Санкт–Петербургский государственный университет

КУПРЯКОВ Владислав Евгеньевич

Выпускная квалификационная работа

Применение искусственных нейронных сетей для распознавания изображений в мобильном приложении

Уровень образования: бакалавриат
Направление 02.03.02
«Фундаментальная информатика и информационные технологии»
Основная образовательная программа
«Программирование и информационные технологии»

Научный руководитель:
кандидат физико-математических наук, доцент кафедры
теории систем управления электрофизической аппаратурой
Козынченко Владимир Александрович

Рецензент:
доктор физико-математических наук
Котина Елена Дмитриевна

Санкт-Петербург
2020 г.

Содержание

Введение

Нейронные сети все чаще внедряются в различные сферы нашей жизни: они применяются к задачам диагностики различных болезней, классификации объектов на фото, позволяют использовать переводчики текста и многое другое. Алгоритмы обучения нейросетей постоянно улучшаются. Компании по всему миру тратят огромное количество ресурсов на проблему совершенствования существующих систем. Спонсируется обучение новых специалистов, которые, в свою очередь, предлагают собственные решения разработанным аналогам. Параллельно с этим растет и вычислительная мощность пользовательских мобильных устройств, открывая новые возможности. Ключевым стал момент, когда разработчикам стала доступна интеграция нейросетевых моделей в мобильные приложения.

Данная работа посвящена разработке мобильного приложения, которое бы стало помощником на дороге автомобилистам, распознавая пешеходов. Продвинутые системы распознавания уже доступны в машинах Tesla с автопилотом, но такие автомобили достаточно дорогие, поэтому не все могут себе их позволить. А мобильное приложение позволило бы расширить количество автомобилистов, использующих возможности нейросетей для помощи на дороге. В свою очередь, количество дорожно-транспортных происшествий снизилось бы, потому что электронные системы распознавания не могут отвлечься от дороги на что либо, в отличии от человека.

Широко известно, что подобные системы распознавания сейчас активно разрабатываются.

Компания Tesla использует в своих машинах специальный модуль, который обрабатывает в реальном времени данные, полученные с камер и датчиков, находящихся по периметру автомобиля, распознает объекты на дороге и корректирует поведение машины на дороге в режиме автопилота. Подробнее было рассказано на Tesla Autonomy Day в 2019 году. Понятно, что сложность разработки такой системы крайне высокая, даже сбор и симуляция достаточного количества данных для обучения нейросетей достаточно нетривиальная задача, но машины компании способны собирать данные о поездках, что упрощало разработку системы.

Компания Google в разработках беспилотных автомобилей изначально тоже использовала собственные данные сервиса Google Street View, в котором было накоплено достаточно много данных о дорогах, для обучения нейросетей.

Но разработка таких решений требует огромного вложения денежных средств, и исходный код доступен только разработчикам этих систем, поэтому хотелось бы создать помощника для водителя, которого могли бы использовать не только владельцы машин с автопилотом, а код был бы доступен всем. Помимо этого, распознаванием людей программа не ограничится, современные технологии позволяют так же интегрировать классификатор изображений в мобильное приложение. Чтобы помогать не только автомобилистам, но и пешеходам, можно классифицировать марку и модель машин, про которых человеку интересно было бы узнать, например, проходя мимо. Такая возможность сэкономила бы большое количество времени на поиске вручную в интернете заинтересованному лицу. Так же возможно использование этой функции в коммерческих целях, как это было сделано в мобильном приложении Auto.ru. Там пользователи получали список объявлений по покупке и продаже распознанного автомобиля. Но исходный код программы так же доступен только внутри компании, а в случае ошибки при классификации пользователю приходилось тратить время на переснятие фотографии автомобиля, либо выбора другой фотографии из галереи.

Что касается плюсов мобильных приложений, то они очевидны, ведь смартфоны всегда под рукой, что дает нам неограниченный доступ к важному функционалу. Если разработчик решает распространять приложение бесплатно, тогда единственной преградой может стать операционная система, под которую разрабатывалась программа. Активно развиваются и возможности для разработчиков, потому что компаниям выгодно, чтобы их продукты были уникальными на рынке. Например, американская компания Apple создает, развивает и поддерживает фреймворки для обучения нейросетей и каждый год выпускает крупные обновления своих библиотек.

Постановка задачи

Цель работы - создание мобильного приложения под управлением операционной системы iOS, которое имеет два режима работы: распознавание объектов и классификация объектов. Для достижения этой цели необходимо решить несколько задач, а именно:

В обоих режимах приложение должно использовать камеру смартфона для получения видео в реальном времени для анализа.

В режиме распознавания объектов на входных данных ищутся люди, при нахождении которых приложение обводит их в рамки и издает предупредительный сигнал для пользователя.

В режиме классификации объектов на входных данных классифицируются марки и модели машин, выводя данные по классификации пользователю на экран.

Обзор литературы

Чтобы понять, что лежит в основе нейросетей, можно обратиться к работе Майкла Нильсена [1], в которой подробно рассказывается об архитектуре нейронных сетей, об их устройстве. Также рассматриваются примеры приложений, которые уже созданы.

О подходе к распознаванию объектов, который применяется в разработке мобильных приложений, можно найти информацию в работе Джозефа Редмона и его коллег [2]. Там описан принцип работы системы, ее плюсы и сравнение с другими решениями, а так же примеры получаемых результатов распознавания.

Достаточно обширная теоретическая база находится в книге [3]. Там расписаны различные алгоритмы обучения нейросетей, их математические обоснования и информация по применению их на практике.

Очень подробно рассказано о возможностях использования нейросетей в мобильном приложении в книге [4], авторы показывают на практике как работают фреймворки, поэтому информация воспринимается легче.

Полезная информация также находится в официальной документации от Apple [5], [6], там рассказывается про принцип работы фреймвор-

ков, которые необходимы при работе с нейросетями, и описаны функции, доступные разработчикам. У сайта достаточно удобная навигация, что позволяет проще находить нужный функционал.

В книге [7], помимо теории по нейросетям, на языке программирования Python приведены примеры по работе с инструментами по построению и тренировке нейросетей.

Книга [8] содержит большое количество графиков и иллюстраций к математической теории по машинному обучению, что упрощает восприятие информации. Включает в себя информацию про то, какие существуют разновидности нейронных сетей, как они устроены и где применяются.

Так же хорошо математическая часть расписана в книге [9], подробно изложены практические аспекты глубокого обучения сетей, закрепленные примерами на языке Python.

Глава 1. Искусственные нейронные сети

1.1 Искусственный нейрон

Рассмотрим устройство простейшего искусственного нейрона — персептрона. Он имеет несколько входов, для примера x_1, x_2, x_3. На самом деле, входов может быть больше, либо меньше, и единственный выход *output*. Эта схема представлена на Рис. 1.

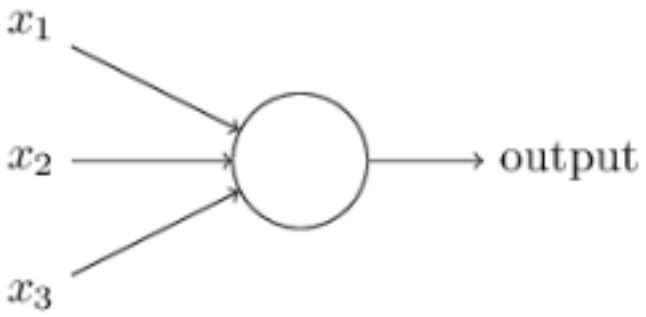

Рис. 1: Схема искусственного нейрона. Изображение из книги [1].

Выход персептрона может иметь два значения 0 или 1, оно вычисляется согласно формуле 1, x_j — некоторые признаки у объекта, значения которых и поступают на соответсвующие входы нейрона, q — это некоторое пороговое значение, а веса w_j — это действительные числа, которые показывают важность соответствующего входа для выхода *output*.

$$output = \begin{cases} 0, & \text{если } \sum_j w_j x_j <= q; \\ 1, & \text{если } \sum_j w_j x_j > q; \end{cases} \tag{1}$$

Так устроен простейший нейрон. На практике, в более сложных моделях нейрона, значение на выходе *output* вычисляется с помощью других функций активации, например сигмоидной по формуле 2, где b — параметр, задаваемый до начала обучения. Такой подход улучшает качество обучения.

$$output = \frac{1}{1 + e^{(\sum_j w_j x_j - b)}} \tag{2}$$

1.2 Сверточные нейронные сети

В многослойном перцептроне, представляющем полносвязную нейронную сеть, каждый нейрон связан с каждым нейроном предыдущего слоя, у каждой связи свой весовой коэффициент.

В сверточной сети, во время выполнения операции свертки, мы проходим по пикселям входного изображения матрицей весов небольшого размера, после каждого сдвига матрицы формируя сигнал активации нейрона следующего слоя. Следующий слой, полученный операцией свертки, формирует карту признаков — ищет закодированные на первом этапе признаки объекта и их координаты. У всех нейронов в одной и той же карте признаков одна и та же матрица весов, именуемая ядром свертки. В сверточной сети множество различных ядер свертки, что позволяет кодировать различные объекты. Каждый проход таким ядром позволяет формировать независимую от других карту признаков.

Веса ядер свертки зачастую настраиваются методом обратного распространения ошибки. Величина среднеквадратичной ошибки рассчитывается, например, по формуле 3, дельта вычисляется по формуле 4.

$$E = \frac{1}{2} * \sum_{j} (t_j - y_j)^2 \tag{3}$$

где E — величина функции ошибки, t_j — желаемый выход нейрона, y_j — действительный выход нейрона.

$$\delta_i = \frac{dy_i}{ds_i} * \sum_{j} \delta_j * w_{ij} \tag{4}$$

где $\frac{dy_i}{ds_i}$ — значение производной функции активации для нейрона j, δ_i — ошибка нейрона i, а δ_j — ошибка для нейрона j из следующего слоя, w_{ij} — вес связи между нейронами i и j.

Достаточно подробно алгоритм и сам процесс обучения расписан в статье [10]. Задача обучения состоит в минимизации функции ошибки между полученным результатом и ожидаемым, для этого мы корректируем весовые коэффициенты.

После операции свертки следует операция подвыборки, представляющая собой уменьшение размерности карт признаков, полученных на предыдущем этапе. Подвергаем данной операции непересекающиеся блоки пикселей, сжимая их, получая матрицу меньшей размерности.

Чтобы уменьшить размерность, чаще всего используется функция максимума, либо берется среднее значение по соседним нейронам. Это позволяет значительно сократить объем изображения и убрать излишнюю подробность признаков.

За несколькими последовательными этапами свертки и подвыборки полученные признаки мы передаем на входы полносвязной нейронной сети, которая и делает окончательные прогнозы по классу исходного объекта. Архитектура сверточной сети представлена на Рис. 2.

Рис. 2: Архитектура сверточной сети. Изображение из статьи [11].

К плюсам сверточных нейронных сетей можно отнести меньшее количество настраиваемых весов, в отличии от полносвязных сетей, ведь тут используется одно ядро свертки для каждой карты признаков, а не для каждого пикселя собственные весовые коэффициенты. Также легко распараллелить вычисления, поскольку в операциях производится многократное умножение матриц. Выделяется и устойчивость классификации к небольшим сдвигам и поворотам исходного изображения.

Минусы сверточных сетей заключаются в большом количестве варьируемых параметров сети, влияющих на результат. Например, сколько слоев свертки и подвыборки использовать для конкретной задачи, какое количество ядер свертки на соответствующем слое свертки, нужно анализировать необходимость слоя подвыборки после слоя свертки.

В данной работе использован фреймворк TuriCreate для создания

нейросетевых моделей, способных классифицировать изображения и определять объекты. Необходимо разобраться, как он работает.

1.3 TuriCreate: классификация изображений

Для классификации изображений фреймворк TuriCreate использует сверточные нейронные сети. Обычно проблема состоит в том, какое количество слоев выбрать, и какими значениями инициализировать изначальные параметры сети. Основная цель инструментария этого фреймворка — уменьшить сложность создания модели, которая подходила бы для классификации большого количества различных классов изображений.

Изначально используется обученная на большом количестве данных сверточная нейронная сеть с множеством различных категорий классификации. Например, может быть использована сеть Squeezenet, обученная отличать 1000 различных категорий.

Можно отметить огромным плюсом этой нейросетевой модели, что при конвертации в модель, которая доступна в мобильной разработке, ее вес на порядок меньше аналогов, например модели, полученной из сети Resnet, которая так же отличает 1000 категорий. В мобильной разработке вес приложения играет важную роль, поэтому при выборе сети на данном этапе это может стать решающим фактором.

Далее удаляется выходной слой этой обученной нейронной сети, чтобы получать сигналы, которые распространялись на удаленный слой, затем передавая их в качестве признаков в классификатор для нашей новой задачи по классификации автомобилей.

Такой подход позволяет избежать настройки вручную множества параметров, отпадает необходимость в ускорении обучения и повышения производительности, даже если недостаточно данных для создания традиционной модели глубокого обучения. Также этот метод эффективен, даже если изначальная сеть не была связана с классификацией объектов, необходимых в произвольной задаче.

1.4 TuriCreate: распознавание объектов

Достаточно долго нейросетевые модели обнаружения объектов разделяли механизмы поиска локации и классификации. Так называемые двухступенчатые детекторы до сих пор дают неплохие результаты, но недавняя работа, описанная в книге [2], объединила эти шаги в единую модель обучения «You Only Look Once»(YOLO), сделав одноступенчатый детектор.

Изначально обучение аналогично описанному в классификации изображений, соответственно нейросеть видела миллионы изображений до того момента, как ей был показан наш датасет. Однако, сеть не была готова к обнаружению объектов, необходимо адаптировать ее для этой задачи.

Это требует сквозной настройки — процесса аккуратного обновления весов для нужной нам задачи, чтобы не потерять изученную ранее полезную информацию. Достаточно длительный процесс, требуется намного больше времени, чем просто на создание модели классификации изображений.

Системы предварительного обнаружения для обнаружения объектов изменяют классификаторы, а здесь используется другой подход. Применяется одна нейросеть для всего изображения, она делит его на регионы и предсказывает координаты рамок с объектом и вероятности для каждого региона. Это показано на Рис. 3.

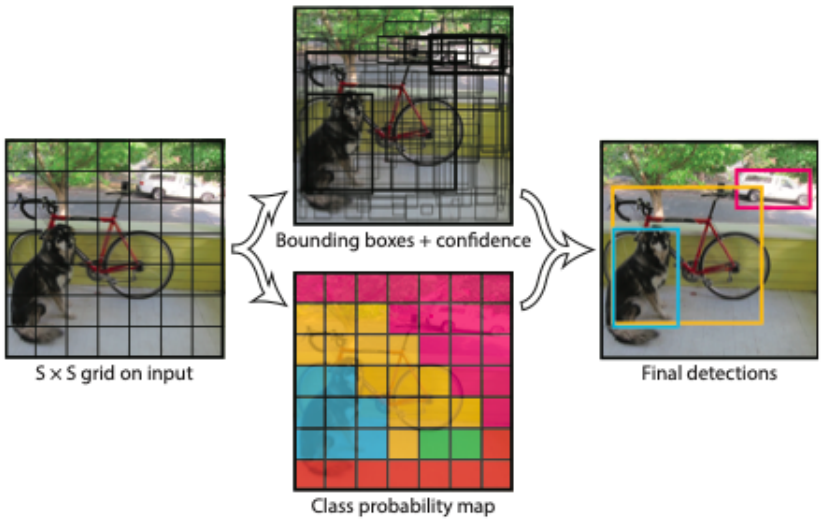

Рис. 3: Работа сети YOLO. Изображение из книги [2].

К плюсам нейросети «YOLO»можно отнести, во-первых, что она очень быстрая, базовая сеть может обрабатывать до 45 кадров в секунду, а быстрая версия сети обрабатывает вплоть до 150 кадров в секунду. Это позволяет обрабатывать потоковое видео, полученное с камер мобильных устройств, в нашем случае под управлением операционной системы iOS, в режиме реального времени.

Во-вторых, изображение просматривается полностью во время обучения и тестирования, поэтому прогнозы зависят от общего контекста изображения. Это позволяет неявно кодировать фоновую информацию классов и их внешний вид.

В-третьих, «YOLO»изучает обобщенные представления рассматриваемых объектов, что позволяет при обучении на изображениях из жизни, а тестировании на художественных произведениях, превосходить другие существующие системы обнаружения объектов.

К минусам можно отнести, что в общем она пока еще отстает по точности от других современных систем обнаружения.

1.5 TuriCreate: расчет точности

В главе 3 рассчитывается точность обученной нейросети с помощью средств фреймворка TuriCreate.

Точность *accuracy* — это отношение между общим количеством прогнозов и количеством правильных прогнозов. При этом, правильные прогнозы рассматриваются для каждого класса одинаково.

$$accuracy = \frac{\text{Количество правильных прогнозов}}{\text{Общее количество прогнозов}}$$

Также будут использоваться термины $training\_accuracy$ — точность предсказания нейросети на изображениях обучающей выборки и $validation\_accuracy$ — точность предсказания на изображениях тренировочной выборки. Результаты $validation\_accuracy$ нужны не для пересчета весов, а чтобы алгоритм не зацикливался на менее важных параметрах, например на цвете фона или освещении.

Глава 2. Реализация мобильного приложения

Реализация состояла из нескольких этапов: изначально необходимо было получить нейросетевую модель, которая была бы способна классифицировать изображения, затем получить модель, которая распознавала бы объекты, возвращая их координаты, и последним шагом была непосредственная разработка мобильного приложения.

2.1 Создание модели классификации изображений

Для создания нейросетевой модели использован фреймворк от компании Apple — TuriCreate. Он позволяет создать и обучить нейросеть, а затем конвертировать ее в модель, которую можно использовать в мобильном приложении. Работа с фреймворком происходила на языке программирования Python. Изначально пропускаем картинки нашего датасета с машинами через фреймворк с помощью функции $load\_images$:

```
carsData = turicreate.image_analysis
                    .load_images('dataset', with_path=True)
```

$dataset$ — это название папки, в которой лежат картинки. Важно отметить, что внутри этой папки картинки дополнительно разделены по папкам, названия которых характеризуют модель, год презентации и марку этой модели машины. Это необходимо для того, чтобы нейросеть при обучении знала, какие машины содержатся в этой папке, проставляем им класс.

Параметр $with\_path$, выставленный в значение $true$, помогает запомнить названия папок с классами машин в данных, которые будут сохраняться для дальнейшей работы в функции $save$. Входным аргументом указываем желаемое название выходного файла и расширение $sframe$, с которым работает данный фреймворк.

```
carsData.save('carsData.sframe')
```

$carsData.sframe$ — это полученные табличные данные, которые хранят в себе информацию об исходных картинках и присвоенных нами им

классах, а так же могут хранить другую полезную информацию. Наглядное представление полученных данных на Рис. 4

	path	image	car_model
79	dataset/Toyota Camry Sedan 20...		Toyota Camry Sedan 2012
80	dataset/Toyota Camry Sedan 20...		Toyota Camry Sedan 2012
81	dataset/Toyota Camry Sedan 20...		Toyota Camry Sedan 2012
82	dataset/Toyota Camry Sedan 20...		Toyota Camry Sedan 2012
83	dataset/Volkswagen Golf Hatchb...		Volkswagen Golf Hatchback 2012
84	dataset/Volkswagen Golf Hatchb...		Volkswagen Golf Hatchback 2012
85	dataset/Volkswagen Golf Hatchb...		Volkswagen Golf Hatchback 2012

Рис. 4: Визуализация обработки данных.

Далее используем функцию *create*, которая и создает нейросетевую модель.

```
model = turicreate.image_classifier
                  .create(train_carsData,
                          target='car_model',
                          model = 'squeezenet_v1.1',
                          max_iterations=500)
```

Она имеет множество входных аргументов, которые позволяют настроить обучение, основные из них:

$train\_carsData$ — некоторое количество данных от общего количества для обучения нейросети. Я использовал для обучения 80% от всех данных. $target$ — это множество классов в наших данных $carsData.sframe$.

model — нейросеть, которая используется для обучения нашей нейросети. *max_iterations* — максимальное количество обучающих итераций.

После этого конвертируем полученную нейросетевую модель *model* в CoreML модель, которую можно использовать в мобильной разработке.

Эта модель при каждом запросе на анализ изображения от приложения возвращает предполагаемый класс объекта и вероятность данного предсказания.

2.2 Создание модели распознавания объектов

Для создания нейросетевой модели распознавания объектов так же использован фреймворк TuriCreate. Изначальная обработка изображений аналогичная, только добавляется этап, помогающий определить координаты объектов (людей) в датасете, состоящем из фотографий с людьми и изображениями, на которых только маска объекта, выделенная определенным цветом. Достаточно пройтись по пикселям маски, чтобы определить ее границы на исходном изображении.

	image	name	label	annotations
8		person_008	person	[{"label":"person","coordinates":{...
9		person_006	person	[{"label":"person","coordinates":{...
10		person_018	person	[{"label":"person","coordinates":{...
11		person_036	person	[{"label":"person","coordinates":{...
12		person_013	person	[{"label":"person","coordinates":{...

Рис. 5: Визуализация обработки данных.

В итоге, можем добавить в наше табличное представление данных столбец, содержащий координаты объектов, которые необходимо будет распознавать. Это показано на Рис. 5.

Следующим шагом идет создание нейросетевой модели, для этого использована функция *create*.

```
model = tc.object_detector
          .create(train_peopleData,
                  feature='image',
                  annotations='annotations',
                  max_iterations=6000)
```

train_peopleData — определенная часть данных от датасета для обучения нейросети. Я использовал 80% от общего количества.

feature — имя колонки с нашими изображениями.

annotations — имя колонки с координатами объектов.

max_iterations — максимальное количество обучающих итераций.

Также доступны и другие параметры для конфигурации обучения нейросети. После обучения конвертируем полученную нейросетевую модель в CoreML модель, которую можно использовать в мобильной разработке.

Эта модель при каждом запросе на поиск объекта на изображении пытается найти объект, в случае успеха на выход передаются координаты найденного объекта и вероятность, с которой распознали этот объект.

2.3 Разработка мобильного приложения

Мобильное приложение разработано с использованием архитектуры программирования «Model View Presenter»(MVP).

В качестве View, слоя представления интерфейса пользователю, в приложении используются два класса: *BaseViewController* и *RecognizerViewController*.

Presenter — это *RecognizerPresenter*. Он отвечает за общение с нейросетевыми моделями, созданными на предыдущих шагах разработки, и передает модели для отображения во View слой.

Наглядно архитектура показана на Рис. 6

Рис. 6: Архитектура мобильного приложения.

Для захвата видео с камеры устройства, использован фреймворк $AVFoundation$. В классе $BaseViewController$ настроены параметры сессии, в рамках которой получаем видео с камеры. Например, установлены параметры вывода видео на экран.

Класс $RecognizerViewController$ включает реализацию протокола $AVCaptureVideoDataOutputSampleBufferDelegate$, содержащий функцию $captureOutput$, в которую системой отдаются полученные кадры с камеры в специальном формате данных $CMSampleBuffer$.

После получения кадров, они передаются в функцию презентера $videoDidCaptured$, где определяется ориентация девайса, чтобы далее вместе с кадрами камеры в формате $CMSampleBuffer$ создать обработчик запросов к нейросетевым моделям $VNImageRequestHandler$.

Этот обработчик из библиотеки Vision, которая предоставляет инструменты для общения с нейросетевыми моделями. Сами запросы к моделям представлены классом $VNRequest$. Выполняем запросы с помощью функции $perform$.

В случае определения марки машины, презентер, в функции $drawImageClassificetionResult$, подготавливает из полученных от модели данных текст с информацией о машине с вероятностью прогноза, а так же логотип марки машины, если он был загружен для данной марки, и передает их в $RecognizerViewController$.

В режиме распознавания людей, презентер, при получении от модели координат найденного объекта, в функции $drawObjectDetectionResult$ обрабатывает координаты и передает их в $RecognizerViewController$ для

отображения.

Далее отображаем полученные данные: в режиме определения марки машины размещаем текстовые поля и иконку марки машины так, чтобы это было видно пользователю на экране. А в режиме распознавания людей добавляем на экран рамку с границами объекта и воспроизводим звуковой сигнал с помощью функции $playSound$.

Переключаться между режимами распознавания помогает кнопка, расположенная внизу экрана мобильного устройства.

Исходный код программы доступен по ссылке [14].

Глава 3. Вычислительный эксперимент

3.1 Описание датасетов

Для данной работы было использовано два датасета с фотографиями.

Первый — «Stanford Cars Dataset»[12], состоит из 16185 фотографий и включает в себя 196 моделей машин. Машины различаются так же по году модели, марке модели и названиям моделей. Фотографии машин одной марки и модели различались ракурсами и цветами автомобиля, а также качеством фотографий. Примеры фотографий видны на Рис. 4.

Второй — «IG02»[13], включает в себя фотографии людей, машин и велосипедов. Для данной работы использовалась часть датасета, содержащая людей. Датасет содержал 311 фотографий людей, а так же в зависимости от количества людей на фото от 1 до 12 масок, на которых люди были выделены красным цветом. Места, где человека перекрывали другие люди или же предметы, были окрашены в зеленый цвет. Пример представлен на Рис. 7.

Рис. 7: Пример фотографии и маски к ней из датасета IG02

В рамках работы каждый датасет делили на две выборки: обучающую и тренировочную. Обучающая выборка — 80% от всего датасета, тренировочная — 20%.

3.2 Классификация моделей машин

Для обучения нейросети, классифицирующей модели машин, была использована нейросеть $squeezenet\_v1.1$, а не аналог $resnet$, потому что после конвертации необходимая нам модель занимала около 5Мб, а если использовать вторую сеть, размер модели будет около 90Мб, что в 18 раз больше, а для мобильной разработки вес приложения — это важный параметр. Используем $squeezenet\_v1.1$ в параметре функции создания модели $model$. Исходя из вычислительных возможностей системы, на которой происходило обучение сети, параметр $batch\_size$ (количество изображений за итерацию) в функции создания модели был установлен в значение 64.

На рисунках 8, 9, 10 представлены результаты обучения программы для различного значения параметра $max\_iterations$ в методе обучения.

Iteration	Passes	Step size	Elapsed Time	Training Accuracy	Validation Accuracy
90	126	1.000000	405.453709	0.856290	0.332031
91	127	1.000000	409.122987	0.862467	0.320312
92	128	1.000000	412.762517	0.869055	0.324219
93	129	1.000000	416.501778	0.878320	0.328125
94	130	1.000000	420.292653	0.880791	0.343750
95	131	1.000000	423.939871	0.883055	0.343750
96	133	0.799744	429.845597	0.887173	0.343750
97	134	0.999680	433.549657	0.893556	0.343750
98	135	1.000000	437.438808	0.896438	0.343750
99	136	1.000000	441.446923	0.899526	0.347656

Рис. 8: Результаты обучения, max_iterations = 100

Iteration	Passes	Step size	Elapsed Time	Training Accuracy	Validation Accuracy
489	606	1.000000	2218.953096	0.999379	0.388235
490	608	0.965190	2224.702615	0.999379	0.388235
491	610	0.965190	2230.474367	0.999379	0.392157
492	611	1.000000	2234.014908	0.999379	0.392157
493	612	1.000000	2237.600650	0.999379	0.392157
494	613	1.000000	2241.237608	0.999379	0.388235
495	614	1.000000	2245.139735	0.999379	0.388235
496	616	0.834455	2250.896301	0.999379	0.388235
497	617	1.000000	2254.539592	0.999379	0.388235
498	618	1.000000	2258.456196	0.999379	0.388235
499	620	0.965621	2264.333978	0.999379	0.388235

Рис. 9: Результаты обучения, max_iterations = 500

Где $training\_accuracy$ — точность предсказания нейросети на изображениях обучающей выборки, а $validation\_accuracy$ — точность предсказания на изображениях тренировочной выборки.

На рисунках видно, что при 100 итерациях активно меняются результаты точности предсказаний, в отличии от 500 и более итераций.

Iteration	Passes	Step size	Elapsed Time	Training Accuracy	Validation Accuracy
988	1271	1.089652	4164.937824	0.999175	0.378906
989	1273	1.003920	4170.664295	0.999175	0.378906
990	1275	0.745416	4176.430569	0.999175	0.378906
991	1277	0.745416	4182.123202	0.999175	0.378906
992	1278	0.931771	4185.695491	0.999175	0.378906
993	1280	1.000000	4191.455960	0.999175	0.378906
994	1281	1.000000	4195.021009	0.999175	0.378906
995	1282	1.000000	4198.560656	0.999175	0.378906
996	1284	0.814123	4204.378189	0.999175	0.378906
997	1285	1.000000	4208.018681	0.999175	0.378906
998	1286	1.000000	4212.078426	0.999175	0.378906
999	1287	1.000000	4215.893047	0.999175	0.378906

Рис. 10: Результаты обучения, max_iterations = 1000

Зависимость точности предсказания от количества итераций представлена на Рис. 11.

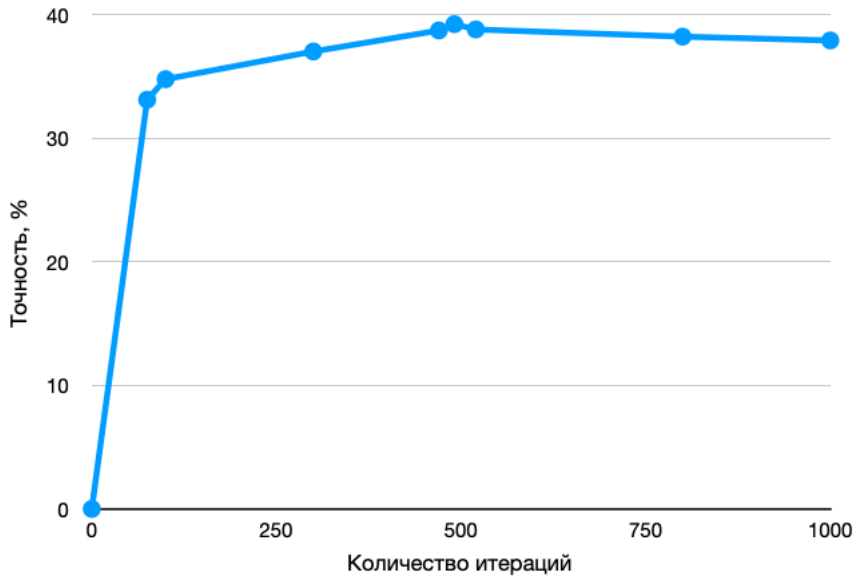

Рис. 11: График зависимости точности предсказания от количества итераций

Оптимальное количество итераций — 491, потом происходит переобучение модели и теряется точность предсказаний. Лучшая точность на тренировочной выборке — 0.392157.

На Рис. 12 и 13 представлена работа приложения в режиме распознавания машин в реальном времени.

По результатам работы, приложение с точностью более 80% распознавало марки автомобилей. Причем, если в датасете отсутствовала старая модель машины некоторого модельного ряда, присутствующего в датасете, нейросетевая модель правильно определяла марку и этот модельный ряд.

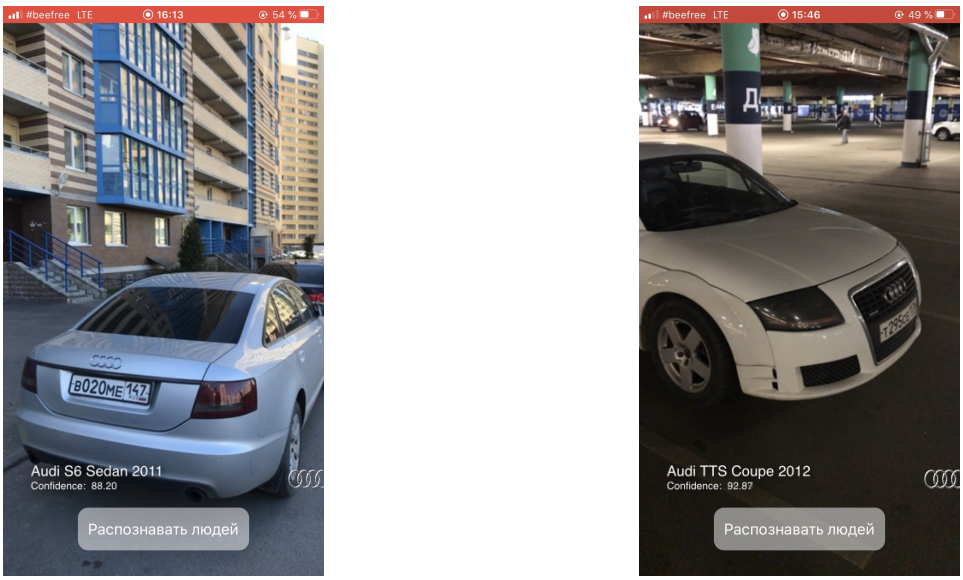

Рис. 12: Примеры работы приложения в реальном времени.

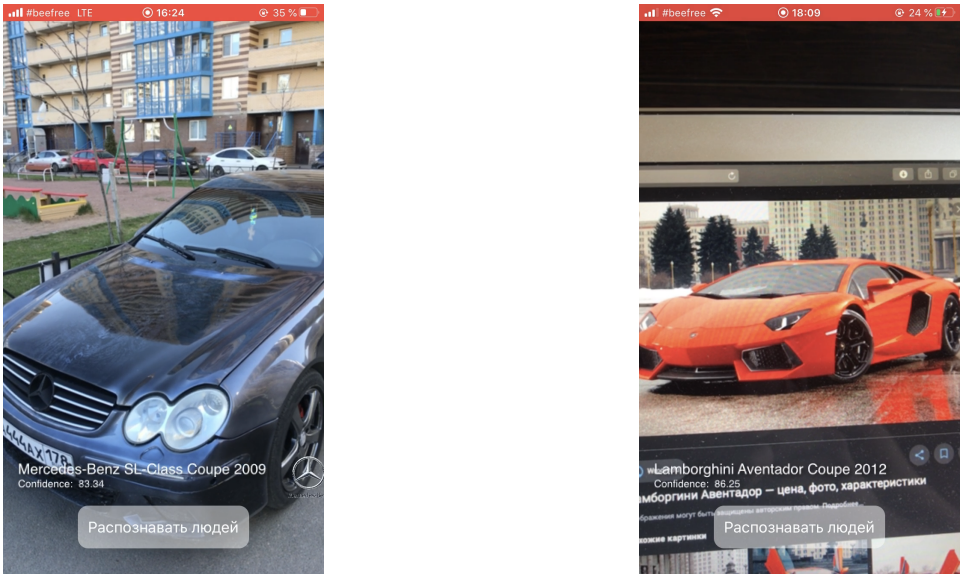

Рис. 13: Примеры работы приложения в реальном времени.

3.3 Распознавание людей

Для создания нейросетевой модели использована нейросеть *darknet–yolo*.

Рисунки 14, 15, 16 показывают, как меняется параметр *loss*, являющийся разницей между предсказанием нейросети и действительностью.

После каждого обучения производился запуск функции фреймворка *predict*, с помощью которой предсказывались координаты рамок людей на

тестовой выборке, и *evaluate*, которая сравнивала эти значения с известными и выводила точность.

Исходя из вычислительных возможностей системы, на которой происходило обучение сети, параметр *batch_size* (количество изображений за итерацию) в функции создания модели был установлен в значение 64. Параметр *grid_shape* (размер сетки, используемой для обнаружения объекта) - это матрица, размер выбран 13x13 так же опираясь на возможности системы. Чем больше этот параметр, тем лучше определяются маленькие объекты, но так же возрастают и вычислительные затраты.

Iteration	Loss	Elapsed Time
2926	2.36141	2h 17m
2931	2.48047	2h 17m
2936	2.56477	2h 17m
2941	2.50519	2h 17m
2946	2.38334	2h 18m
2951	2.3268	2h 18m
2956	2.28853	2h 18m
2961	2.31544	2h 18m
2966	2.42953	2h 18m
2971	2.24449	2h 19m
2976	2.2474	2h 19m
2981	2.18615	2h 19m
2986	2.17112	2h 19m
2991	2.34859	2h 20m
2996	2.32294	2h 20m
3000	2.32237	2h 20m

Рис. 14: Процесс обучения, max_iterations = 3000

Точность определения рамок на тестовой выборке: 0.13916

Iteration	Loss	Elapsed Time
5946	2.23	4h 38m
5951	2.38083	4h 39m
5956	2.38277	4h 39m
5961	2.33622	4h 39m
5966	2.30168	4h 39m
5971	2.26033	4h 39m
5976	2.43886	4h 40m
5981	2.31517	4h 40m
5986	2.4131	4h 40m
5991	2.41339	4h 40m
5996	2.34782	4h 41m
6000	2.285	4h 41m

Рис. 15: Процесс обучения, max_iterations = 6000

Точность определения рамок на тестовой выборке: 0.35956

Iteration	Loss	Elapsed Time
11941	2.22559	9h 16m
11946	2.13533	9h 16m
11951	2.22817	9h 16m
11956	2.11994	9h 16m
11961	2.12291	9h 17m
11966	2.14325	9h 17m
11971	2.13324	9h 17m
11976	2.16136	9h 17m
11981	2.12885	9h 18m
11986	2.14375	9h 18m
11991	2.05333	9h 18m
11996	2.04471	9h 18m
12000	2.0117	9h 18m

Рис. 16: Процесс обучения, max_iterations = 12000

Точность определения рамок на тестовой выборке: 0.00893

Оптимальным параметром было выбрано 6000 итераций, если использовать больше, то происходит переобучение нейросети, если меньше, то недообучение нейросети. Пример работы приложения на Рис 17, 18.

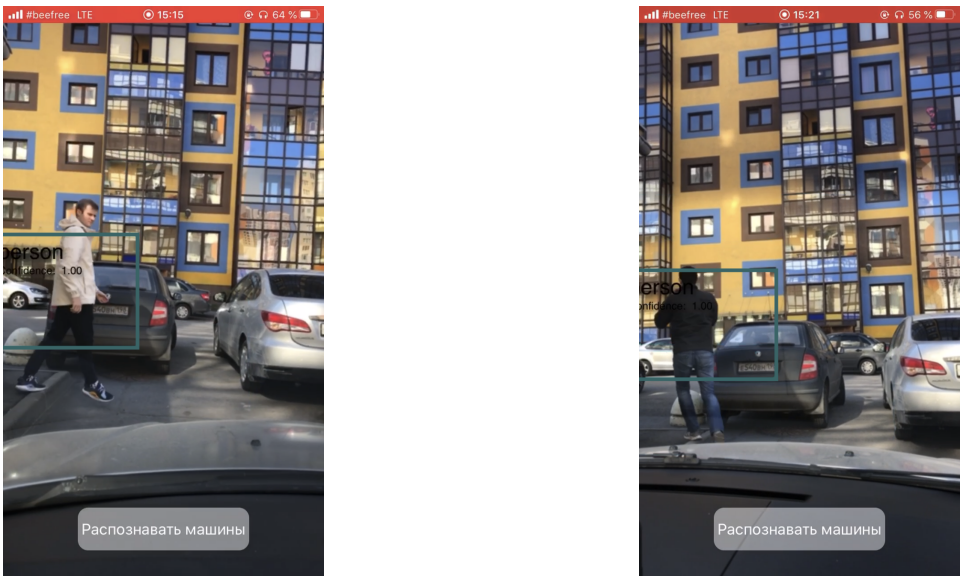

Рис. 17: Примеры работы приложения в реальном времени.

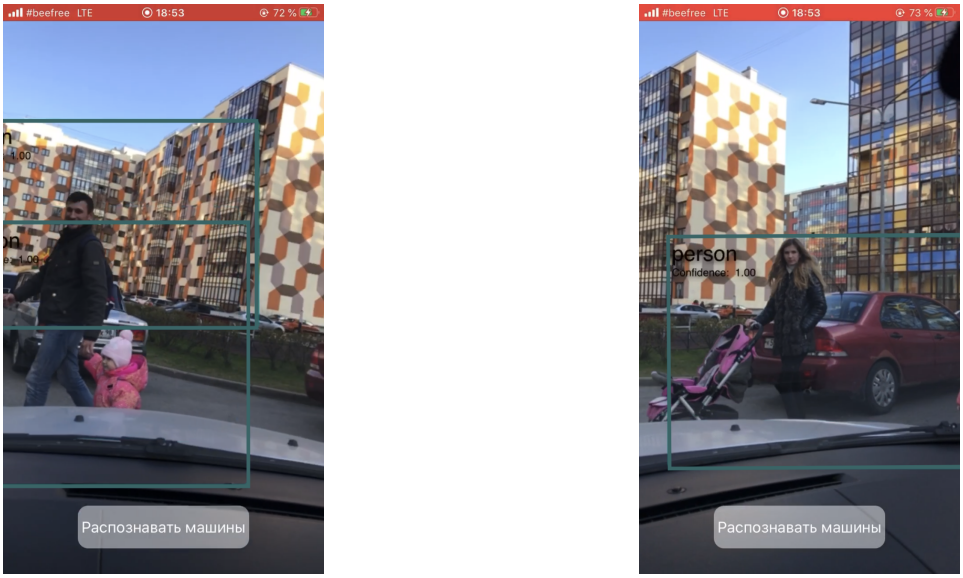

Рис. 18: Примеры работы приложения в реальном времени.

Таким образом, приложение определяет людей в кадре и рисует рамки, но рамки получаются не совсем по фигуре людей, что подтверждает тестовую точность рамок 0.35956.

Заключение

В данной работе создано мобильное приложение, которое классифицирует модели машин и распознает людей в реальном времени.

Оно может использоваться в качестве помощника водителю на дороге, предупреждая его в случае обнаружения человека, или же помогать определить интересующую модель машины.

Такая система может быть доступна всем людям, у которых операционная система на телефоне iOS с версией 11.0 и новее. Она бесплатна и требует только скачивания на телефон.

Список литературы

[1] Michael Nielsen «Neural Networks and Deep Learning»2016

[2] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi «You Only Look Once: Unified, Real-Time Object Detection»2015

[3] Ian Goodfellow Yoshua Bengio Aaron Courville «Deep Learning (Adaptive Computation and Machine Learning series)»2016

[4] Paris Buttfield-Addison, Jonathon Manning, Tim Nugent, Mars Geldard «Practical Artificial Intelligence with Swift From Fundamental Theory to Development of AI-Driven Apps»2019

[5] CoreML Framework
URL: https://developer.apple.com/documentation/coreml

[6] Vision Framework
URL: https://developer.apple.com/documentation/vision

[7] Микелуччи У. «Прикладное глубокое обучение. Подход к пониманию глубоких нейронных сетей на основе метода кейсов.»пер. с англ. СПб.: БХВ-Петербург, 2020.

[8] Жерон О. «Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow»/ пер. с англ. СПб.: ООО Альфа-книга, 2018

[9] Паттанаяк С. «Глубокое обучение и TensorFlow для профессионалов. Математический подход к построению систем искусственного интеллекта на Python»пер. с англ. СПб.: ООО "Диалектика 2019.

[10] Статья по обучению сверточной нейронной сети
URL: https://habr.com/ru/post/348028/

[11] Землевский Андрей Дмитриевич «ИССЛЕДОВАНИЕ АРХИТЕКТУРЫ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ ДЛЯ ЗАДАЧИ РАСПОЗНАВАНИЯ ОБРАЗОВ»

[12] Датасет с автомобилями «Stanford Cars Dataset»
URL: https://ai.stanford.edu/ jkrause/cars/car_dataset.html

[13] Датасет с людьми «INRIA Annotations for Graz-02»
URL: https://lear.inrialpes.fr/people/marszalek/data/ig02/

[14] Ссылка на исходный код разработанной программы
URL: https://github.com/revoltMoon/Diplom