения app/src/main/res (resources) в специальную папку row, в которой хранятся файлы в необработанном формате. Для построчного чтения файла и сериализации потока данных в объекты объявленного класса сущности реализован метод readLandmarks (листинг 9.3).

Листинг 9.3. Загрузка данных

fun readLandmarks(context: Context): List<LandmarkEntity> {  
 val landmarks = *mutableListOf*<LandmarkEntity>()  
 val inputStream = context.*resources*.openRawResource(R.raw.*landmarks*)  
 val reader = CSVReader(InputStreamReader(inputStream))  
 reader.*use* { csvReader ->  
 val header = csvReader.readNext()  
 var line: Array<String>?  
 while (csvReader.readNext().*also* { line = it } != null){  
 line?.*let* {  
 val id = it[0].*toInt*()  
 val name = it[1]  
 val latitude = it[2].*split*(", ")[0].*toDouble*()  
 val longitude = it[2].*split*(", ")[1].*toDouble*()  
 val url = it[3]  
 val description = "it[4]"  
 val landmark = LandmarkEntity(  
 id = id,  
 name = name,  
 geo = LatLng(latitude, longitude),  
 url = url,  
 description = description  
 )  
 landmarks.add(landmark)  
 }  
 }  
 }  
 return landmarks  
}

## Главный экран

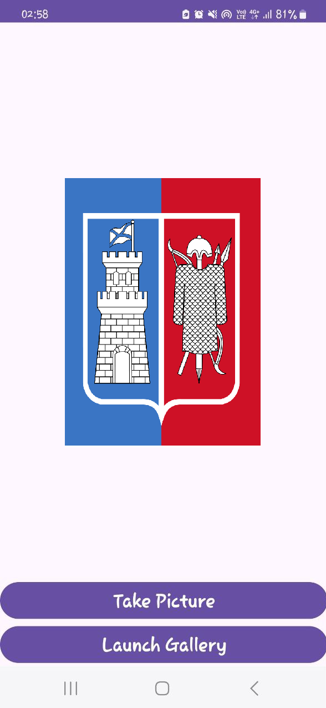
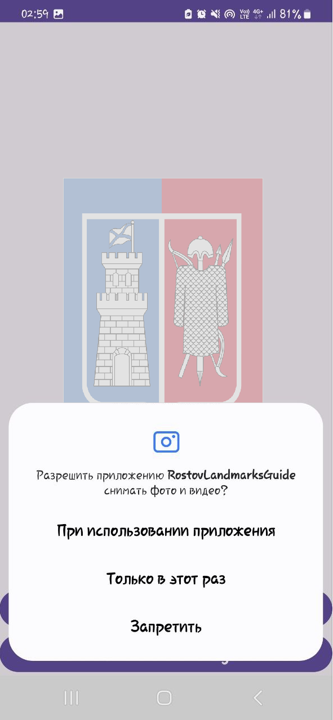
 

Рис. 9.3. Главный экран

При открытии приложения пользователь попадает на главный экран (WelcomeScreen) (рис. 9.3). Так как приложение предназначено для распознавания достопримечательностей Ростова-на-Дону, в центре изображен герб города, что не несет функциональной нагрузки. В нижней части экрана расположены две кнопки: Take Picture и Launch Gallery, которые окрашены в фиолетовый цвет. Во время своей работы приложение окрашивает в этот же оттенок статус бар телефона.

При нажатии на кнопку Launch Gallery пользователь переходит в системную галерею мобильного устройства для того, чтобы выбрать изображение (листинг 9.4). Для этого создается намерение (intent) на открытие галереи с кодом, по которому можно будет отследить результат выполнения намерения.

Листинг 9.4. Открытие галереи

binding.galleryButton.setOnClickListener(View.OnClickListener {  
 val cameraIntent = Intent(Intent.*ACTION\_PICK*, MediaStore.Images.Media.*EXTERNAL\_CONTENT\_URI*)  
 startActivityForResult(cameraIntent, 1)  
})

При нажатии на кнопку Take Picture пользователь переходит в системную камеру мобильного устройства для того, чтобы создать фотографию (листинг 9.5). Происходит проверка, есть ли среди разрешений данного приложения разрешение на доступ к камере. Если разрешения еще нет, как в случае первого запуска приложения на мобильном устройстве, оно запрашивается у пользователя. Получив разрешение, так же создается намерение (intent) на запуск камеры с кодом, по которому можно будет отследить результат выполнения намерения.

Листинг 9.5. Открытие камеры

binding.cameraButton.setOnClickListener(View.OnClickListener {  
 if (checkSelfPermission(  
 requireContext(),  
 Manifest.permission.*CAMERA*  
) == PackageManager.*PERMISSION\_GRANTED*  
) {  
 val cameraIntent = Intent(MediaStore.*ACTION\_IMAGE\_CAPTURE*)  
 startActivityForResult(cameraIntent, 3)  
 } else {  
 requestPermissions(*arrayOf*(Manifest.permission.*CAMERA*), 100)  
 }  
})

После окончания обработки соответствующего нажатой кнопке намерения, срабатывает метод OnActivityResult с параметром реквест-код, который был указан при старте намерения. Отследив результат по реквест-коду, загружается полученная из камеры или галереи фотография. Затем, метод classifyImage масштабирует изображения до размера входных данных модели (224х224) и запускается прямой проход через нейронные сети. Полученные предсказания объединяются методом голосования, который был описан в предыдущих главах. По идентификатору итогового класса создается объект достопримечательности LandmarkEntity. Переход на экран демонстрации осуществляется библиотекой Jetpack Navigation, при этом созданный объект достопримечательности передается по ключу с помощью бандла (bundle) (листинг 9.6).

Листинг 9.6. Передача сериализованного объекта на экран демонстрации

if (landmark != null) {  
 Navigation.findNavController(requireView())  
 .navigate(  
 R.id.*action\_welcomeScreenFragment\_to\_landmarkScreenFragment*,  
 *bundleOf*(LANDMARK\_KEY *to landmark*)  
 )  
}

## Экран демонстрации



Рис. 9.4. Экран демонстрации

Экран демонстрации (LandmarkScreen) предназначен для визуализации результатов работы ансамбля сверточных нейронных сетей (рис. 9.4). В верхней части экрана находится полное название туристического объекта. По центру отображается обрезанное входное изображение и краткое текстовое описание предсказанной достопримечательности. В нижней части экрана расположены две кнопки: Show on Map и Open in Web, окрашенные в фиолетовый цвет.

После инициализации всех виджетов экрана мы извлекаем из бандла по ключу объект LandmarkEntity и записываем его во вьюмодель (ViewModel). Вся дальнейшая работа будет производиться с объектом из вьюмодели. Это дает гарантию, что объект не будет утерян при смене конфигурации, например, при повороте экрана.

При нажатии на кнопку Open in Web пользователь переходит в веббраузер на страницу с полным описанием достопримечательности или на официальный сайт туристического объекта, например, если это храм или музей. Для этого создается намерение (intent) с указание url адреса (листинг 9.7).

Листинг 9.7. Открытие URL в браузере

binding.buttonOpenUrl.setOnClickListener {  
 val intent = Intent(Intent.ACTION\_VIEW, Uri.parse(viewModel.landmark.url))  
 startActivity(intent)  
}

При нажатии на кнопку Show on Map пользователь переходит на экран карты. В бандл по ключу передаются название и значения геологической позиции объекта из поля geo класса LandmarkEntity как latitude и longitude.

## Экран карты

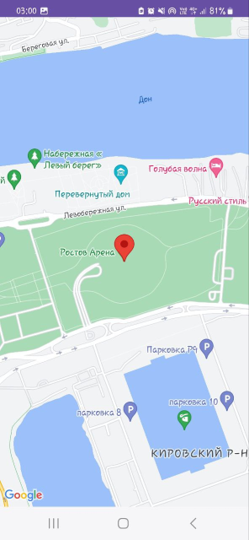


Рис. 9.5. Экран карты

Экран карты предназначен для отображения местоположения предсказанной ансамблем сверточных нейронных сетей достопримечательности на карте города (рис. 9.5). Фиолетовыми маркерами отмечены все туристические объекты, которые входят в набор данных RndLandmarks. Маркер текущего объекта окрашен в красный цвет. Рядом с маркером располагается подпись с названием достопримечательности.

Данный экран реализует интерфейс OnMapReadyCallback. Для этого переопределен метод OnMapReady. При инициализации экрана происходит асинхронная загрузка карты (листинг 9.8).

Листинг 9.8. Асинхронная загрузка карты

val mapFragment = *childFragmentManager*.findFragmentById(R.id.*map*) as SupportMapFragment  
mapFragment.getMapAsync(this)

После того как карта полностью загрузилась и готова к использованию, срабатывает метод OnMapReady (листинг 9.9). В этом методе извлекаются параметры локации и название, создается и добавляется на карту маркер текущего объекта и остальных объектов из набора данных, а также камера перемещается к текущему объекту.

Листинг 9.9. Отрисовка маркеров

override fun onMapReady(googleMap: GoogleMap) {  
 map = googleMap  
 val latitude = *arguments*?.getDouble(LATITUDE\_KEY) ?: 0.0  
 val longitude = *arguments*?.getDouble(LONGITUDE\_KEY) ?: 0.0  
 val title = *arguments*?.getString(LOCATION\_NAME) ?: "Landmark Location"  
 val location = LatLng(latitude, longitude)  
 val iconAll = BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.*HUE\_MAGENTA*) map.addMarker(MarkerOptions().position(location).title(title))  
 LandmarkEntity.readLandmarks(requireContext()).*forEach* {  
 if (it.name != title) { map.addMarker(MarkerOptions().position(it.geo!!).title(it.name).icon(iconAll))  
 }  
 }  
 map.moveCamera(CameraUpdateFactory.newLatLngZoom(location, 15f))

Таким образом на трех экранах полностью реализован возложенный на приложение основной и дополнительный функционал. Мобильное приложение было протестировано на устройствах Samsung Galaxy S22+, Samsung Galaxy S9 и Samsung J6+.

# Заключение

В результате выполнения данной работы создано мобильное приложение под платформу Android, распознающее достопримечательности по фото, средствами машинного обучения. Приложение работает без доступа к геологическим данным и сети интернет.

Для этого были решены следующие задачи:

* проведен анализ существующих методов решения задачи классификации достопримечательностей, основанных на машинном обучении и возможных готовых датасетов;
* создан датасет RnDLandmark достопримечательностей Ростова-на-Дону (более 2 тыс изображений для 46 классов);
* собрана информационная база, дополняющая изображения (краткое описание, геоданные и url);
* проведены эксперименты с обучением различных моделей;
* составлен ансамбль из 4 наиболее подходящих (размер, качество) моделей;
* разработано мобильное приложение для ансамбля нейросетей;
* в приложение интегрированно Google Maps SDK.

# Литература

1. Андрей Бояров «Как и зачем мы делали распознавание достопримечательностей в Облаке Mail.ru» 14.05.2019

URL – https://habr.com/ru/company/vk/blog/451542/ (дата обращения 17.03.2023)

1. Татьяна Корешкова «Механизмы поиска подходящего датасета» 15.04.2021

URL – https://rdc.grfc.ru/2021/04/analytics-computer-vision/?ysclid=

lbwb12yi2j953189125#post-368-\_Toc68698335 (дата обращения 20.05.2023)

1. А. А. Цветков, Д. К. Шорох, М. Г. Зубарева [и др.]. «Алгоритмы распознавания объектов» 29.06.2016

URL –https://moluch.ru/conf/tech/archive/166/10825/?ysclid=lbwb0vvjgc941096367 (дата обращения 19.03.2023)

1. Tobias Weyand, Andre Araujo, Bingyi Cao, Jack Sim «Google Landmarks Dataset v2 A Large-Scale Benchmark for Instance-Level Recognition and Retrieval» 02.11.2020

URL – https://arxiv.org/pdf/2004.01804.pdf (дата обращения 23.05.2023)

1. Zu Kim, Andre Araujo, Bingyi Cao, Cam Askew, Jack Sim, Mike Green, N’Mah Fodiatu Yilla, Tobias Weyand «Towards A Fairer Landmark Recognition Dataset» 06.06.2022

URL – https://arxiv.org/pdf/2108.08874.pdf (дата обращения 25.11.2023)

1. Christof Henkel, Philipp Singer «Supporting large-scale image recognition with out-of-domain samples» 04.10.2020

URL – https://arxiv.org/pdf/2010.01650 (дата обращения 07.12.2023)

1. Min Yang, Cheng Cui, Xuetong Xue, Hui Ren, Kai Wei «2nd Place Solution to Google Landmark Recognition Competition 2020» 13.10.2020

URL – https://arxiv.org/pdf/2210.01624 (дата обращения 20.12.2023)

1. Min Yang, Dongliang He, Miao Fan, Baorong Shi, Xuetong Xue, Fu Li, Errui Ding, Jizhou Huang «Dolg: Single-stage image retrieval with deep orthogonal fusion of local and global features» 05.10.2021

URL – https://arxiv.org/pdf/2108.02927 (дата обращения 19.01.2024)

1. Документация ONNX 1.17.0 "Сериализация»

URL – https://onnx.ai/onnx/api/serialization.html (дата обращения 30.01.2024)

1. Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, Dmitry Kalenichenk «Quantization and training of neural networks for efficient integer-arithmetic-only inference» 15.12.2017

URL – https://arxiv.org/pdf/1712.05877 (дата обращения 3.02.2024)

1. Документация TFLite ModelMaker «Image classification with TensorFlow Lite Model Maker»

URL – https://www.tensorflow.org/lite/models/modify/model\_maker/image\_classification (дата обращения 20.02.2024)