**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**
**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**
**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Департамент программной инженерии

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»,
профессор департамента программной
инженерии, канд. техн. наук

_____ В.В. Шилов
«___» _____ 2018 г.

# Выпускная квалификационная работа

на тему «Сервис распознавания лиц для идентификации личности»

по направлению подготовки 09.03.04 «Программная инженерия»

| | |
|---|---|
| Научный руководитель<br>Профессор департамента программной инженерии, д.т.н.<br>Д.В. Александров<br>_____<br>Подпись, Дата | Выполнил<br>студент группы БПИ 143<br>4 курса бакалавриата<br>образовательной программы<br>«Программная инженерия»<br>Н.О. Константиновский<br>_____<br>Подпись, Дата |

**Москва 2018**

# Реферат

В работе рассматривается архитектура, функционал и технологии используемые в сервисе для распознавания лиц.

Сервис спроектирован для интеграции в мобильное приложение на платформе iOS в качестве фреймворка. Это сделано для того, чтобы уменьшить время и стоимость распознавания лиц, за счет снижения нагрузки на интернет траффик и реализации большинства вычислений на мобильном устройстве.

Сервис использует глубокую сверточную нейронную сеть для кодирования лиц людей. В данном сервисе используется клиент-серверная архитектура. Сервер хранит базу данных с лицами и производит поиск по коду лица. В результате, в данной работе демонстрируется подходы и модели для разработки сервиса.

Работа содержит 42 страницы, 3 главы, 8 рисунков, 37 источников, 8 приложений.

**Ключевые слова:** сверточные нейронные сети; распознавание лиц; клиент-серверные системы; мобильные приложения

# Abstract

This document presents the architecture, functionality and technologies used in a service for facial recognition.

The service is designed to be integrated as a framework into a mobile application on the iOS platform. The service is designed to reduce the time and cost of using face recognition services, by reducing the load on Internet traffic and conducting most of the process for recognizing faces on a mobile device.

The service architecture is used a deep neural network to encode the detected faces. The client-server architecture is required to determine the person. The server stores a database with face codes and a classifier that determines the person. As a result, we demonstrate the service architecture and the opportunities for approaches and models of machine learning to develop a service for face recognition.

The paper contains 42 pages, 3 chapters, 8 illustrations, 37 bibliography items, 8 appendices.

**Keywords:** convolutional neural networks; face recognition; client-server systems; mobile applications

# Основные определения, термины и сокращения

| Термин | Описание |
|---|---|
| Неронная сеть | Математическая модель, а также ее программное воплощение, построенное по принципу биологических нейронных сетей. |
| Сверточная нейронная сеть | Специальная архитектура нейронных сетей, нацеленная на эффективное распознавание изображений. |
| CoreML | Фреймворк для интеграции моделей машинного обучения в iOS устройства, представленный Apple в 2017 году. |
| SVM | Набор алгоритмов машинного обучения с учителем, предназначенных для классификациии и регрессионного анализа. |
| API | Набор готовых функций предоставляемых приложением для использования во внешних программах. |

# Содержание

# Введение

Технологии распознавания лиц в текущий момент активно развиваются и применяются в различных сферах. Технология распознавания лиц позволяет идентифицировать личность людей на фотографии или видео и использовать эту информацию для различных целей. Использование этой технологии в бизнесе позволяет маркетологам наиболее детально изучать своих клиентов, и делать персонализированные рекламные предложения, что увеличивает конверсию рекламной кампании [20]. Эксперты считают, что подход к персонализации через распознавание клиента может значительно повысить эффективность персонализированной рекламы, а также аналитики прогнозирования поведения клиентов, что уже применяется в крупнейших розничных магазинах, например Walmart [22].

В настоящий момент такие системы распознавания лиц разрабатываются в основном как облачные решения [33], что увеличивает использование интернет-трафика, а также снижает скорость обработки при медленном интернет соединении, поскольку в реальном времени необходимо передавать фотографии или видеоизображения с камеры. Однако, с увеличением вычислительных мощностей мобильных устройств стало возможным интегрировать необходимые модели нейронных сетей, предназначенных для извлечения признаков лица, в мобильное устройство, оставив на серверной части только классификацию изображений и хранение базы данных с распознанными людьми. В случае же небольших баз данных и, соответственно, уменьшении размера классификаторов, можно проводить вычисления исключительно на мобильном устройстве. Данный подход позволяет снизить использование интернет-трафика за счет того, что по сети передаются признаки лица, представленные в виде вектора, а не изображение целиком. В случае же s расположения классификатора и базы данных, можно полностью избежать использование интернет сети. Тем не менее, среди самых популярных сервисов распознавания лиц, только Face++ [18] предоставляет функционал SDK для мобильных устройств, который позволяет определять, сравнивать и отслеживать лица, однако, не предоставляет функционала для распознавания и идентификации личности, без использования интернета. Остальные самые популярные сервисы предоставляют только API, который позволяет отправить фото или, в некоторых случаях, видео и получить результат распознавания в ответ на запрос.

Таким образом, разработка сервиса, предоставляющего функционал по распознаванию лиц, выполняющий большинство или всю вычислительную работу на мобильном устройстве, могла бы существенно уменьшить цену использования такого сервиса, а также, позволило бы использовать сервис в местах с ограниченным интернет трафиком. Это позволило бы использовать такой сервис с существенно более низкими затратами, чем аналогичные сервисы, которые разрабатывались как облачные решения.

В рамках данной выпускной квалификационной работы была поставлена **цель** – разработать сервис распознавания лиц для идентификации личности, выполняющий вычисления по извлечению признаков лица на мобильном устройстве на платформе iOS и предоставляющий возможность идентифицировать личность в локальной или удаленной базе данных.

Для достижения цели были поставлены следующие **задачи** выпускной квалификационной работы:

1. Изучить подходы распознавания лиц;
2. Выбрать наиболее подходящий подход распознавания лиц для реализации на мобильном устройстве;
3. Изучить подходы поиска личности по лицу в базе данных;
4. Выбрать наиболее подходящий подход для поиска личности по лицу в базе данных для реализации на мобильном устройстве;
5. Разработать необходимые для реализации сервиса модели для распознавания лиц на языке Python;
6. Разработать серверную часть сервиса распознавания лиц для идентификации личности на языке Python;
7. Разработать API сервиса;
8. Разработать SDK сервиса для платформы iOS;
9. Разработать тестовое iOS приложение, использующее SDK сервиса, демонстрирующее работу сервиса распознавания лиц для идентификации личности.

Исходя из поставленных задач была определена следующая структура работы:

1. В главе 1 приведен обзор подходов распознавания лиц;
2. В главе 2 описано проектирование сервиса;
3. В граве 3 описана техническая реализация сервиса;
4. В заключении перечислены основные полученные результаты и пути дальнейшей работы для усовершенствования сервиса;
5. В приложениях приведена техническая документация сервиса.

# Глава 1. Обзор подходов распознавания лиц

В данной главе проводится обзор существующих подходов к распознаванию лиц. Производится анализ подходов с точки зрения точности распознавания, а также вычислительной сложности, поскольку выбранный подход будет использоваться на мобильном устройстве, у которого вычислительные мощности ограничены.

## 1.1. Общий подход к распознаванию лиц

На данный момент существует большое разнообразие алгоритмов и подходов распознавания лиц, однако у всех у них есть типовые шаги, которые присутствуют в любом подходе [17]. Эти шаги (рис. 1.1) являются базовыми для всех подходов, однако в разных подходах отличаются алгоритмы реализации этих шагов.

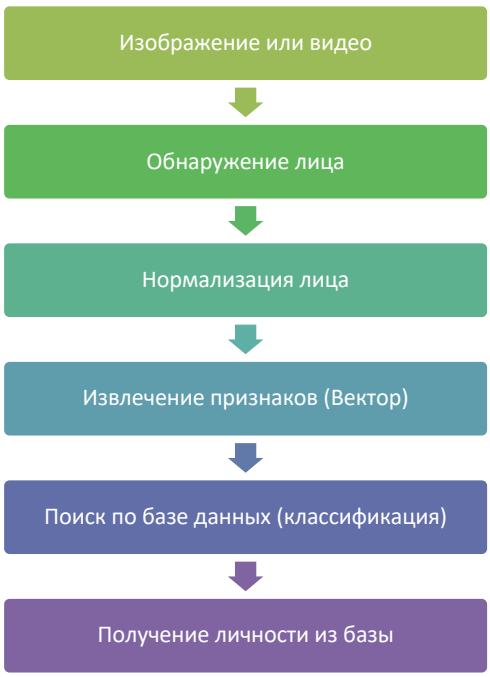


Рисунок 1.1. Типовые шаги распознавания лица

Процесс начинается с получения фото или видео изображения. После этого на изображении или в видео обнаруживаются лица и дальнейшая работа ведется только с фото или видео изображением лица. Затем изображение нормализуется и передается на следующий шаг. На шаге извлечения признаков происходит извлечение признаков нормализованного изображения лица. Эти признаки представляются в виде вектора. Полученный вектор характеризует лицо. Далее, по полученному вектору происходит поиск в базе данных, целью которого является нахождение лиц, близких по значению к вектору. Теперь рассмотрим каждый из шагов подробнее.

### 1.1.1. Обнаружение лица

Существует большое количество алгоритмов обнаружения лиц на фотографии, однако, на данный момент, наиболее универсальным и точным [15] алгоритмом является алгоритм обнаружения лица по гистограмме направленных градиентов [21]. Данный подход для обнаружения лиц является наиболее подходящим и используется сейчас повсеместно.

### 1.1.2. Нормализация лица

Лицо на изображении может находиться в разных ракурсах. Для увеличения точности распознавания необходимо добиться того, чтобы такие признаки лица, как нос, рот, глаза находились примерно в одном и том же месте на всех фотографиях [14], этот процесс называется нормализацией. Для обнаружения признаков лица применяется алгоритм использующий регрессионные деревья [29]. После этого применяется алгоритм для нормализации лица [16], для этого к изображению применяются аффинные преобразования.

### 1.1.3. Извлечение признаков (кодировка лица)

После того как лицо нормализовано, из него можно извлечь признаки или, как говорят иначе, кодировка лица. Для извлечения признаков используются различные алгоритмы, сравнение которых будет проведено далее, однако, это ключевой шаг для идентификации лица, получение признаков лица, представляемые вектором, который характеризует искомое лицо.

### 1.1.4. Поиск по базе данных

На данном шаге необходимо найти в базе данных человека, характеристики лица которого наиболее близки к характеристике лица, закодированного на предыдущем шаге. Это задача мульти-классовой классификации. Существуют также различные алгоритмы, при помощи которых можно произвести классификацию, сравнение этих алгоритмов будет проведено далее.

## 1.2. Алгоритм кодировки лица

Для кодировки и распознавания лица используется множество различных алгоритмов. Существует исследования [1, 11], по итогам которых наиболее точным алгоритмом для распознавания лиц становился алгоритм PCA, метод главных компонент. Так же отмечалась высокая точность применения нейронных сетей для данной задачи, однако у подхода, который использовал полно-связанный слой как последний, для классификации и, соответственно идентификации личности, есть проблема потери точности при увеличении количества классов,

а также необходимость переобучать нейронную сеть каждый раз, когда в базу добавляется код нового лица [2].

В 2015 году исследователями из Google был предложен иной подход для распознавания лиц с использованием сверточной нейронной сети [19]. В данном подходе не обучается полно-связанный слой нейронной сети для классификации. Вместо этого нейронная сеть выдает 128-размерный вектор, который является кодировкой изображения лица (рис. 1.2). Нейронная сеть обучается таким образом, что разные фотографии одного человека будут кодироваться векторами, Евклидово расстояние между которыми меньше, чем между кодировками фотографий того же человека и другого человека. Нейронная сеть обучается таким образом благодаря использованию функции триплет потери, которая во время обучения получает результаты прохода через нейронную сеть двух фотографий одного человека (якорное изображение и позитивное изображение), а также фотографии другого человека (негативное изображение). Далее веса нейронной сети корректируются так, чтобы расстояние между кодами якорного и позитивного изображения стало меньше, чем между кодами якорного и негативного (рис. 1.3).

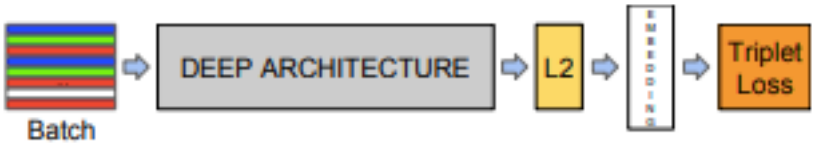


Рисунок 1.2. Структура модели


Рисунок 1.3. Триплет потеря

Плюсы данного подхода в том, что при помощи нейронной сети можно получить 128 характеристик лица, задающихся числами, которые в последствии можно использовать как код, уникальный для каждого человека. Таким образом, в случае если дальнейшая классификация будет происходить на серверной части, передача на сервер 128 чисел значительно менее ресурсоемкий процесс, чем передача изображения, что уже дает возможность экономии интернет-трафика и увеличения скорости распознавания. Поэтому в дальнейшем будем использовать этот алгоритм.

## 1.3. Алгоритм классификации

На выходе нейронной сети мы получаем 128 характеристик лица человека. После этого необходимо в базе данных известных лиц найти человека, характеристики которого наиболее близки к характеристикам входного изображения. Для такой задачи применяется алгоритмы классификации. Мы рассматриваем алгоритмы классификации не только с точки зрения точности, но и с точки зрения производительности, а также масштабируемости под большое количество классов.

Алгоритмов классификации достаточно много и сказать какой лучше использовать без исследования под конкретную задачу невозможно. В статье исследователей из Google для задачи классификации использовался SVM классификатор [32]. Изучение и сравнение алгоритмов классификации для данной задачи тема отдельного исследования, поэтому в данном случае будет использоваться подход, описанный в оригинальной статье с использованием SVM классификатора.

Изначально SVM классификатор создавался как бинарный классификатор [32]. В случае мульти-классовой классификации используют различные способы для классификации с помощью SVM [27]. В сравнении с другими подходами наиболее точным [3] в большинстве случаев получается подход one-against-one [23]. Однако, у данного подхода есть существенный недостаток, так как, например для $N$ классов необходимо $N(N-1)/2$ бинарных SVM классификаторов. Также вырастает вычислительная сложность, так как все эти классификаторы должны участвовать в принятии решения.

Существует подход, строящийся на архитектуре бинарного дерева и использующий бинарные SVM классификаторы для принятия бинарного решения [31]. Архитектура данного подхода представляет собой бинарное дерево, в узлах которого решение принимаем SVM классификатор (рис. 1.4).

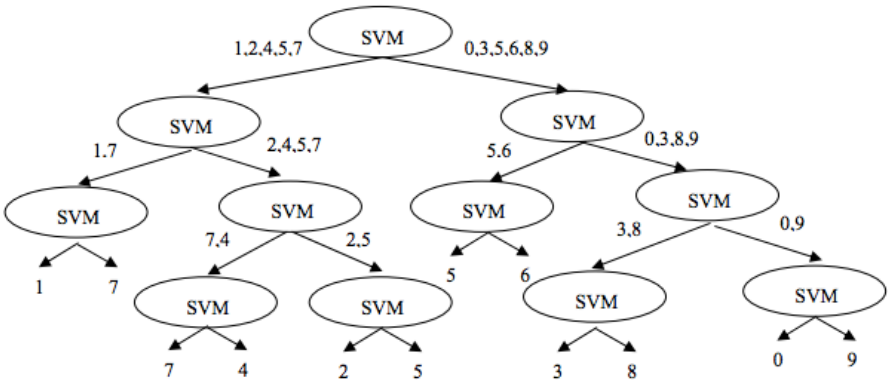


Рисунок 1.4. Архитектура SVM классификатора основанного на бинарном дереве

В случае выбора алгоритма кластеризации данных Kernel-based Self-Organized Map (K-SOM), а также создании пересечений в кластерах, данный алгоритм классификации не уступает алгоритму one-against-one по точности. В тоже время для работы данного алгоритма требуется создание только $(N-1)$ SVM классификатора, а также для принятия решения требуется работа только $\log_2 N$ классификаторов.

## 1.4. Алгоритм кластеризации

SVM классификатор, основанный на бинарном дереве, на каждом шаге принимает решение, к какой из 2 групп, состоящих из множества классов, принадлежит входное значение. Существует много способов разделить множество классов на 2 группы, и по существу для каждой конкретной задачи нужно проводить отдельное исследование, какой алгоритм будет наиболее эффективным на используемых данных. В нашей разработке будем использовать самый популярный алгоритм кластеризации k-means [5]. В соответствии со статьей исследователей, разработавших подход к кодировке лица [19], лица похожих людей будут иметь близкие по значению вектора, что дает возможность хорошо кластеризировать полученные вектора.

Алгоритм кластеризации автоматически делит тренировочные данные на кластеры. Таким образом, необходимо будет делить каждое новое множество на два новых до тех пор, пока в одном кластере не останутся вектора, принадлежащие фотографии одного человека (в одном кластере останется один класс) с большой точностью. Также необходимо учитывать, что кластеры должны быть хорошо сбалансированы по количеству векторов, так как SVM классификаторы плохо работают на несбалансированных данных [10].

## Выводы по главе

В данной главе был проведен анализ существующих подходов к распознаванию лиц. Была представлена общая схема подхода к распознаванию лиц и основных шагов распознавания. Для каждого шага были приведены примеры алгоритмов, позволяющие реализовать этот шаг различными способами. Был проведен анализ различных алгоритмов для кодировки лица, классификации и кластеризации кодов лица. Для кодировки лица был выбран подход, описанный в статье исследователей из Google в 2015 году, позволяющий кодировать изображение любого лица в 128-размерный вектор. Евклидово расстояние между векторами двух изображений лица одного человека меньше, чем расстояние между векторами двух изображений разных людей. Это позволяет проводить сравнение, кластеризацию и классификацию изображений лиц. Кроме того, был выбран подход для классификации лиц и

поиска личности в базе данных. Для этого используется алгоритм, комбинирующий бинарное дерево и бинарный SVM классификатор. Такой подход позволяет снизить количество обучаемых классификаторов, а также уменьшить вычислительную сложность классификации. Все выбранные подходы и алгоритмы будут использованы при реализации сервиса распознавания лиц для идентификации личности.

# Глава 2. Проектирование сервиса

В данной главе подробно рассматриваются проблемы проектирования сервиса для распознавания лиц. Сначала проводится анализ клиент-серверной архитектуры сервиса и выбираются основные подходы в реализации такой архитектуры. После общей архитектуры рассказывается о архитектуре и используемых подходах в проектировании SDK сервиса на платформе iOS. Далее рассказывается о архитектуре серверной части сервиса, API сервера и архитектура модуля обучения классификатора.

## 2.1. Архитектура сервиса

В данной работе была поставлена цель разработать сервис распознавания лиц для идентификации личности, выполняющий вычисления по извлечению признаков лица на мобильном устройстве на платформе iOS и предоставляющий возможность идентифицировать личность в локальной или удаленной базе данных. Для достижения поставленной цели работы необходимо разработать клиент-серверный сервис.

Клиентская часть сервиса производит обнаружение лица на изображении или в видео потоке, нормализацию обнаруженного изображения лица и вычисления по кодированию нормализованного изображения при помощи глубокой сверточной нейронной сети. Далее, полученный код, представленный в виде 128 размерного вектора, отправляется на сервер при помощи API, для произведения поиска личности по базе данных. Клиентская часть сервиса реализована в виде SDK для устройства на платформе iOS. SDK предоставляет перечисленный выше функционал для сторонних приложения, а также предоставляет готовую реализацию для доступа к API сервиса.

Серверная часть выполняет задачи по обработке запросов со стороны клиентской части, произведение классификации полученного от клиента кода лица, поиск в базе данных распознанной личности и возвращение ответа клиенту информации о распознанной личности. Для выполнения этой задачи сервер предоставляет API, позволяющий получить доступ к функционалу серверной части. Помимо этого серверная часть предоставляет функционал web-приложения, позволяющее авторизоваться пользователю в системе, просмотреть профили личностей, занесенных в базу данных и фотографии личностей, на которых обучен классификатор. Web-приложение предоставляет возможность добавлять и удалять личности и фотографии личностей. Вся информация хранится в базе данных. Помимо этого, web-сервер предоставляет возможность переобучения классификатора, для добавления новых личностей и их фотографий для распознавания.

## 2.2. Архитектура SDK сервиса

При разработке архитектуры SDK сервиса необходимо учитывать возможности платформы iOS, а также предоставить максимальную универсальность использования возможностей сервиса. Основной функционал, предоставляемый SDK сервиса следующий:

1. Обнаружение лиц на изображении;
2. Обнаружение лиц в видеопотоке;
3. Нормализация изображения лица;
4. Распознавание лица на изображении с использованием сторонней базы данных;
5. Распознавание лица в видеопотоке с использованием сторонней базы данных;
6. Распознавание лица на изображении с использованием локального хранилища данных;
7. Распознавание лица в видеопотоке с использованием локального хранилища данных.

Далее рассматриваются подходы к построению архитектуры сервиса, подходы к использованию моделей машинного обучения на устройстве iOS, подходы к работе с API серверной части, а также подход к тестирования SDK сервиса.

### 2.2.1. Архитектура SDK

Для разработки iOS приложения Apple предоставляет официальную документацию [6], в которой описан общий подход к построению архитектуры iOS приложения. В общем виде она представляет из себя паттерн MVC, в котором различные объекты iOS приложения отнесены к модели, представлению или контроллеру (рис. 2.1).

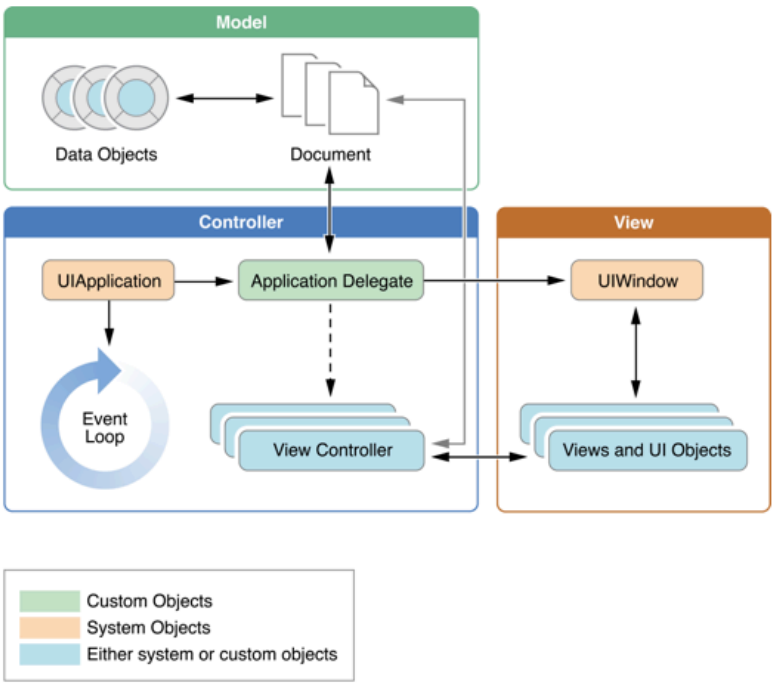


Рисунок 2.1. Архитектура iOS приложения

В случае разработки SDK отсутствует слой представления, SDK предоставляет программный интерфейс для доступа к функционалу сервиса. Тем не менее, необходимо использовать эту структуру, таким образом, чтобы приложения, использующие SDK не нарушали бы данного подхода.

Помимо этого, для тестирования SDK разработано приложение, демонстрирующее возможности сервиса (оно будет описано далее). Это приложение разработано с использованием данного паттерна и демонстрирует каким образом необходимо интегрировать SDK в мобильное приложение.

Таким образом, при разработке данного SDK нет необходимости отклоняться от стандартного подхода, предложенного официальной документацией Apple. Он предоставляет достаточную гибкость для поддержки и изменения отдельных модулей приложения, что позволяет снизить трудоемкость внесения изменений в приложение.

### 2.2.2. Модели машинного обучения

Ключевой частью SDK являются модели машинного обучения, предоставляющие возможность производить обнаружение лица на изображении, обнаружение характеристик лица (глаза, нос, рот), кодирование лица в вектор и классифицирование полученного кода, в случае использования локального хранилища для идентификации личности. Таким образом, необходим функционал внедрения моделей машинного обучения в iOS приложение.

В 2017 году Apple представила фреймворк Core ML [13], предоставляющий функционал для использования компьютерного зрения, обработки естественного языка, а также внедрения алгоритмов машинного обучения в игры (рис. 2.2).

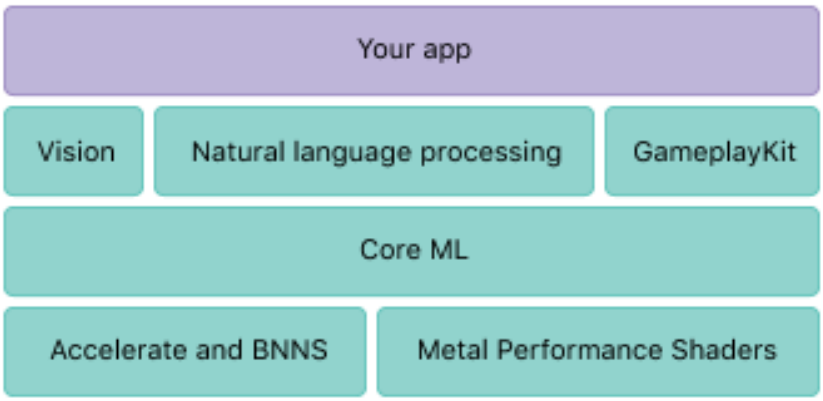


Рисунок 2.2. Функционал Core ML

Помимо этого, Core ML предоставляет возможность интегрировать в iOS приложение заранее обученную модель машинного обучения, которую можно конвертировать из стороннего фреймворка в формат Core ML при помощи  Core ML Tools [12]. Список

фрейворков, из которых можно конвертировать модели, и виды доступных для использования моделей приведены ниже (табл. 1).

Таблица 1

Модели и фреймворки поддерживаемые Core ML Tools

| Тип модели | Поддерживаемые модели | Фреймворки |
|---|---|---|
| Нейронные сети | Прямого распространения, сверточные, рекуррентные | Caffe v1<br>Keras 1.2.2+ |
| Древовидные ансамбли | Случайный лес, усиленные деревья, деревья решений | scikit-learn 0.18<br>XGBoost 0.6 |
| SVM | Скалярная регрессия, многоклассовая классификация | scikit-learn 0.18<br>LIBSVM 3.22 |
| Обобщенные линейные модели | Линейная регрессия, логистическая регрессия | scikit-learn 0.18 |
| Feature engineering | Разреженная векторизация, плотная векторизация, категориальная обработка | scikit-learn 0.18 |
| Pipeline models | Последовательные модели | scikit-learn 0.18 |

В первой главе были определены модели машинного обучения, которые будут использованы при разработке сервиса, рассмотрим подходы к их реализации подробнее.

Для обнаружения лица на изображении используется функционал фреймворка Vision [36], позволяющий обнаружить лицо на изображении, а также выделить характеристики лица. Для нормализации изображения лица также будут разработан алгоритм, в соответствии со статьей, в которой он описывался [29]. Стандартные библиотеки, реализующие этот алгоритм отсутствуют, а сторонние библиотеки слишком большие для решения только этой задачи и значительно увеличат размер SDK. Кроме этого они не используют функционал CoreML для оптимизации работы под iOS.

В качестве нейронной сети для кодирования нормализованного изображения в вектор используется глубокая сверточная нейронная сеть, обученная методом, описанным исследователями из Google в 2015 году [19]. Обучение нейронной сети таким методом требует больших вычислительных мощностей, а также базы данных промаркированных фотографий лиц людей, состоящих из более чем 200 миллионов записей. При разработке данного сервиса отсутствуют такие технические возможности, а также базы данных такого размера отсутствуют в публичном доступе. Поэтому при разработке данного сервиса будет использоваться

обученная нейронная сеть проекта Open Face [30]. Она обучена по той же методологии, что описана в статье исследователей из Google, однако при ее обучении использовалась база данных меньшего размера, что уменьшило точность нейронной сети, однако она все равно остается достаточно высокой. Однако, данная нейронная сеть написана с использованием фреймворка Torch, что не дает возможности использовать ее в iOS приложении. Для того чтобы интегрировать эту нейронную сеть в iOS приложение, необходимо сначала конвертировать ее в формат Caffe [9] или Keras [26], а затем конвертировать в формат CoreML. Не существует готового конвертера, позволяющего конвертировать модель нейронной сети формата Torch в один из этих двух форматов, поэтому специально для данного сервиса был разработан конвертер из формата Torch в формат Keras. Описание и принцип работы конвертера будет приведен в следующей главе. Далее нейронную сеть в формате Keras можно конвертировать в формат CoreML, однако некоторые слои данной нейронной сети не поддерживаются по умолчанию CoreML, поэтому разработана реализация этих слоев внутри SDK сервиса. Описание реализации слоев также приведено в третьей главе.

Также при помощи CoreML можно интегрировать заранее обученную модель SVM классификатора, что позволит хранить данные для идентификации личности локально. Однако, как видно из таблицы, при помощи CoreML нет возможности реализовать алгоритм SVM классификатора основанном на бинарном дереве, поэтому реализация такого алгоритма останется только на серверной части. Также, CoreML не дает возможности обучать классификатор на устройстве, поэтому обучение классификатора также останется только на серверной стороне, а SDK будет использовать только заранее обученные классификаторы.

CoreML доступна начиная с версии iOS 11. Однако, разработка дополнительных слоев, необходимых для работы выбранной нейронной сети, стала доступна только с версии iOS 11.2 [25]. Поэтому разработка будет вестись для версии iOS 11.2.

### 2.2.3. Работа с API

Для коммуникации с серверной частью сервиса, SDK использует API сервиса, доступ к которому можно получить по адресу http://<*адрес_сервера*>/api. Подробное описание используемых методов API в SDK приведено в третьей главе.

Работа с сетью по отправке запросов и получения ответов производится стандартными методами URLSession [35]. Для передачи данных между клиентом и сервером используется формат JSON. Для сериализации и десериализации из формата JSON используются стандартные методы NSJSONSerialisation [28].

### 2.2.4. Входные данные

В качестве входных данных для распознавания лиц SDK предоставляет возможность использовать фотографию или видео поток с главной камеры iOS устройства в реальном времени. Для получения изображения с камеры используется функционал фреймворка AVFoundation [8]. Поток с камеры обрабатывается асинхронно, это позволяет получать плавную картинку в реальном времени с камеры, а результат распознавания лиц получать и отображать с задержкой. Время задержки зависит от используемого метода распознавания (локальное хранилище или удаленное хранилище данных).

### 2.2.5. Демонстрация SDK сервиса

Специально для демонстрации возможностей SDK сервиса разработано демонстрационное приложение, позволяющее ознакомиться с основным функционалом SDK. Данное приложение использует основные функции по распознаванию лиц SDK, демонстрируя работу по распознаванию лица на фотографии, а также распознавание лиц в реальном времени с главной камеры мобильного устройства. Это приложение создано для демонстрации способа интеграции SDK в iOS приложения, чтобы в последствии разработчики могли использовать SDK в своих приложениях корректным способом. Кроме того при помощи данного приложения проводилось тестирование работы SDK.

## 2.3. Архитектура серверной части сервиса

При разработке архитектуры серверной части необходимо учитывать все задачи, которые должна решать серверная часть. Серверная часть сервиса решает следующие задачи:

1. Отображение и редактирование данных по распознаваемым личностям и их базы фотографий посредством веб интерфейса;
2. API для клиентской части сервиса, предоставляющий функционал для распознавания личности;
3. Обучение классификатора, позволяющее производить поиск личности по коду, полученному из фотографии его лица;
4. Нормализация фотографий для обучения классификатора;
5. Нейронная сеть, используемая при обработке и кодировании фотографий на сервере;
6. Хранение данных.

Разработанная архитектура и выбранные подходы должны покрывать все эти задачи. На основании этих задач выбирается методы и технологии, используемые в разработке сервера.

### 2.3.1. Архитектура сервера

Выше перечислены все задачи сервера, которые он должен выполнять. Помимо стандартных задач, типичных для любого сервера (web приложение, API, хранение данных), здесь добавляется задача по обучению классификатора, а также по нормализации фотографии и кодированию ее в вектор при помощи нейронной сети. Для того чтобы выполнить все эти задачи наиболее эффективно был выбран язык Python 3.6. На данный момент Python является одним из наиболее популярных языков программирования для машинного обучения, а также для веб программирования [37]. Для этого языка есть большое количество различных библиотек, которые можно использовать при разработке такого сервиса.

Ниже (рис. 2.3) представлена схема, на которой отображены основные модули сервера, а также то, как они взаимодействуют между собой.

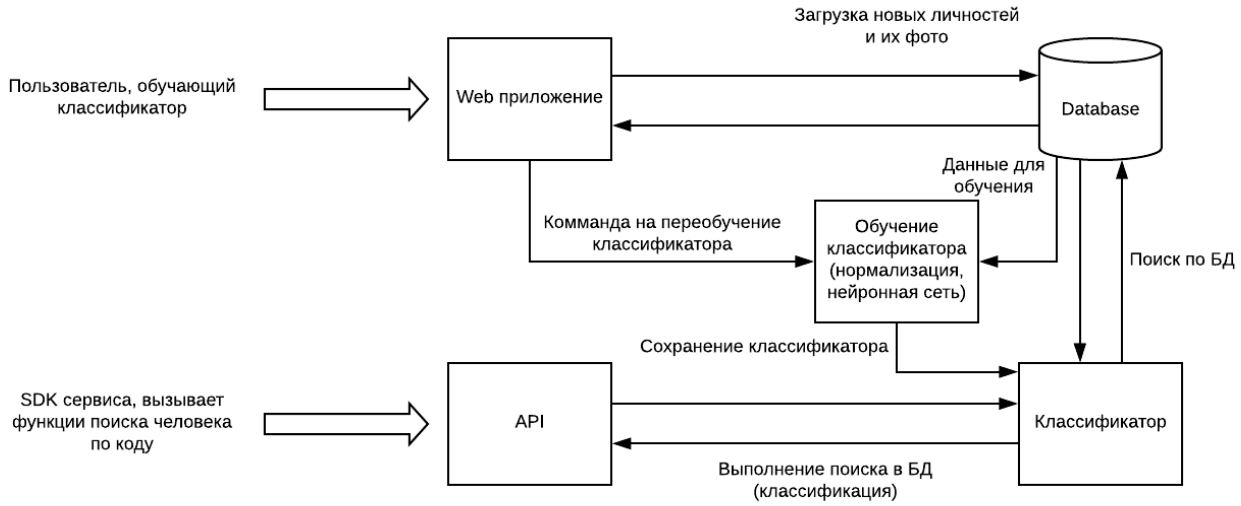


Рисунок 2.3. Архитектура сервера

Далее каждый модель этой архитектуры будет разобран подробнее.

### 2.3.2. Архитектура Web приложения

Web приложение представляется из себя основной интерфейс, для управления данными о личностях, которые хранятся в базе данных. Кроме этого, через веб интерфейс, пользователь может добавлять и удалять фотографии личности на сервер для обучения классификатора. После того как пользователь заканчивает редактирование, он может нажать кнопку переобучения классификатора, которая запустить переобучение классификатора в фоновом режиме. Когда обучение закончится, для клиентской части сервиса станет доступен поиск по новой базе данных.

В качестве web сервера был выбран web сервер Flask. Это свободно распространяемый веб-фреймворк, позволяющий строить легко адаптируемые веб приложения под различные нужды [34]. В состав фреймворка входит поддержка секретных cookie, обработка http запросов, встроенная работа с шаблонами Jinja2 для генерации пользовательского интерфейса в html формате, обработка REST запросов. Помимо этого для фреймворка Flask существует множество расширений, по работе с базами данных, кешированию, выполнению задач в фоне и многие другие расширения, позволяющие быстро создать веб приложение и адаптировать его под необходимые требования. Таким образом при Flask позволяет выполнить все поставленные задачи.

### 2.3.3. Архитектура API

Для взаимодействия серверной и клиентской части необходимо разработать программный интерфейс (API), позволяющий получать необходимые данные с сервера. Для этого необходимо разработать REST API, которое позволит взаимодействовать с сервером по определенному списку функций, вне зависимости от того, с какой платформы обращается к нему клиент. Это позволяет использовать API на любой платформе и при этом дает возможность менять клиент, не завися от сервера, в то время как разработка и сложность архитектуры серверной части сильно упрощается, что дает возможность быстро и эффективно масштабировать серверную часть [7]. Доступ к API будет производиться по ссылке http://<*адрес_сервера*>/api. Подробный список методов API будет описан в третьей главе.

### 2.3.4. Классификатор

В первой главе были описаны классификаторы SVM различных типов. При разработке данного сервиса также будут использоваться различные типы классификаторов. Это связано с тем, что на платформе iOS можно запускать только SVM классификаторы, обученные при помощи библиотеки scikit-learn, что не дает возможности использовать классификатор использующий в своей основе бинарное дерево. Однако, данный классификатор можно разработать на языке Python и использовать его только на серверной часть. Далее, в зависимости от настроек, можно выбрать, какой классификатор будет приоритетным и какой будет заниматься распознаванием личности. Описание реализации классификатора в виде бинарного дерева будет приведено в третьей главе.

Помимо этого, классификатор после обучения записывается в файл, для того чтобы в случае перезапуска сервера, его можно было бы быстро восстановить не переобучая с нуля. Также, этот файл используется для выгрузки на клиент, в случае если используется локальный классификатор для распознавания личности.

### 2.3.5. Архитектура обучения классификатора

Обучение классификатора на большом объеме данных трудоемкая задача и может занять продолжительное время. В это время пользователь должен иметь возможность продолжать взаимодействовать с системой, а старый классификатор все еще должен работать на старых данных. По окончании обучения классификатора, пользователь видит изменение в интерфейсе, индикатор рядом с новыми личностями и фотографиями меняется с красного на зеленый.

Выполнение обучения в фоне выполняется за счёт библиотеки Python multiprocessing. Она позволяет выполнять задачи асинхронно, что дает возможность продолжать работу на сервере, пока идет обучение классификатора.

### 2.3.6. Архитектура базы данных

В данном сервисе используется база данных PostgeSQL. Это популярная, свободно распространяемая база данных. В базе данных на сервере хранится 4 сущности:

- users – таблица пользователей. На данный момент регистрация в сервисе не доступна, поэтому выполняет роль хранения данных для авторизации в сервисе для одного пользователя.
- persons – таблица распознаваемых личностей. Хранит персональную информацию о личности.
- photos – таблица фотографий. Хранит имя фотографии на сервере, а также id личности, которой принадлежит. После обучения классификатора хранит вектор, кодирующий фотографию.
- classifiers – таблица классификаторов. Используется для хранения версии классификатора, а также состояния на текущий момент (обучается или готов к использованию).

Ниже приведена схема базы данных (рис. 2.4)

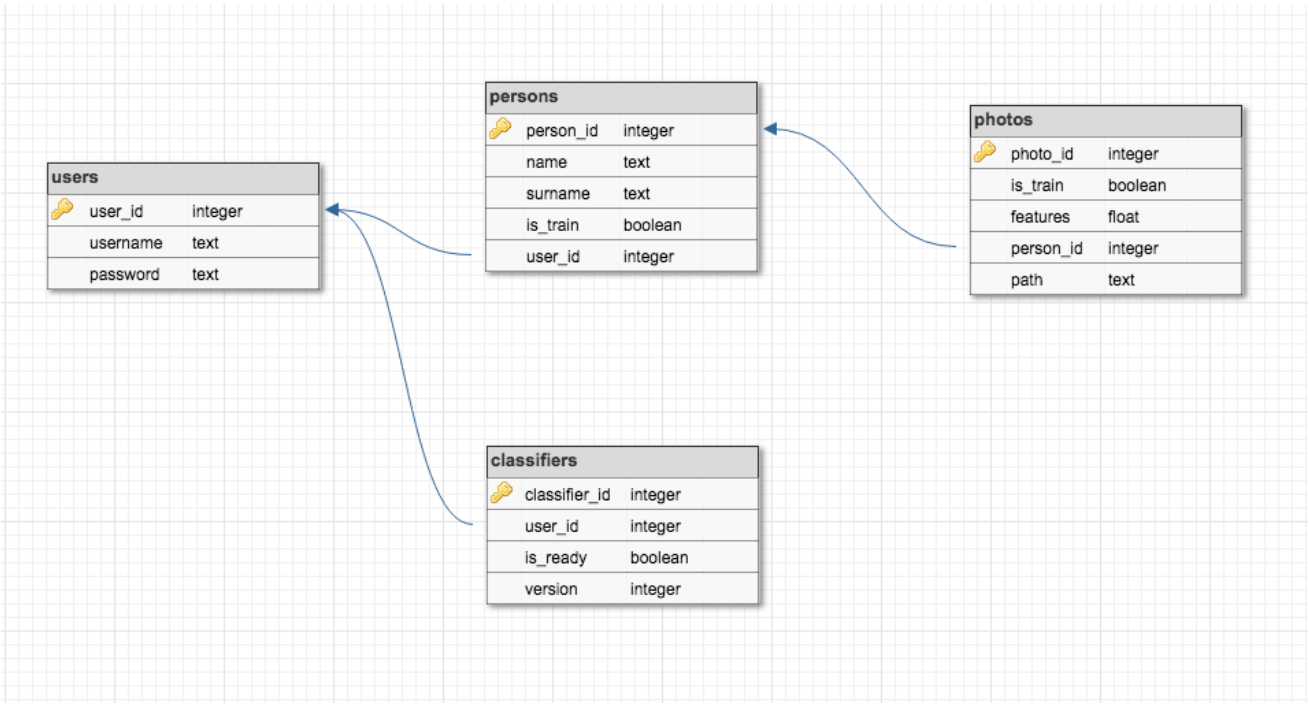


Рисунок 2.4. Схема базы данных

## Выводы по главе

В данной главе был проведен анализ архитектуры приложения. Были рассмотрены различные подходы, технологии и возможности реализации сервиса распознавания лиц на практике.

Для SDK сервиса было выбрано использовать готовую нейронную сеть, предварительно разработав конвертер, для конвертации модели нейронной сети из формата Torch в Keras. Затем, при помощи Core ML Tool модель конвертируется в формат CoreML. Слои нейронной сети, которые не поддерживаются фреймворком Keras по умолчанию и, как следствие, не будут конвертированы в формат CoreML, будут реализованы на языке Swift внутри SDK.

Помимо этого, в SDK будет реализован алгоритм нормализации изображения лица. Для этого не будут использоваться сторонние библиотеки, поскольку они не используют средств iOS, а также чрезмерно большие по размеру для внедрения их в SDK для реализации одной функции.

Серверная часть будет реализована на языке Python 3.6. В качестве веб-сервера был выбран вер-сервер Flask. В качестве базы данных была выбрана PostgreSQL.

На сервере разработано web приложение, позволяющее управлять данными, хранящимися на сервере. При помощи него можно добавлять и удалять личности, а также их фотографии. Кроме этого, при помощи web-интерфейса запускается процесс переобучения классификатора.

Помимо этого, на сервера разработано два вида классификаторов. Обычный многоклассовый SVM классификатор использует готовую реализацию из библиотеки scikit-learn. Классификатор, базирующийся на бинарном дереве также использует функционал scikit-learn, однако дополняет его реализацией SMV классификации основанном на бинарном дереве.

Помимо этого, на сервере разработано REST API, позволяющее клиенту получать данные с сервера.

# Глава 3. Техническая реализация и результаты

В данной главе подробно описаны ключевые моменты реализации сервиса с технической точки зрения. Здесь будут описаны реализации конвертации модели нейронной сети в необходимы формат, описан способ разработки дополнительных слоев нейронной сети на платформе iOS, реализация нормализации изображения лица средствами iOS, реализация классификатора основанного на бинарном дереве, разработанные методы API и подробности реализации Web-приложения.

## 3.1. Конвертация моделей машинного обучения

Основой всего сервиса является глубокая сверточная нейронная сеть [19]. Обучение такой нейронной сети требует много времени и ресурсов, которые отсутствуют при разработке данного сервиса. Поэтому было принято решение использовать готовую нейронную сеть OpenFace. Для использования ее на iOS устройстве, в первую очередб ее необходимо перевести в формат Keras. За это отвечает класс TotchToKeras. Для конвертации необходимо ввести

```
from Torch2KerasConverter.torch_to_keras import TorchToKeras

converter = TorchToKeras((96, 96, 3))

converter.torch_to_keras('nn4.small2.v1.new.t7', 'KerasSmall2V1')
...
```

Где “nn4.small2.v1.new.t7” имя файла с моделью Torch. После выполнения работы скрипта будет создано 2 файла KerasSmall2V1.py с кодом генерации модели и KerasSmall2V1.h5 с параметрами (весами) нейронной сети. Рассмотрим класс TorchToKeras поподробнее. В нем есть две основные функции. Первая lua_recursive_model, производит рекурсивный проход по вложенным модулям (Sequential и nn.Incenption слои) модели и повторяет все слои в формате Keras. В последовательных модулях алгоритм проходит циклом, пока не дойдет до нового вложенного модуля или до конца. Таким образом, в результате рекурсивного прохода мы получаем абсолютно аналогичную модель.

```
def lua_recursive_model(self, module, prev=None, isFirst=False, inp=None):
    layers = None

    if prev is None:
        prev = inp
```

```
    for m in module.modules:
        name = type(m).__name__

        if name == 'TorchObject':
            name = m._typename.replace('cudnn.', '')
            m = m._obj

        if name == 'SpatialConvolution' or name == 'nn.SpatialConvolutionMM':
            if not hasattr(m, 'groups') or m.groups is None:
                m.groups = 1

            if m.padH and m.padW is not None or 0:
                layers = ZeroPadding2D(padding=(m.padW, m.padH))(prev)
                prev = layers

            weights = np.transpose(m.weight.numpy()
                                   .reshape((m.nOutputPlane, m.nInputPlane, m.kW,
m.kH)), (2, 3, 1, 0))
            bias = m.bias.numpy()
            layers = Convolution2D(m.nOutputPlane, (m.kW, m.kH), strides=(m.dW,
m.dH),
                                   weights=[weights, bias])(prev)
        elif name == 'SpatialBatchNormalization':
            weights = m.weight.numpy()
            bias = m.bias.numpy()
            running_mean = m.running_mean.numpy()
            running_var = m.running_var.numpy()
            layers = BatchNormalization(axis=len(self.inputShape),
momentum=m.momentum, epsilon=m.eps,
                                        weights=[weights, bias, running_mean,
running_var])(prev)
        elif name == 'ReLU':
            layers = Activation('relu')(prev)
...
```

Выше приведена часть кода функции. Как видно, для того, чтобы была возможность конвертировать слои, нужно на каждый слой формата Torch прописать реализацию в формате Keras.

Теперь посмотрим на функцию lua_recursive_source:

```
def lua_recursive_source(self, module, xv='x', prev='x', isFirst=False):
    s = []

    if isFirst:
        prev = 'inp'

    for m in module.modules:
        name = type(m).__name__

        if name == 'TorchObject':
            name = m._typename.replace('cudnn.', '')
            m = m._obj

        if name == 'SpatialConvolution' or name == 'nn.SpatialConvolutionMM':
            if not hasattr(m, 'groups') or m.groups is None:
                m.groups = 1

            if m.padH and m.padW is not None or 0:
                s += ['{} = ZeroPadding2D(padding=({}, {}))({})'.format(xv,
m.padW, m.padH, prev)]
                prev = xv

            s += ['{} = Convolution2D({}, ({}, {}), strides=({},
{}))({})'.format(xv, m.nOutputPlane, m.kW, m.kH,

m.dW, m.dH, prev)]
        elif name == 'SpatialBatchNormalization':
            s += ['{} = BatchNormalization(axis={}, momentum={},
epsilon={})({})'.format(xv, len(self.inputShape),

m.momentum, m.eps, prev)]
        elif name == 'ReLU':
            s += ["{} = Activation('relu')({})".format(xv, prev)]
...
```

Эта функция работает по аналогии с предыдущей, но создает не модель в формате Keras, а исходный код, который при запуске воспроизведет модель.

Кроме этого, для конвертации созданы дополнительные функции, реализующие слои, которых нет в Keras. Они используют фреймворк tensorflow и backend фреймворка Keras. В последствии их же нужно будет реализовать на платформе iOS. Ниже код слоев, которые отсутствуют в Keras.

```
import tensorflow as tf
from keras import backend as K


def lrn(a, size, alpha, beta):
    return tf.nn.lrn(a, depth_radius=size, alpha=alpha, beta=beta)


def square(a):
    return a**2


def mulConstant(a, const):
    return a * const


def sqrt(a):
    return K.sqrt(a)


def l2Normalize(a, axis):
    return K.l2_normalize(a, axis=axis)
```

В результате работы этого скрипта появляются файлы, содержащие нейронную сеть. Данный скрипт может конвертировать следующие слои нейронной сети из формата Torch в формат Keras:

- SpatialConvolution and nn.SpatialConvolutionMM
- SpatialBatchNormalization
- ReLU
- Sequential
- SpatialMaxPooling
- SpatialCrossMapLRN
- SpatialLPPooling
- Square
- SpatialAveragePooling
- MulConstant
- Sqrt

- Reshape and View
- Linear
- Normalize
- DepthConcat
- nn.Inception

Других конвертеров, которые могли бы конвертировать все перечисленные слои из формата Torch в формат Keras не существует на данный момент. Скрипт выложен в свободный доступ на сайте GitHub: https://github.com/NikKonst/Torch2KerasConverter.

## 3.2. Дополнительные слои нейронной сети для платформы iOS

Здесь будут перечислены реализации дополнительных слоев нейронной сети на платформе iOS, которые не поддерживаются по умолчанию CoreML. Во всех реализациях используется стандартная iOS библиотека Accelerate, позволяющая производить быстрые математические операции над векторами и матрицами.

### 3.2.1. Square

Здесь описывается реализация слоя, возводящего в квадрат значение, поданное на вход.

```
let input = inputs[i]
let output = outputs[i]

let count = input.count
let iptr = UnsafeMutablePointer<Float>(OpaquePointer(input.dataPointer))
let optr = UnsafeMutablePointer<Float>(OpaquePointer(output.dataPointer))

// output = input^2
var countAsInt32 = Int32(count)
var pow2: [Float] = Array(repeating: 2, count: count)
vvpowf(optr, &pow2, iptr, &countAsInt32)
```

### 3.2.2. Sqrt

Слой, извлекающий квадратный корень из своего входящего значения.

```
let input = inputs[i]
let output = outputs[i]

let count = input.count
let iptr = UnsafeMutablePointer<Float>(OpaquePointer(input.dataPointer))
```

```
            let optr =
UnsafeMutablePointer<Float>(OpaquePointer(output.dataPointer))
            print(count)

            // output = sqrt(input)
var countAsInt32 = Int32(count)
vvsqrtf(optr, iptr, &countAsInt32)
```

### 3.2.3. MulConstant

Умножение предыдущего слоя на константу.

```
let input = inputs[i]
let output = outputs[i]

let count = input.count
let iptr = UnsafeMutablePointer<Float>(OpaquePointer(input.dataPointer))
let optr = UnsafeMutablePointer<Float>(OpaquePointer(output.dataPointer))

// output = input * const
vDSP_vsmul(iptr, 1, &const, optr, 1, vDSP_Length(count))
```

### 3.2.4. LRN

LRN – Locale response normalization, способ нормализации в нейронных сетях (Krizhevsky, и др.). Представляется следующей формулой (1).

(1)

$$b_{x,y}^{i} = \frac{a_{x,y}^{i}}{\left(k + \alpha \sum_{j=\max\left(0,i-\frac{n}{2}\right)}^{\min\left(N-1,i+\frac{n}{2}\right)} \left(a_{x,y}^{i}\right)^{2}\right)^{\beta}}$$

$a_{x,y}^{i}$ – i-й вход на позиции x, y

$b_{x,y}^{i}$ – i-й выход на позиции x, y

N – количество сверточных слоев, предшествующих нормализации.

n – окно нормализации (в нашем случае 5)

$k, \alpha, \beta$ – прочие параметры, в нашем случае 1; 0.0001; 0.75 соответственно

```
let input = inputs[i]
let output = outputs[i]

let count = input.count
```

```
let iptr = UnsafeMutablePointer<Float>(OpaquePointer(input.dataPointer))
let optr = UnsafeMutablePointer<Float>(OpaquePointer(output.dataPointer))

// output = input^2
var countAsInt32 = Int32(count)
var pow2: [Float] = Array(repeating: 2, count: count)
vvpowf(optr, &pow2, iptr, &countAsInt32)

// output = sum window 5(input^2)
vDSP_vswsum(optr, 1, optr, 1, vDSP_Length(count-4), vDSP_Length(depth_radius))

// output = alpha * (sum window 5(input^2))
vDSP_vsmul(optr, 1, &alpha, optr, 1, vDSP_Length(count))

// output = k + alpha * (sum window 5(input^2))
vDSP_vsadd(optr, 1, &k, optr, 1, vDSP_Length(count))

// output = (k + alpha * (sum window 5(input^2))) ^ b
var powBeta: [Float] = Array(repeating: beta, count: count)
vvpowf(optr, &powBeta, optr, &countAsInt32)

// output = input / ((k + alpha * (sum window 5(input^2))) ^ b)
vvdivf(optr, iptr, optr, &countAsInt32)
```

### 3.2.5. L2Normalize

L2Normalize – L2 нормализация (Krogh, et al.). Представляется в следующем виде (2).

(2)

$$b_i = a_i / \sqrt{\sum_{j=1}^{N} (a_j)^2}$$

$a_i$ – i-й вход

$b_i$ – i-й выход

N – длина вектора

```
let input = inputs[i]
let output = outputs[i]

let count = input.count
```

```
let iptr = UnsafeMutablePointer<Float>(OpaquePointer(input.dataPointer))
let optr = UnsafeMutablePointer<Float>(OpaquePointer(output.dataPointer))

// output = input^2
var countAsInt32 = Int32(count)
var pow2: [Float] = Array(repeating: 2, count: count)
vvpowf(optr, &pow2, iptr, &countAsInt32)

// output = sum(input^2)
vDSP_sve(optr, 1, optr, vDSP_Length(count))

// output = sqrt(sum(input^2))
vvsqrtf(optr, optr, &countAsInt32)

// output = input / sqrt(sum(input^2))
vDSP_vsdiv(iptr, 1, optr, optr, 1, vDSP_Length(count))
```

## 3.3. Нормализация изображения лица для платформы iOS

За нормализацию изображения лица отвечает класс MFFaceNormalization. Он по переданным ему координатам характеристик лица, выделяет 3 необходимые характеристики (уголки глаз и кончик носа) и приводит путем аффинного преобразования эти точки к заранее заданным нормальным координатам.

```
func normalizeAndScaleTo96(face: MFFaceRect, image: UIImage) -> UIImage {

var A: [Double] = [Double(face.mainPoints[0].x), Double(face.mainPoints[1].x),
Double(face.mainPoints[2].x),
                           Double(face.mainPoints[0].y),
Double(face.mainPoints[1].y), Double(face.mainPoints[2].y),
                           1.0, 1.0, 1.0]

var const = Double(image.size.height)
var X: [Double] = [0.194157, 0.7888591, 0.4949509,
                           0.16926692, 0.15817115, 0.5144414,
                           1.0, 1.0, 1.0]

// X = X * image.size.width
vDSP_vsmulD(X, 1, &const, &X, 1, vDSP_Length(X.count))
```

```
vDSP_vsmulD(A, 1, &const, &A, 1, vDSP_Length(A.count))

A = self.invert(matrix: A)

var Result: [Double] = [0, 0, 0,
                        0, 0, 0,
                        0, 0, 0]

vDSP_mmulD(&X, 1, &A, 1, &Result, 1, 3, 3, 3)

let transform = CGAffineTransform(a: CGFloat(Result[0]), b: CGFloat(Result[3]), c:
CGFloat(Result[1]), d: CGFloat(Result[4]), tx: CGFloat(Result[2]), ty:
CGFloat(Result[5])).scaledBy(x: 1, y: -1)
var coreImage = image.ciImage

if (!(coreImage != nil)) {
      coreImage = CIImage.init(cgImage: image.cgImage!)
}


coreImage = coreImage?.transformed(by: transform)

return self.resizeImageToSquare(image: self.convert(cmage: coreImage!), newWidth:
96)
}
```

## 3.4. Классификатор SVM основанный на бинарном дереве

SVM классификатор, основанный на бинарном дереве представлен классом *SVMBinaryClassificator*, в котором есть два основных метода. Метод инициализации принимает на вход массив векторов кодов лиц и соответствующий массив, маркирующих их. Далее при помощи метода кластеризации k means он делит массив векторов на два кластера. Если хоть в одном из 2-х полученных кластеров осталось более одного класса, на этот кластер рекурсивно вызывается метод инициализации следующего бинарного классификатора. Следующие классификаторы присваиваются переменным *next0* и *next1* соответственно. В случае, если в кластере остались вектора только одного класса, рекурсивные вызовы прекращаются, переменной *cls0* (или *cls1*, в зависимости от номера кластера), присваивается номер класса,

которому принадлежит кластер. И так до тех пор, пока все кластеры не будут содержать ровно один класс.

Метод *predict* производит последовательную рекурсивную классификацию, до тех пор пока не дойдет до листа бинарного дерева.

```
class SVMBinaryClassificator:
    bin_clf = None
    next0 = None
    next1 = None
    cls0 = None
    cls1 = None

    def __init__(self, vecs, labels):
        vecs = np.array(vecs)
        labels = np.array(labels)
        kmeans = KMeans(n_clusters=2, random_state=0).fit(vecs)
        print(kmeans.labels_)
        print(labels)

        self.bin_clf = create_classifier(vec=vecs, labels=kmeans.labels_)

        mask0 = kmeans.labels_ == 0
        mask1 = kmeans.labels_ == 1

        uniq0 = np.unique(labels[mask0])
        if len(uniq0) > 1:
            self.next0 = SVMBinaryClassificator(vecs[mask0], labels[mask0])
        else:
            self.cls0 = uniq0[0]

        uniq1 = np.unique(labels[mask1])
        if len(uniq1) > 1:
            self.next1 = SVMBinaryClassificator(vecs[mask1], labels[mask1])
        else:
            self.cls1 = uniq1[0]

    def predict(self, vec):
        if self.bin_clf is not None:
            res = int(self.bin_clf.predict([vec]))
```

```
            if res == 0:
                if self.next0 is None:
                    return self.cls0
                else:
                    return self.next0.predict(vec)
            elif res == 1:
                if self.next1 is None:
                    return self.cls1
                else:
                    return self.next1.predict(vec)
        else:
            return None
```

## 3.5. API

Было разработано REST API сервиса для передачи данных на клиент. Взаимодействие идет в формате JSON. Доступ к API идет по ссылке http://<*адрес_сервера*>/api. Доступ ко всем функциям API по http методу POST.

Таблица 2

Методы API сервиса

| Функция | Входные параметры | Выходные параметры | Описание |
|---|---|---|---|
| vector | vector – массив из 128 значений в формате float | person – строка с именем и фамилией искомого человека | По коду лица возвращает имя и фамилию искомого человека |
| load | load – числовой параметр, указывает что необходимо загрузить классификатор | Ссылка на скачиваемый классификатор | Скачать актуальный классификатор |
| image | image – файл-картинка, на котором распознается лицо | person – строка с именем и фамилией искомого человека | По фото возвращает имя и фамилию искомого человека |

Все эти методы реализованы в функционале SDK и готовы к использованию.

## 3.6. Web-приложение

Основой веб-приложения являются 3 файла, в которых находятся методы, принимающие и обрабатывающие запросы от пользователя.

### 3.6.1. Login

В этом файле находятся методы:

- login – отображает окно логина;
- do_login – совершает логин (устанавливает сессию пользователя);

```
session['logged_in'] = True
session['user_id'] = int(result.user_id)
session['username'] = result.username
```

- logout – выход пользователя из системы.

### 3.6.2. Persons

В этом файле находятся методы, обрабатывающие запросы связанные с просмотром, добавлением и удалением профилей личностей. Также тут находится обработка переобучения классификатора:

- persons – метод отображает таблицу с личностями. При нажатии на кнопку переобучения, вызывается этот же метод с параметром «retrain». Переобучение вызывается следующим образом;

```
pool = Pool(processes=1)  # Start a worker processes.
pool.apply_async(retrain_classifier, [session.get('user_id')])
```

- add_person – открывает окно добавления личности;
- add_person_do – обрабатывает нажатие на кнопку «Add» в окне добавления личности.

### 3.6.3. Photos

В этом файле находятся методы, обрабатывающие запросы связанные с просмотром, добавлением и удалением фотографий:

- photos – отображение таблицы с фотографиями личности;
- add_ photo – отображение формы добавления фото;
- add_ photo _do – обработка загрузки фото, проверка корректности входного типа данных и количества лиц на фото.

## Выводы по главе

В этой главе была описана техническая реализация сервиса и полученные результаты. В результате был разработан сервис, производящий основные вычисления по распознаванию лиц на мобильном устройстве. Это позволяет значительно снизить количество передаваемой информации по сети интернет, что снижает время задержки ответа от сервера. В случае же использования локального классификатора, можно вовсе избежать использование интернета. Все это демонстрирует возможность оптимизации существующих сервисов по распознаванию личности.

Такой сервис можно легко внедрить в любое iOS приложение и он не будет требовать долгого процесса внедрения сервиса. Все это может снизить цену подобных сервисов, а также уменьшить использование интернет трафика сервисов для распознавания лиц.

# Заключение

В рамках данной ВКР был разработан сервис по распознаванию лиц для идентификации личности, который производит вычисления по поиску лица на изображении, нормализации лица, а также кодированию лица на мобильно устройстве. Были рассмотрены и выбраны приоритетные подходы к распознаванию лиц. Для кодирования лица был выбран подход, представленный исследователями из Google, позволяющем кодировать лицо в вектор размерности 128. Далее на таких векторах обучается SVM классификатор, позволяющий производить классификацию векторов новых фотографий личностей.

Кроме этого была разработана архитектура сервиса. Было решено использовать фреймворк CoreML для интеграции моделей машинного обучения в сервис. Также была разработана архитектура серверной части сервиса, предоставляющей функционал по хранению данных, обучению классификатора распознаваемых лиц, а также предоставляющий web интерфейс и API для клиентской части.

В результате был разработан сервис по распознаванию лиц, выполняющий большую часть вычислений по распознаванию на мобильном устройстве. Такой сервис значительно снижает использование интернет трафика, что заметно уменьшает время распознавания из-за снижения времени передачи информации по сети интернет. Также это в целом уменьшает цену и сложность интеграции функционала распознавания лиц в мобильные приложения, так как было разработано специальное SDK для интеграции в приложение.

Дальнейшее развитие приложения может заключаться в увеличении точности распознавания лиц. Здесь можно использовать множество подходов, например использование рекуррентных сетей для увеличения точности в распознавании лиц в видео.

# Список литературы


1. **A Comparison of Different Face Recognition Algorithms** [В Интернете] / авт. Yi-Shin L., Wai-Seng N. и Chun-Wei L.. - 2009 г.. - 25 Апреля 2018 г.. - http://cs.unc.edu/~chunwei/papers/2009pr_face_recog.pdf.
2. **A comparison of facial recognition's algorithms** [В Интернете] / авт. Delbiaggio N. // Theseus. - 2017 г.. - 25 Апрель 2018 г.. - https://www.theseus.fi/bitstream/handle/10024/132808/Delbiaggio_Nicolas.pdf?sequence=1&isAllowed=y.
3. **A Comparison of Methods for Multi-class Support Vector Machines** / авт. Chih-Wei и Chih-Jen.
4. **A Simple Weight Decay Can Improve Generalization** [В Интернете] / авт. Krogh Anders и Hertz John // papers nips. - 20 Май 2018 г.. - https://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.pdf.
5. **An Efficient k-Means Clustering Algorithm: Analysis and Implementation** [В Интернете] / авт. Kanungo Tapas, Mount David и Netanyahu Nathan // cs umd. - 20 Май 2018 г.. - https://www.cs.umd.edu/~mount/Projects/KMeans/pami02.pdf.
6. **App Programming Guide for iOS** [В Интернете] / авт. Apple // Apple developer. - https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html.
7. **Architectural Styles and the Design of Network-based Software Architectures** / авт. Roy Thomas. - IRVINE : [б.н.], 2000 г..
8. **AVFoundation** [В Интернете] // Apple developer. - 20 Май 2018 г.. - https://developer.apple.com/av-foundation/.
9. **Caffe** [В Интернете] // BerkeleyVision. - 20 Май 2018 г.. - http://caffe.berkeleyvision.org/installation.html.
10. **Class-Boundary Alignment for Imbalanced Dataset Learning** [Журнал] / авт. Wu G. и Chang E.. - Washington, DC : In ICML, 2003 г..
11. **Comparison of face Recognition Algorithms on Dummy Faces** [Журнал] / авт. Singh A.. - [б.м.] : The International Journal of Multimedia & Its Applications, 2012 г..
12. **Converting Trained Models to Core ML** [В Интернете] / авт. Apple // Apple developer. - 20 Май 2018 г.. - https://developer.apple.com/documentation/coreml/converting_trained_models_to_core_ml.
13. **Core ML** [В Интернете] / авт. Apple // Apple developer. - 20 Май 2018 г.. - https://developer.apple.com/documentation/coreml.

14. **Emerging Trends in Engineering and Technology** [Конференция] / авт. Chaudhari S. и Kale A. // Face Normalization: Enhancing Face Recognition. - 2010.
15. **Face Detection Algorithms & Techniques** [В Интернете] // Face Detection. - 2017 г.. - 25 Апрель 2018 г.. - https://facedetection.com/algorithms/.
16. **Face Normalization** [В Интернете] / авт. Kinage K.. - 2011 г.. - 25 Апреля 2018 г.. - http://shodhganga.inflibnet.ac.in/bitstream/10603/9106/6/06_chapter%204.pdf.
17. **Face Recognition: Issues, Methods and Alternative Applications** [Раздел книги] / авт. Wójcik W., Gromaszek K. и Junisbekov M. // Face Recognition / авт. книги Ramakrishnan S.. - [б.м.] : IntechOpen, 2016.
18. **Face++** [В Интернете] // Face++. - 2017 г.. - 25 Апрель 2018 г.. - https://www.faceplusplus.com/.
19. **FaceNet: A Unified Embedding for Face Recognition and Clustering** [Журнал] / авт. Schroff F., Kalenichenko D. и Philbin J.. - [б.м.] : IEEE Xplore, 2015 г..
20. **Facial Recognition Could Drive The Next Online Marketing Revolution: Here's How** [В Интернете] / авт. DeMers J. // Forbes.com. - 2017 г.. - 13 Февраля 2018 г.. - https://www.forbes.com/sites/jaysondemers/2017/11/27/facial -recognition-could-drive-the-next-online-marketing-revolution-heres- how/.
21. **Histograms of Oriented Gradients for Human Detection** [В Интернете] / авт. Dala N. и Triggs B. // Inrialpes. - 25 Апрель 2018 г.. - http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf.
22. **How Facial Recognition is Shaping the Future of Marketing Innovation** [В Интернете] / авт. Louis M. // Inc.com. - 16 Февраля 2017 г.. - 13 Февраль 2018 г.. - http://inc.com/molly-reynolds/how-facial-recognition-is-shaping-the-future-of-marketing-innovation.html.
23. **Image Classification Using SVMs: One-against-One Vs One-against-All** [В Интернете] / авт. Gidudu A., Hulley G. и Marwala T. // Arxiv. - 25 Апреля 2018 г.. - https://arxiv.org/pdf/0711.2914.pdf.
24. **ImageNet Classification with Deep Convolutional Neural Networks** [В Интернете] / авт. Krizhevsky Alex, Sutskever Ilya и Hinton Geoffrey // cs toronto edu. - 20 Май 2018 г.. - http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf.
25. **Integrating Custom Layers** [В Интернете] // Apple developer. - 20 Май 2018 г.. - https://developer.apple.com/documentation/coreml/core_ml_api/integrating_custom_layers?language=objc.
26. **Keras** [В Интернете] // Keras. - 20 Май 2018 г.. - https://keras.io/.
27. **Multi-Class Support Vector Machine** [В Интернете] / авт. Wang Z. и Xue X. // Springer. - 2014 г.. - 25 Апреля 2018 г.. -

https://www.springer.com/cda/content/document/cda_downloaddocument/9783319022994-c1.pdf?SGWID=0-0-45-1446422-p175468473.

28. **NSJSONSerialization** [В Интернете] // Apple developer. - 20 Май 2018 г.. - https://developer.apple.com/documentation/foundation/nsjsonserialization?language=objc.
29. **One Millisecond Face Alignment with an Ensemble of Regression Trees** [В Интернете] / авт. Kazemi V. и Sullivan J.. - 2014 г.. - 25 Апрель 2018 г.. - http://www.csc.kth.se/~vahidk/papers/KazemiCVPR14.pdf.
30. **OpenFace** [В Интернете] // OpenFace. - 20 Май 2018 г.. - https://cmusatyalab.github.io/openface/.
31. **Support Vector Machines with Binary Tree Architecture for Multi-Class Classification** [В Интернете] / авт. Sungmoon C. и Sang H. // Semantic Scholar. - 1 Январь 2004 г.. - 25 Апрель 2018 г.. - https://pdfs.semanticscholar.org/7c2b/446a67d5ae23406f40b351d19a6fc6d728fe.pdf.
32. **Support-vector networks** [Журнал] / авт. Cortes C. и Vapnik V.. - [б.м.] : Machine Learning, 1995 г..
33. **Top 10 Facial Recognition APIs of 2018** [В Интернете] / авт. Walling A. // RapidAPI. - 10 Ноябрь 2017 г.. - 25 Апрель 2018 г.. - https://blog.rapidapi.com/2017/11/10/top-10-facial-recognition-apis-of-2017/.
34. **Top 10 Python Web Frameworks to Learn in 2018** [В Интернете] // hackernoon. - 20 Май 2018 г.. - https://hackernoon.com/top-10-python-web-frameworks-to-learn-in-2018-b2ebab969d1a.
35. **URLSession** [В Интернете] // Apple developer. - 20 Май 2018 г.. - https://developer.apple.com/documentation/foundation/urlsession.
36. **Vision** [В Интернете] / авт. Apple // Apple developer. - 20 Май 2018 г.. - https://developer.apple.com/documentation/vision.
37. **What is the best programming language for Machine Learning?** [В Интернете] / авт. Voskoglou Christina // Towards Datascience. - 20 Май 2018 г.. - https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7.