

**Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский технологический университет «МИСиС»**

ИНСТИТУТ ИТАСУ

КАФЕДРА Инженерной кибернетики

НАПРАВЛЕНИЕ 01.03.04 Прикладная математика

КУРСОВАЯ РАБОТА

по дисциплине «Разработка клиент-серверных приложений»

на тему:

Идентификация личности с помощью нейронных сетей на примере соц. сети

"ВКонтакте"

Студенты

Кущ А.А. Кальянова А.В.

Руководитель работы

Сержантова М.В.

Оценка

Институт ИТАСУ

Кафедра Инженерной кибернетики

Направление 01.03.04 Прикладная математика

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

по дисциплине " Разработка клиент-серверных приложений "

Студенту группы: БПМ-18-1 Кущ А.А.

1. Тема работы

Идентификация личности с помощью нейронных сетей на примере соц. сети "ВКонтакте"

2. Общая цель работы

Реализовать клиент-серверное приложение с использованием deep learning на основе CNN для идентификации человека в социальной сети ВКонтакте по любой фотографии лица (при условии, что человек зарегистрирован в ней)

3. Личная цель работы:

Обработка данных. Выбор лучшего алгоритма распознавания лица, подробное описание модели/архитектуры, реализация и внедрение в программу. Хранение данных. Выбор оптимальной СУБД для хранения данных.

4. Исходные данные

5. Основная литература.

6. Перечень основных этапов исследования и форма промежуточной отчетности по каждому этапу:
Актуальность темы, формулировка содержательной и математической постановок задачи, рассмотрение алгоритма решения задачи и техническая реализация

7. Использование ПО

Операционная система: любая

Используемые языки программирования: python

Среда разработки: telegram, vk

8. Перечень (примерный) основных вопросов, которые должны быть рассмотрены и проанализированы в литературном обзоре: особенности получения данных из ВКонтакте, способы распознавания лиц, модели нейросетей, наиболее подходящая СУБД для реализации приложения, особенности выдачи результата в Telegram.

9. Руководитель работы

(подпись)

Оглавление

1. Сбор данных	5
1.1. Работа с VK-api	5
1.1.1. Авторизация.	5
1.1.2. Формат сбора данных.	6
1.1.3. Варианты ошибок и их обработка.....	6
2. Обработка данных.....	8
2.1. Методы распознавания лиц	8
2.2.1. Elastic graph matching.....	8
2.2.2. Neural Network.....	9
2.2.3. CMM/HMM.....	10
2.2.4. PCA.....	11
2.2.5. AAM/ASM.	15
2.2.6. Выводы.....	18
2.2. Нейросеть.....	19
2.2.1 Типы нейросетей и задачи.	20
2.2.2. Convolutional Neural Network.....	22
2.2.3. Обзор популярных моделей и выбор лучшей.	29
2.2.3.1. Wide ResNet	29
2.2.3.2. Inception-ResNet.	32
2.2.3.3. Light CNN.	35
2.2.4. Реализация с помощью MTCNN + Inception-ResNet-V1.....	40
I этап.....	41
II этап.	43
III этап.	43
2.3. Сравнение с помощью евклидовой метрики.....	43
3. Хранение данных	44
3.1. Виды СУБД и их особенности.....	44
3.1.1. SQLite.....	44
3.1.2. MySQL.	45
3.1.3. PostgreSQL.....	47
3.2. Сравнение СУБД.....	50
3.3. Выбор СУБД.....	51
4. Выдача результата	53
4.1. Работа с Telegram-api.....	53
4.1.1. Авторизация.	53
4.2. Интерфейс.....	54

4.2. Обработка запроса/фотографии.....	55
5. Реализация	56
5.1. Краткое описание.....	56
Class Auth.....	56
Class ParsePageVK.....	56
Class DataBase.....	56
Class FinderVk.....	56
Class ResetDB.	56
Class TelegramBot.....	56
Файл private.py.	56
5.2. Telegram bot	57
Интерфейс.....	57
Литература.....	59

1. Сбор данных

Безусловно очень важной частью большинства алгоритмов и методов являются исходные данные. Так и в задаче распознавания лица неизбежно необходимы изображения лиц людей. Источником этих данных была выбрана социальная сеть «ВКонтакте». В ней для реализации сбора данных существует специальный интерфейс – API ВКонтакте, который позволяет получать данные из базы данных социальной сети. Что в свою очередь происходит с помощью отправки http-запросов к серверу.

1.1. Работа с VK-api

В то же время для работы с API существуют специальные обертки (wrappers), которые зачастую упрощают работу с ними и позволяют выполнять сложные задачи, состоящие из нескольких запросов в виде комбинации простых функций. API ВКонтакте не исключение и для него написано несколько таких вспомогательных оберток. Одна из наиболее популярных из них – это `vk_api` (модуль для языка Python).

Работа с API для сбора информации включает в себя несколько основных пунктов, которые рассмотрим далее:

- 1) Получение доступа к информации
- 2) Сбор данных
- 3) Обработка возможных ошибок при этом взаимодействии

1.1.1. Авторизация.

Для получения доступа ко всем методам API необходимо получить ключ доступа (access token), который необходимо передавать в запросах. Этот ключ представляет собой строку из латинских букв и цифр и позволяет не передавать логин и пароль приложению, за счет чего аккаунт не может быть скомпрометирован.

Существует несколько возможных способа получения ключа доступа:

- 1) Implicit flow – самый простой и короткий вариант. Ключ возвращается на устройство пользователя в виде дополнительного параметра URL и может быть использован непосредственного с данного устройства. Использует авторизацию пользователя.
- 2) Authorization code flow – используется для вызова методов API ВКонтакте с серверной части приложения (например, из PHP). Полученный таким способом ключ не привязан к определенному IP-адресу, но набор прав ограничен из соображений безопасности.
- 3) Client credentials flow – авторизация по секретному ключу приложения. Содержит ряд ограничений. Используется только в случае, когда не требуется авторизация пользователя или сообщества.

1.1.2. Формат сбора данных.

Перед сбором данных нужно определиться с данными, которые необходимы для конкретной конечной цели. В случае решения задачи нахождения человека в социальной сети необходимы фотографии человека и несколько его основных данных, таких как: пол и id страницы.

При выборе необходимых фотографий, нужно учитывать несколько основных аспектов, которые могут повлиять на обработку фотографии:

- 1) В идеале на фотографии должен присутствовать только 1 человек – владелец страницы. В противном случае произойдет зашумление (ошибочное распознавание), что негативно скажется на работе алгоритма.
- 2) Если страница открыта, выборка фотографий происходит из альбомов. Нужно выбрать альбомы, в которых более вероятно будут фотографии пользователя. В большинстве случаев такие фотографии находятся в альбомах «фотографии со страницы» или «фотографии со стены».
- 3) Если на фотографии нет лиц, то необходимо выбрать другую фотографию.
- 4) Когда страница закрыта, единственным источником данных является главная фотография на странице пользователя.

Порой количество фотографий на странице человека может достигать нескольких тысяч. Такой объем данных займет слишком много памяти и потребует много времени для их процессинга. К тому же, большая часть таких фотографий может быть старой и не отображать того, как выглядит человек на данный момент. Ограничение в количестве используемых фотографий поможет решить эту проблему. В то же время фотографий не должно быть слишком мало, иначе верное определение человека будет менее вероятным. Поэтому оптимальным будет использовать, например, только 10 наиболее актуальных и удовлетворяющих ранее описанным признакам фотографий, а остальные не рассматривать.

Знание пола человека в свою очередь убыстряет распознавание человека, так как можно сразу отбросить значительную часть пользователей противоположного пола. Но с другой стороны, нейросеть не всегда может корректно определить пол человека, что полностью исключит корректность работы алгоритма. Данной идеи требует доработки и реализована не была.

1.1.3. Варианты ошибок и их обработка.

При работе с API ВКонтакте следует учесть несколько моментов, которые могут привести к ошибкам в работе программы. Обработка производится по номеру ошибки. Условно их можно разделить на две основные группы:

- 1) Ошибки при авторизации и технические ошибки
- 2) Ошибки при получении необходимых данных

В первом случае это, прежде всего, неверный выбор способа получения ключа доступа. Такая ошибка обычно следует из необходимости учитывать подключена ли двухфакторная аутентификация на странице, с помощью которой получен ключ доступа, или достаточно использовать «implicit flow» для получения ключа доступа.

Слишком большое количество запросов в секунду тоже может привести к ошибке. Тогда нужно задать больший интервал между вызовами. Множество однотипных обращений и множество вызовов одного метода одновременно лучше не использовать, так как это тоже приводит к ошибкам.

При получении данных следует обратить внимание на то, что:

- 1) Страница пользователя может быть закрытой (профиль является приватным). В данном случае нельзя получить доступ к альбомам и большинству данных. Соответственно, обработать можно только основную информацию страницы и главную фотографию на ней.
- 2) Информация большинства страниц не отображается для незарегистрированных пользователей. Из-за этого подключение вида «client credentials flow» не даст результата в этом случае.
- 3) Удаленная или заблокированная страница не может быть обработана.

Для обработки ошибок, связанных с получением данных, достаточно добавить модули, которые будут их «ловить», и в зависимости от этого обрабатывать сбор данных соответствующим образом. Также при обработке ошибок будет полезно знать номер, который возвращается при их появлении. Вот основные из тех, которые могут быть получены:

3 – неверное название вызываемого метода
5 – авторизация не удалась (скорее всего из-за неверно выбранной схемы авторизации)
6 – слишком много запросов в секунду
8 – неверный синтаксис или список параметров запроса
15 – доступ к контенту запрещен
20 – действие запрещено для не Standalone приложений
21 – ключ доступа приложения недействителен
29 – достигнут лимит на вызов метода
30 – профиль является приватным и поэтому информация недоступна с используемым ключом доступа
200 – доступ к альбому запрещен
3301 – необходимо использование двухфакторной аутентификации

2. Обработка данных

При сборе любой информации немаловажную роль играет обработка данных. Далее будут рассмотрены различные эффективные методы преобразования данных с целью выявления характерных признаков индивидуальности лица каждого человека.

2.1. Методы распознавания лиц

2.2.1. Elastic graph matching.

Суть метода сводится к эластичному сопоставлению графов, описывающих изображения лиц. Лица представлены в виде графов со взвешенными вершинами и ребрами. На этапе распознавания один из графов – эталонный – остается неизменным, в то время как другой деформируется с целью наилучшей подгонки к первому. В подобных системах распознавания графы могут представлять собой как прямоугольную решетку, так и структуру, образованную характерными (антропометрическими) точками лица.

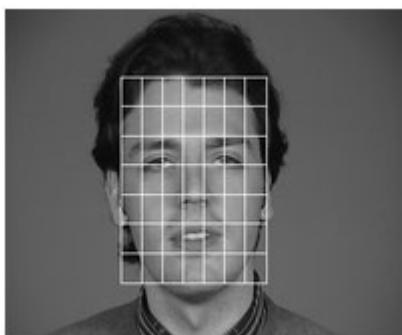


Рисунок 2. регулярная решетка

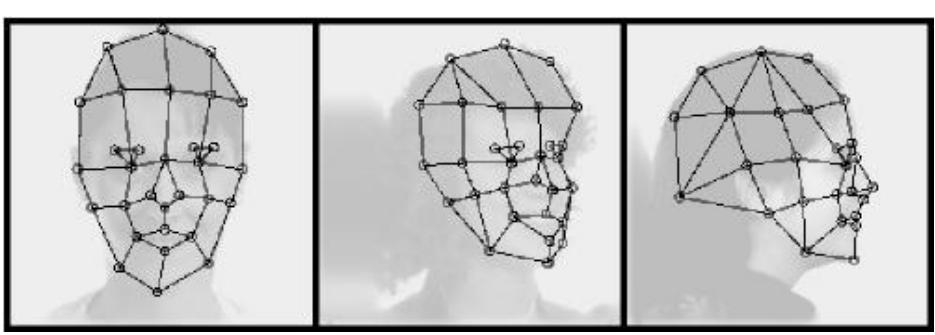


Рисунок 1. граф на основе антропометрических точек лица

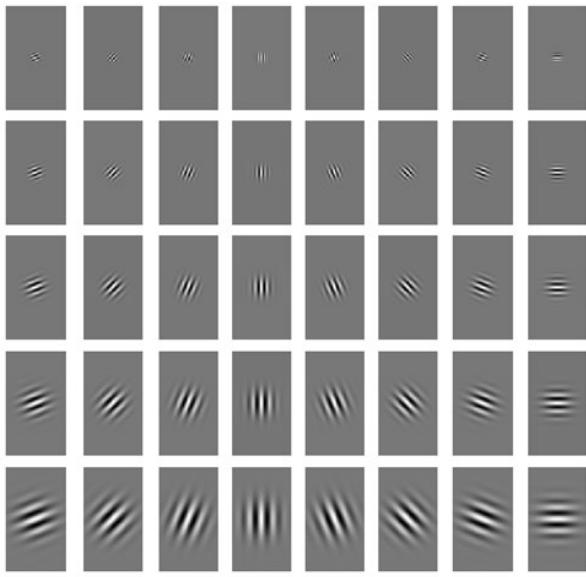


Рисунок 3. фильтры Габора

В вершинах графа вычисляются значения признаков, чаще всего используют комплексные значения фильтров Габора или их упорядоченных наборов – Габоровских вейвлет (строи Габора), которые вычисляются в некоторой локальной области вершины графа локально путем свертки значений яркости пикселей с фильтрами Габора.

Ребра графа взвешиваются расстояниями между смежными вершинами. Различие (расстояние, дискриминационная характеристика) между двумя графиками вычисляется при помощи некоторой ценовой функции деформации, учитывающей как различие между значениями признаков, вычисленными в вершинах, так и степень деформации ребер графа.

Деформация графа происходит путем смещения каждой из его вершин на некоторое расстояние в определённых направлениях относительно ее исходного местоположения и выбора такой ее позиции, при которой разница между значениями признаков (откликов фильтров Габора) в вершине деформируемого графа и соответствующей ей вершине эталонного графа будет минимальной. Данная операция выполняется поочередно для всех вершин графа до тех пор, пока не будет

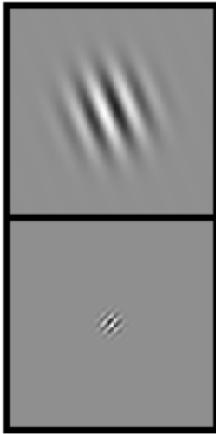
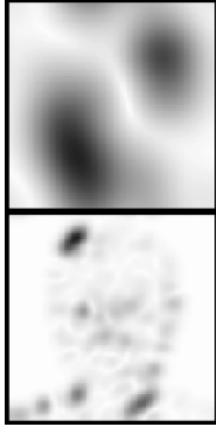
Входное изображение	Пример двух фильтров Габора	Результат свертки входного изображения лица и фильтров Габора
		

Рисунок 4. пример свертки изображения лица с двумя фильтрами Габора



Рисунок 5. пример деформации графа в виде регулярной решетки

В отдельных публикациях указывается 95-97%-ая эффективность распознавания даже при наличии различных эмоциональных выражениях и изменении ракурса лица до 15 градусов. Однако разработчики систем эластичного сравнения на графах ссылаются на высокую вычислительную стоимость данного подхода. Например, для сравнения входного изображения лица с 87 эталонными тратилось приблизительно 25 секунд при работе на параллельной ЭВМ с 23 транспьютерами [15] (Примечание: публикация датирована 1993 годом). В других публикациях по данной тематике время либо не указывается, либо говорится, что оно велико.

Недостатки: высокая вычислительная сложность процедуры распознавания. Низкая технологичность при запоминании новых эталонов. Линейная зависимость времени работы от размера базы данных лиц.

2.2.2. Neural Network.

В настоящее время существует около десятка разновидности нейронных сетей. Одним из самых широко используемых вариантов является сеть, построенная на многослойном перцептроне, которая позволяет классифицировать поданное на вход изображение/сигнал в соответствии с предварительной настройкой/обучением сети.

Обучаются нейронные сети на наборе обучающих примеров. Суть обучения сводится к настройке весов межнейронных связей в процессе решения оптимизационной задачи методом градиентного спуска. В процессе обучения НС происходит автоматическое извлечение ключевых признаков, определение их важности и построение взаимосвязей между ними. Предполагается, что обученная НС сможет применить опыт, полученный в процессе обучения, на неизвестные образы за счет обобщающих способностей.

достигнуто наименьшее суммарное различие между признаками деформируемого и эталонного графов. Значение ценовой функции деформации при таком положении деформируемого графа и будет являться мерой различия между входным изображением лица и эталонным графиком. Данная «релаксационная» процедура деформации должна выполняться для всех эталонных лиц, заложенных в базу данных системы. Результат распознавания системы – эталон с наилучшим значением ценовой функции деформации.

Наилучшие результаты в области распознавания лиц (по результатам анализа публикаций) показала Convolutional Neural Network, которая является логическим развитием идей таких архитектур НС как когнитрона и неокогнитрона. Успех обусловлен возможностью учета двумерной топологии изображения, в отличие от многослойного перцептрона.

Отличительными особенностями CNN являются локальные рецепторные поля (обеспечивают локальную двумерную связность нейронов), общие веса (обеспечивают детектирование некоторых черт в любом месте изображения) и иерархическая организация с пространственными сэмплингом (spatial subsampling). Благодаря этим нововведениям CNN обеспечивает частичную устойчивость к изменениям масштаба, смещениям, поворотам, смене ракурса и прочим искажениям.

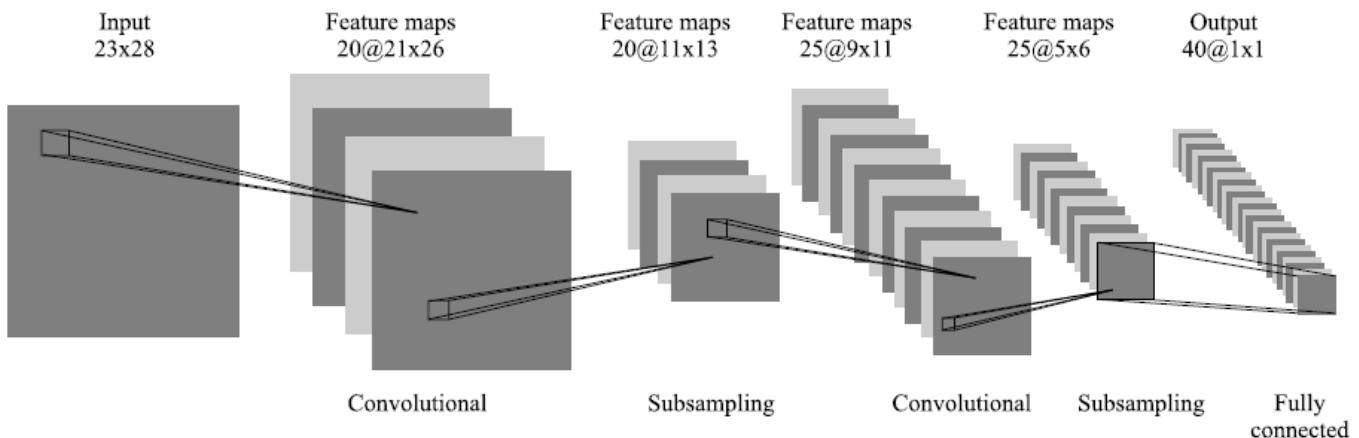
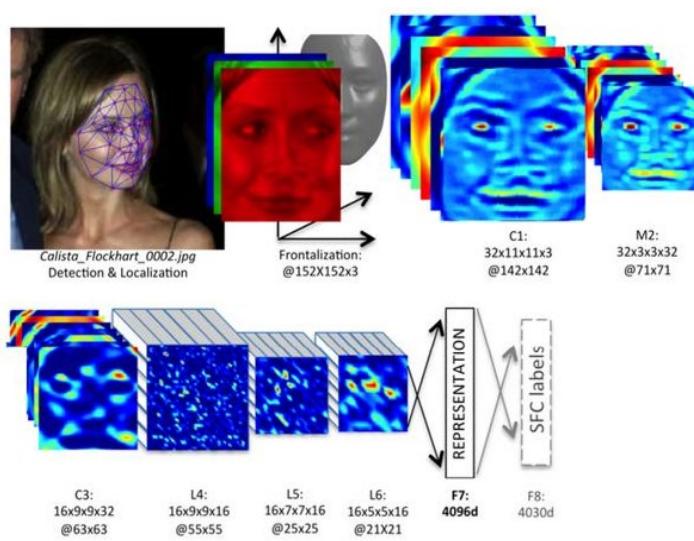


Рисунок 6. схематичное изображение CNN

Тестирование CNN на базе данных ORL, содержащей изображения лиц с небольшими изменениями освещения, масштаба, пространственных поворотов, положения и различными эмоциями, показало 96% точность распознавания.

Свое развитие CNN получили в разработке DeepFace, которую приобрел Facebook для распознавания лиц пользователей своей соцсети. Все особенности архитектуры носят закрытый характер.



Недостатки нейронных сетей: добавление нового эталонного лица в базу данных требует полного переобучения сети на всем имеющемся наборе (достаточно длительная процедура, в зависимости от размера выборки от 1 часа до нескольких дней). Проблемы математического характера, связанные с обучением: попадание в локальный оптимум, выбор оптимального шага оптимизации, переобучение и т. д. Трудно формализуемый этап выбора архитектуры сети (количество нейронов, слоев, характер связей). Обобщая все вышесказанное, можно заключить, что НС – «черный ящик» с трудно интерпретируемыми результатами работы.

2.2.3. СММ/НММ.

Одним из статистических методов распознавания лиц являются скрытые Марковские модели (СММ) с дискретным временем. СММ используют статистические свойства сигналов и учитывают

непосредственно их пространственные характеристики. Элементами модели являются: множество скрытых состояний, множество наблюдаемых состояний, матрица переходных вероятностей, начальная вероятность состояний. Каждому соответствует своя Марковская модель. При распознавании объекта проверяются сгенерированные для заданной базы объектов Марковские модели и ищется максимальная из наблюдаемых вероятность того, что последовательность наблюдений для данного объекта сгенерирована соответствующей моделью.

На сегодняшний день не удалось найти примера коммерческого применения СММ для распознавания лиц.

Недостатки:

- необходимо подбирать параметры модели для каждой базы данных;
- СММ не обладает различающей способностью, то есть алгоритм обучения только максимизирует отклик каждого изображения на свою модель, но не минимизирует отклик на другие модели.

2.2.4. PCA.

Одним из наиболее известных и проработанных является метод главных компонент (principal component analysis, PCA), основанный на преобразовании Карунена-Лоева.

Первоначально метод главных компонент начал применяться в статистике для снижения пространства признаков без существенной потери информации. В задаче распознавания лиц его применяют главным образом для представления изображения лица вектором малой размерности (главных компонент), который сравнивается затем с эталонными векторами, заложенными в базу данных.

Главной целью метода главных компонент является значительное уменьшение размерности пространства признаков таким образом, чтобы оно как можно лучше описывало «типичные» образы, принадлежащие множеству лиц. Используя этот метод можно выявить различные изменчивости в обучающей выборке изображений лиц и описать эту изменчивость в базисе нескольких ортогональных векторов, которые называются собственными (eigenface).

Полученный один раз на обучающей выборке изображений лиц набор собственных векторов используется для кодирования всех остальных изображений лиц, которые представляются взвешенной комбинацией этих собственных векторов. Используя ограниченное количество собственных векторов, можно получить сжатую аппроксимацию входному изображению лица, которую затем можно хранить в базе данных в виде вектора коэффициентов, служащего одновременно ключом поиска в базе данных лиц.

Суть метода главных компонент сводится к следующему. Вначале весь обучающий набор лиц преобразуется в одну общую матрицу данных, где каждая строка представляет собой один экземпляр изображения лица, разложенного в строку. Все лица обучающего набора должны быть приведены к одному размеру и с нормированными гистограммами.

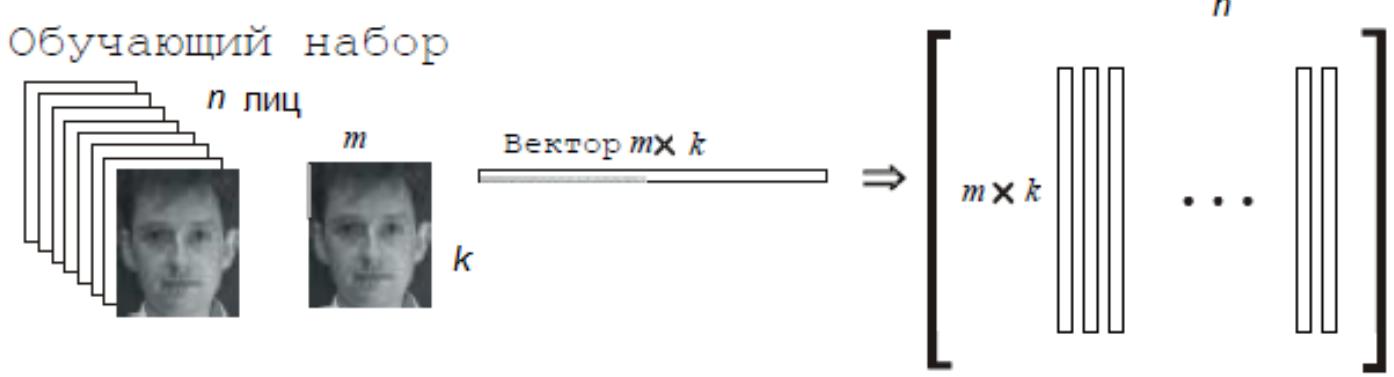


Рисунок 7. Преобразования обучающего набора лиц в одну общую матрицу X

Затем производится нормировка данных и приведение строк к 0-му среднему и 1-й дисперсии, вычисляется матрица ковариации. Для полученной матрицы ковариации решается задача определения собственных значений и соответствующих им собственных векторов (собственные лица). Далее производится сортировка собственных векторов в порядке убывания собственных значений и оставляют только первые k векторов по правилу:

$$\frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^n \lambda_i} > \text{Threshold (0.9 or 0.95)}$$

Principal Component Analysis

Covariance Algorithm

1. Zero mean: $x_{ij} = x_{ij} - \mu_i$ $\mu_i = \frac{1}{n} \sum_{j=1}^n x_{ij}$
2. Unit variance: $x_{ij} = \frac{x_{ij}}{\sigma_j}$ $\sigma_i = \sqrt{\frac{1}{n-1} \sum_{j=1}^n (x_{ij} - \mu_i)^2}$
3. Covariance matrix: $\Sigma = \mathbf{X}\mathbf{X}^T$
4. Compute eigenvectors of Σ : \mathbf{W}^T
5. Project \mathbf{X} onto the k principal components

$$\begin{array}{c} \boxed{\begin{array}{c} \parallel \\ \parallel \\ \parallel \end{array} \mathbf{Y} \\ k \times n } \\ = \end{array} \begin{array}{c} \boxed{\begin{array}{c} \parallel \\ \parallel \\ \parallel \end{array} \mathbf{W}^T} \\ k \times d \end{array} \times \begin{array}{c} \boxed{\begin{array}{c} \parallel \\ \parallel \\ \parallel \end{array} \mathbf{X}} \\ d \times n \end{array}$$

$k < d$

Алгоритм РСА

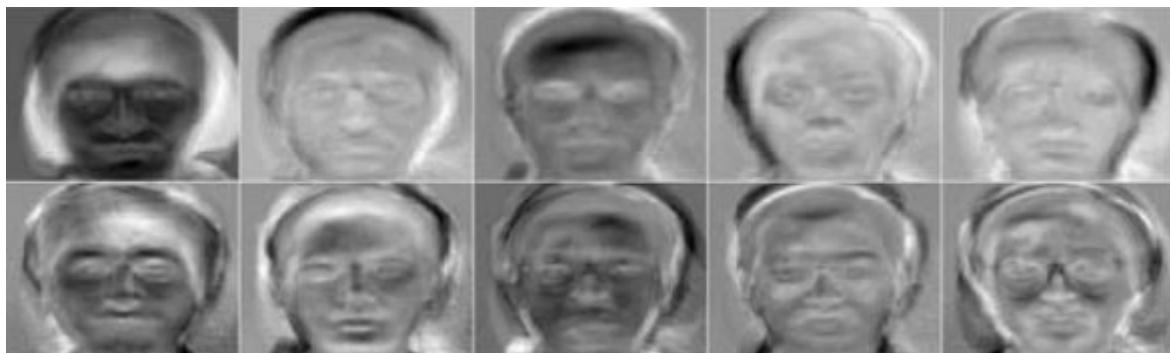


Рисунок 8. Пример первых десяти собственных векторов (собственных лиц), полученных на обучаемом наборе лиц

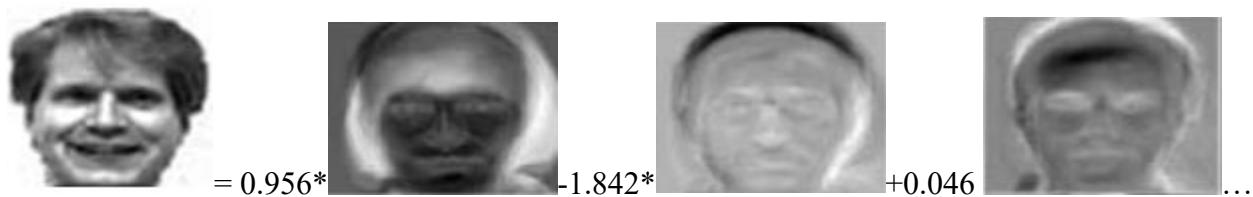
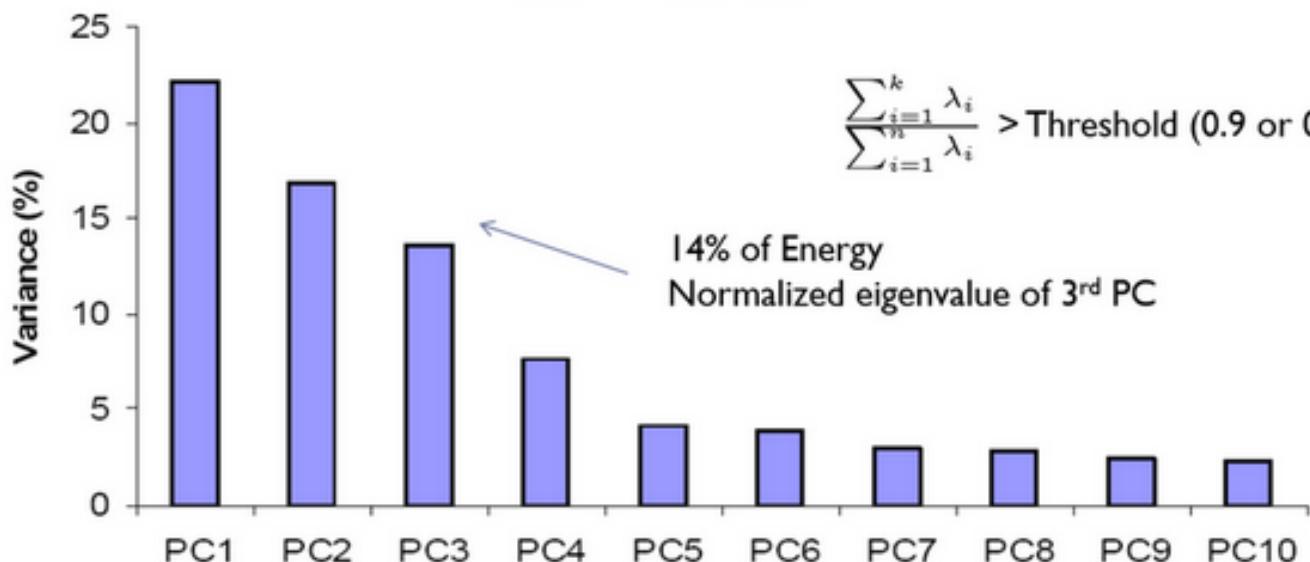


Рисунок 9. Пример построения (синтеза) человеческого лица с помощью комбинации собственных лиц и главных компонент

▶ How many PCs?

$$\mathbf{Y} = \mathbf{WX}$$

$k \times n$ $k \times d$ $d \times n$



$$\frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^d \lambda_i} > \text{Threshold (0.9 or 0.95)}$$

There are d eigenvectors

Choose first p eigenvectors based on their eigenvalues

Final data has only k dimensions

Рисунок 11. Принцип выбора базиса из первых лучших собственных векторов

Dimensionality Reduction

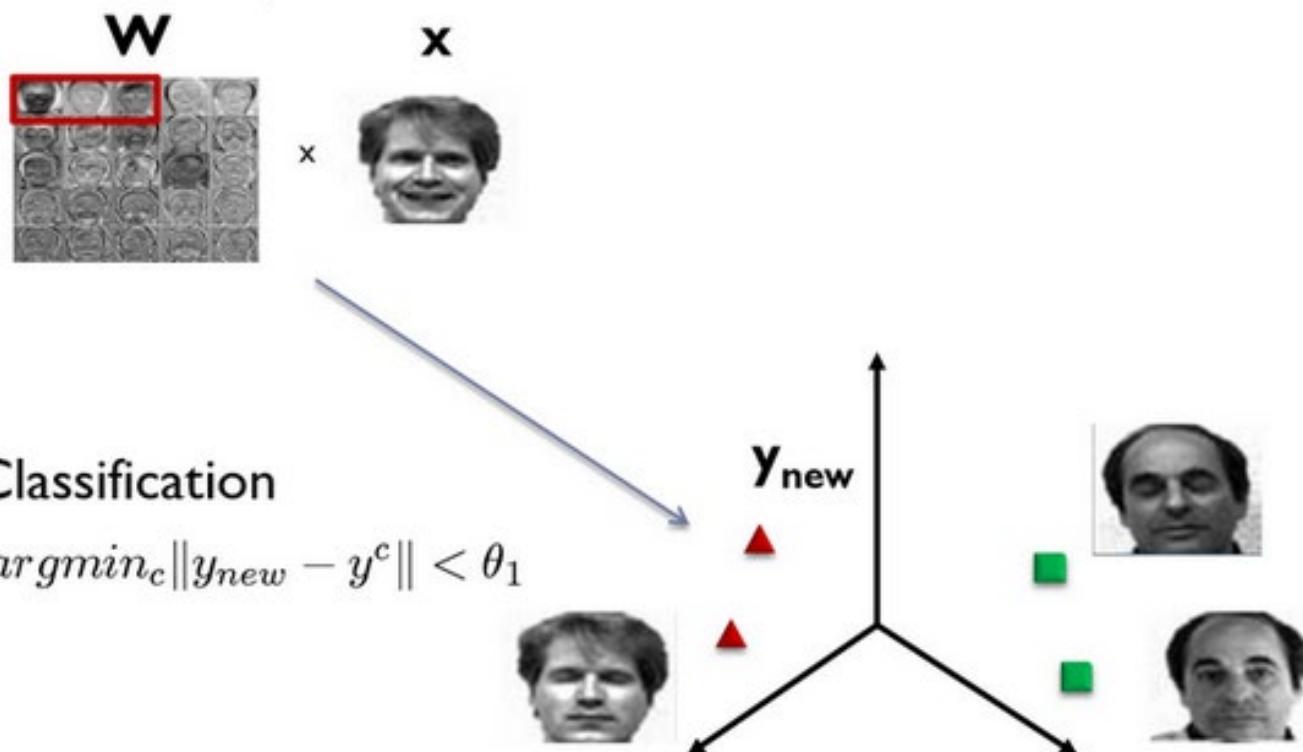
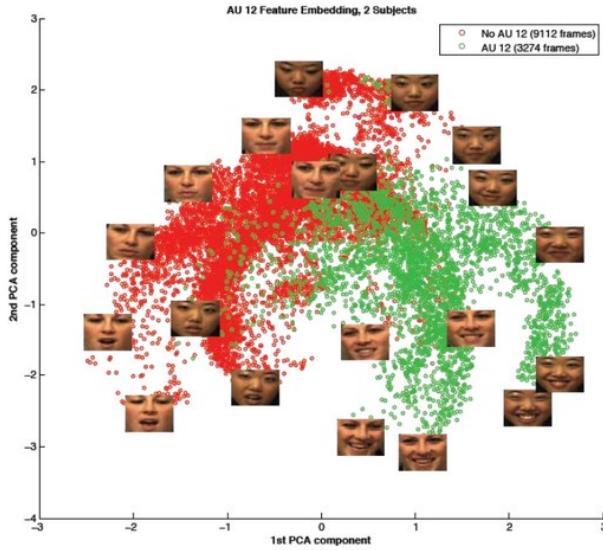


Рисунок 10. Пример отображения лица в трехмерное метрическое пространство, полученное по трем собственным лицам и дальнейшее распознавание



Метод главных компонент хорошо зарекомендовал себя в практических приложениях. Однако, в тех случаях, когда на изображении лица присутствуют значительные изменения в освещенности или выражении лица, эффективность метода значительно падает. Все дело в том, что РСА выбирает подпространство с такой целью, чтобы максимально аппроксимировать входной набор данных, а не выполнить дискриминацию между классами лиц.

Было предложено решение этой проблемы с использованием линейного дискриминанта Фишера (в литературе встречается название “Eigen-Fisher”, “Fisherface”, LDA). LDA выбирает линейное подпространство, которое максимизирует отношение:

$$\frac{\Phi^T S_b \Phi}{\Phi^T S_w \Phi}, \text{ где } S_b = \sum_{i=1}^m N_i (\bar{x}_i - \bar{x})(\bar{x}_i - \bar{x})^T \text{ – матрица межклассового разброса;}$$

$$S_w = \sum_{i=1}^m \sum_{x \in X_i} (\bar{x}_i - \bar{x})(\bar{x}_i - \bar{x})^T \text{ – матрица сежклассового разброса.}$$

Матрица внутриклассового разброса; m – число классов в базе данных.

LDA ищет проекцию данных, при которой классы являются максимально линейно сепарабельны (см. рисунок ниже). Для сравнения РСА ищет такую проекцию данных, при которой будет максимизирован разброс по всей базе данных лиц (без учета классов). По результатам экспериментов в условиях сильного бакового и нижнего затенения изображений лиц Fisherface показал 95% эффективность по сравнению с 53% Eigenface.

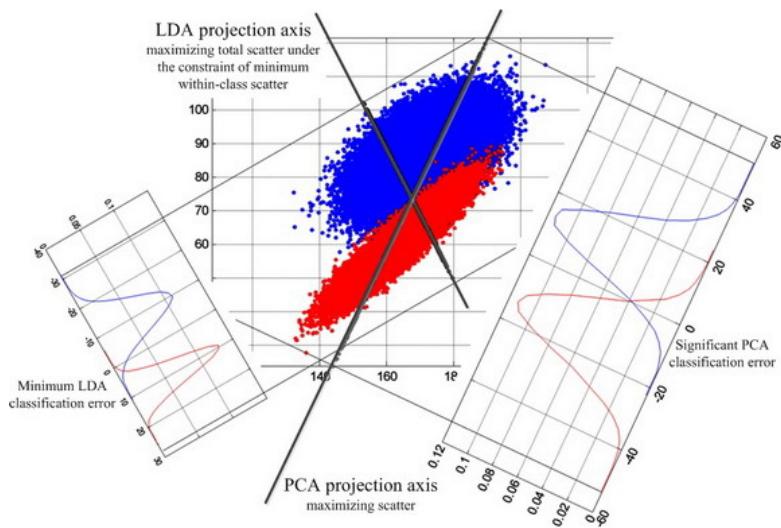


Рисунок 12. Принципиальное отличие формирования проекций РСА и LDA

Отличие PCA от LDA:

- PCA: выполняет уменьшение размерности, сохраняя при этом как можно больше дисперсии в пространстве с высокой размерностью.
- LDA: выполняет уменьшение размерности, сохраняя при этом как можно больше информации о классовой дискриминации.

2.2.5. AAM/ASM.

Active Appearance Models (AAM)

Активные модели внешнего вида — это статистические модели изображений, которые путем разного рода деформаций могут быть подогнаны под реальное изображение. Данный тип моделей в двумерном варианте был предложен Тимом Кутсом и Крисом Тейлором в 1998 году. Первоначально активные модели внешнего вида применялись для оценки параметров изображений лиц.

Активная модель внешнего вида содержит два типа параметров: параметры, связанные с формой (параметры формы), и параметры, связанные со статистической моделью пикселей изображения или текстурой (параметры внешнего вида). Перед использованием модель должна быть обучена на множестве заранее размеченных изображений. Разметка изображений производится вручную. Каждая метка имеет свой номер и определяет характерную точку, которую должна будет находить модель во время адаптации к новому изображению.

Процедура обучения ААМ начинается с нормализации форм на размеченных изображениях с целью компенсации различий в масштабе, наклоне и смещении. Для этого используется так называемый обобщенный Прокрустов анализ.

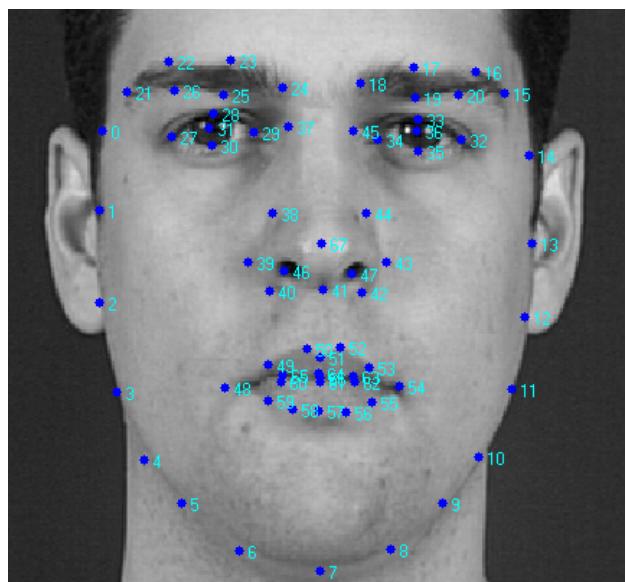


Рисунок 14. Пример разметки изображения лица из 68 точек, образующих форму AAM.

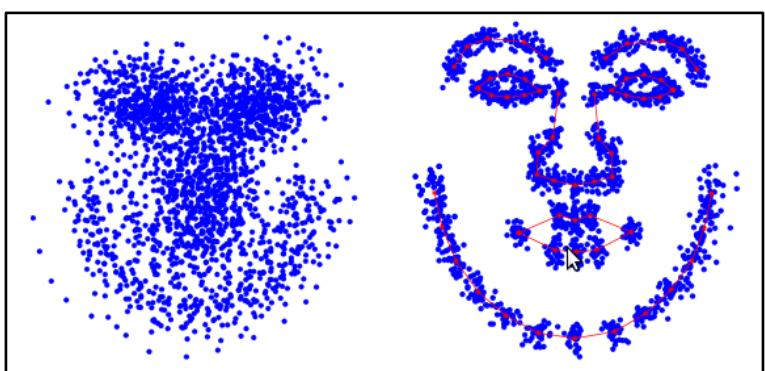


Рисунок 13. Координаты точек формы лица до и после нормализации

Из всего множества нормированных точек затем выделяются главные компоненты с использованием метода PCA.

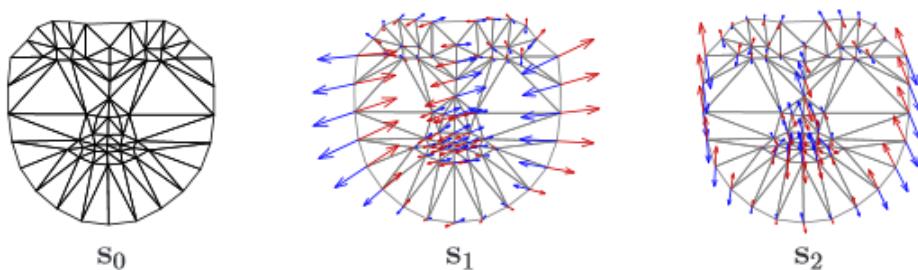


Рисунок 15. Модель формы AAM состоит из триангуляционной решетки S_0 и линейной комбинации смещений S относительно S_0 .

Далее из пикселей внутри треугольников, образуемых точками формы, формируется матрица, такая что, каждый ее столбец содержит значения пикселей соответствующей текстуры. Стоит отметить, что используемые для обучения текстуры могут быть как одноканальными (градации серого), так и многоканальными (например, пространство цветов RGB или другое). В случае многоканальных текстур векторы пикселов формируются отдельно по каждому из каналов, а потом выполняется их конкатенация. После нахождения главных компонент матрицы текстур модель ААМ считается обученной.

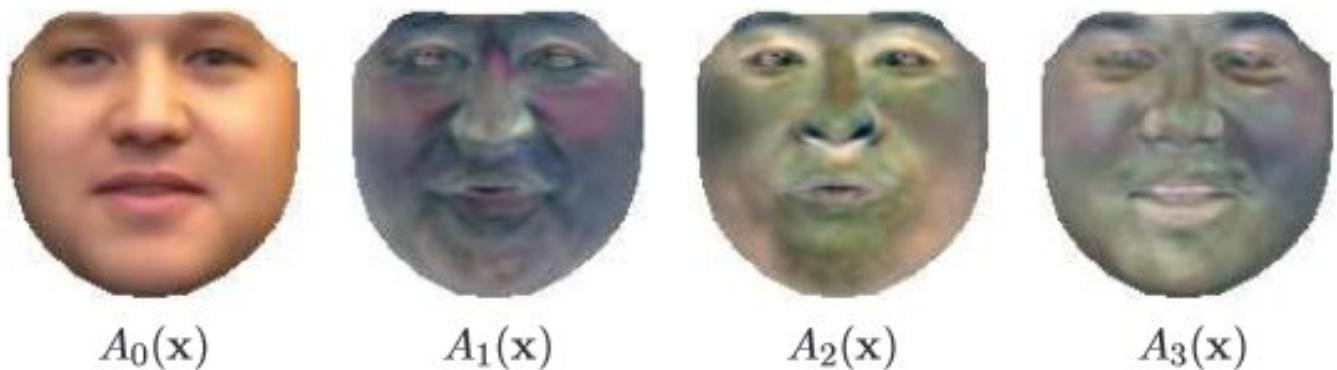


Рисунок 17. Модель внешнего вида ААМ состоит из базового вида A_0 , определенного пикселями внутри базовой решетки S_0 и линейной комбинации A_i относительно A_0

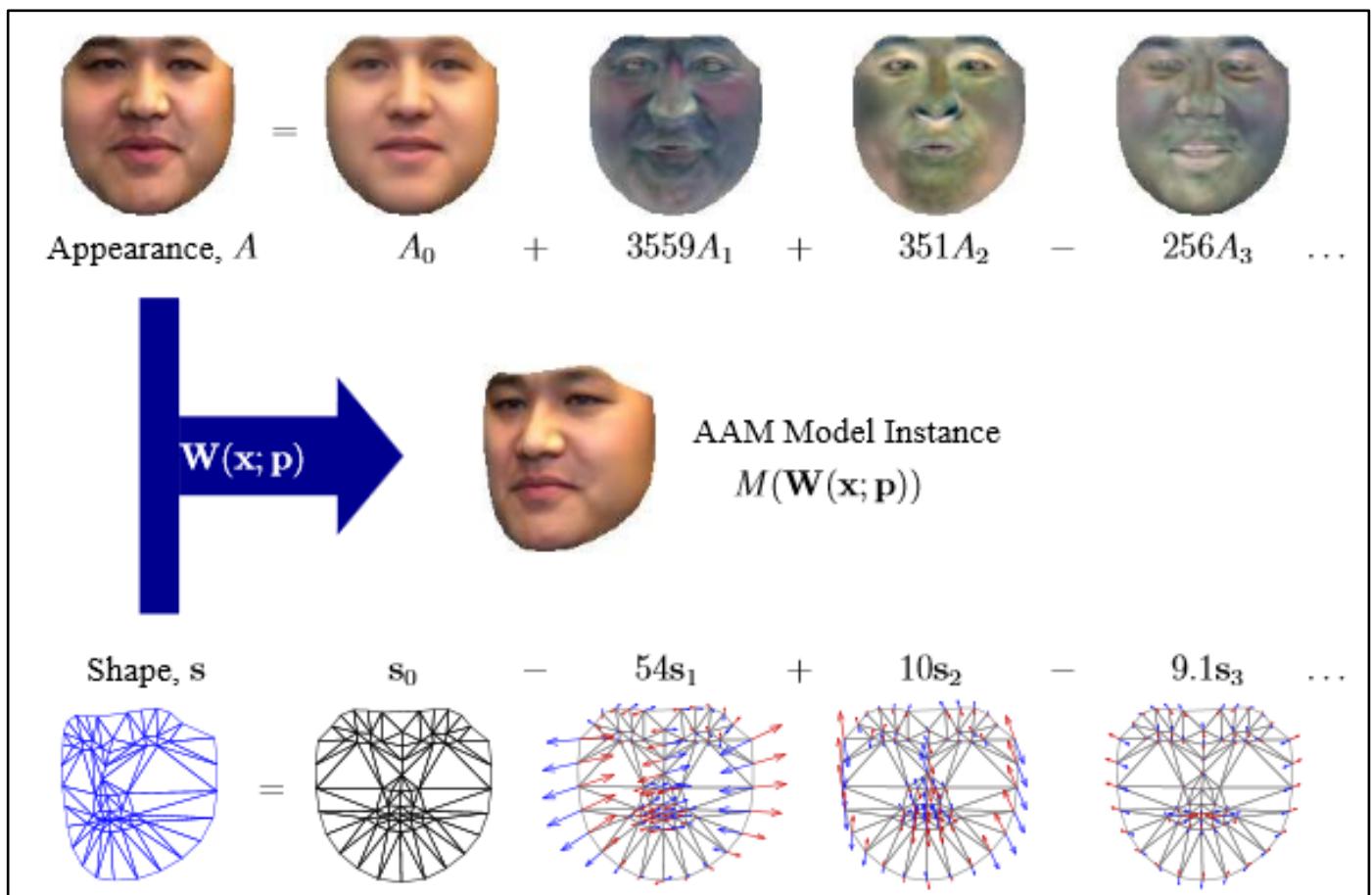


Рисунок 16. Пример конкретизации ААМ.

Итоговая модель лица получается, как комбинация двух моделей – формы и внешнего вида. Подгонка модели под конкретное изображение лица выполняется в процессе решения оптимизационной задачи, суть которой сводится к минимизации функционала методом градиентного спуска. Найденные при этом параметры модели и будут отражать положение модели на конкретном изображении.

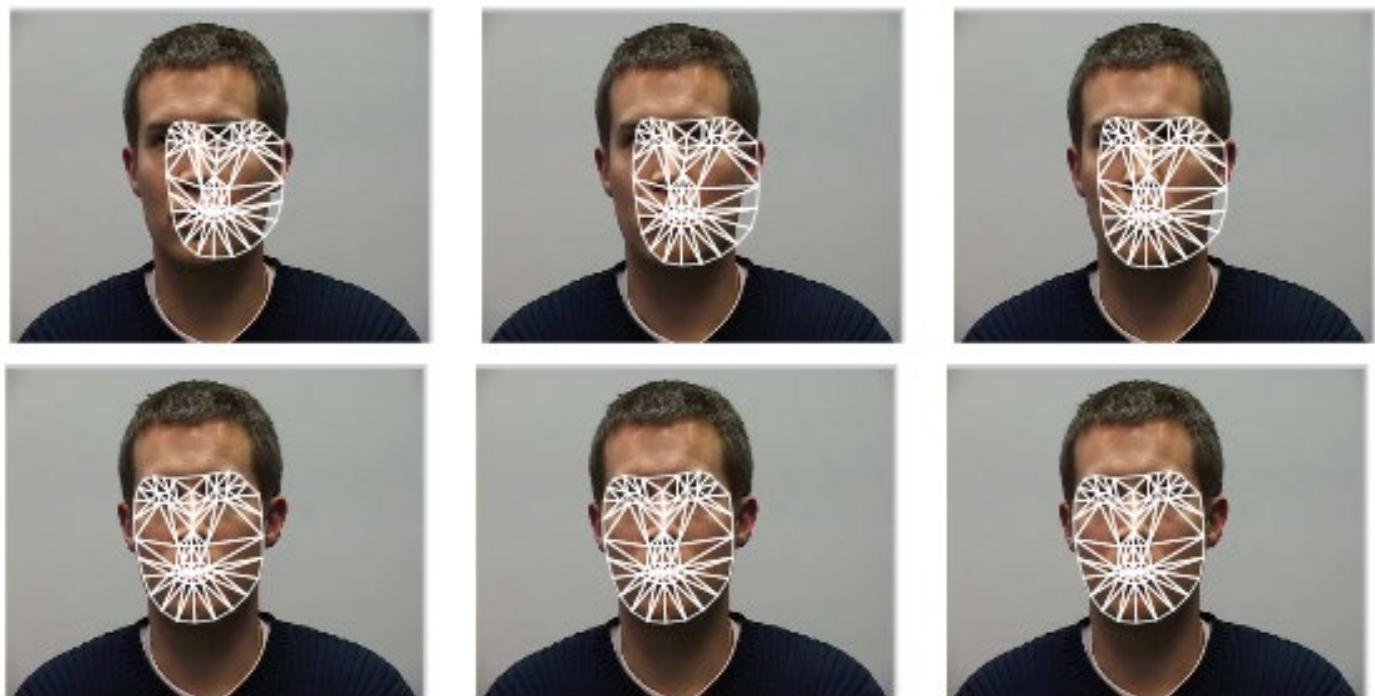


Рисунок 18. Пример подгонки модели на конкретное изображение за 20 итераций процедуры градиентного спуска.

С помощью ААМ можно моделировать изображения объектов, подверженных как жесткой, так и нежесткой деформации. ААМ состоит из набора параметров, часть которых представляют форму лица, остальные задают его текстуру. Под деформации обычно понимают геометрическое преобразование в виде композиции переноса, поворота и масштабирования. При решении задачи локализации лица на изображении выполняется поиск параметров (расположение, форма, текстура) ААМ, которые представляют синтезируемое изображение, наиболее близкое к наблюдаемому. По степени близости ААМ подгоняющему изображению принимается решение – есть лицо или нет.

Active Shape Models (ASM)

Суть метода ASM заключается в учете статистических связей между расположением антропометрических точек. На имеющейся выборке изображений лиц, снятых в анфас. На изображении эксперт размечает расположение антропометрических точек. На каждом изображении точки пронумерованы в одинаковом порядке.

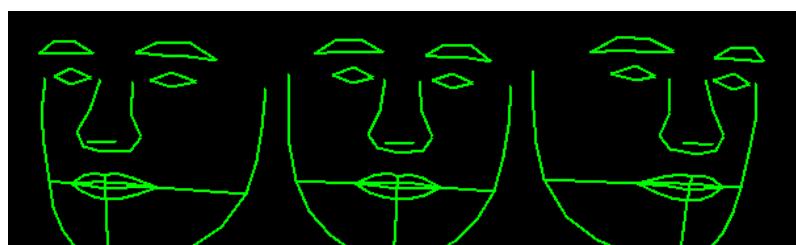
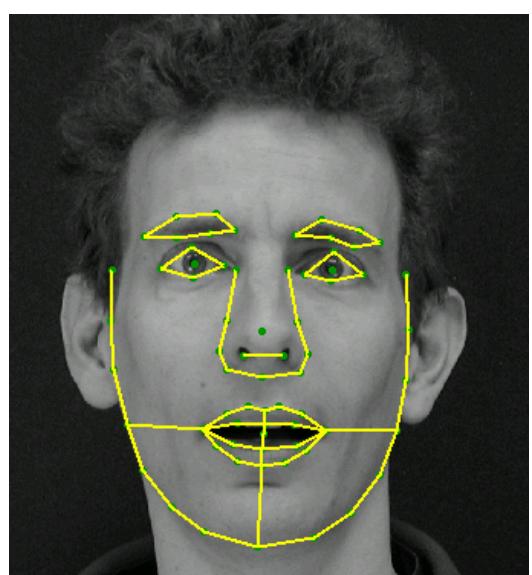


Рисунок 19. Пример представления формы лица с использованием 68 точек



Для того чтобы привести координаты на

всех изображениях к единой системе обычно выполняется т.н. обобщенный прокрустов анализ, в результате которого все точки приводятся к одному масштабу и центрируются. Далее для всего набора образов вычисляется средняя форма и матрица ковариации. На основе матрицы ковариации вычисляются собственные вектора, которые затем сортируются в порядке убывания соответствующих им собственных значений. Модель ASM определяется матрицей Φ и вектором средней формы \bar{s} .

Тогда любая форма может быть описана с помощью модели и параметров:

$$b_i = \Phi^T \bar{s}_i = \Phi^T (s_i - \bar{s})$$

Локализации ASM модели на новом, не входящем в обучающую выборку изображении осуществляется в процессе решения оптимизационной задачи.

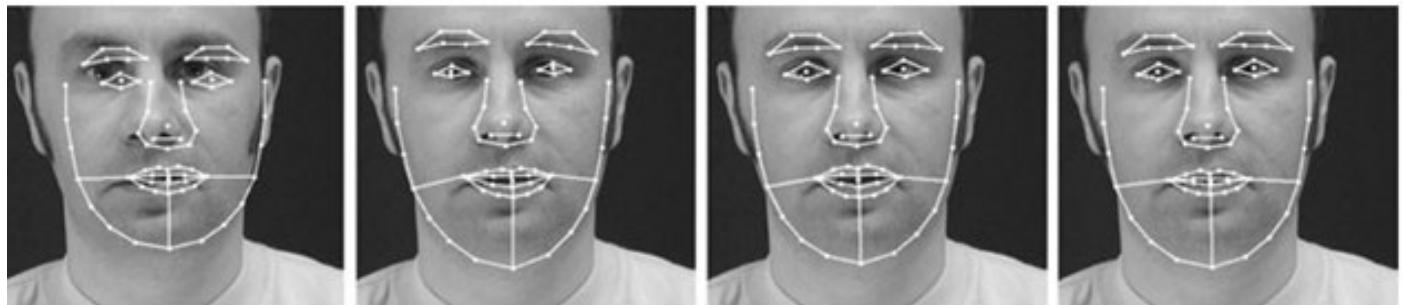


Рисунок 20. Иллюстрация процесса локализации модели ASM на конкретном изображении: а) начальное положение б) после 5 итераций в) после 10 итераций г) модель сошлась

Однако все же главной целью ААМ и ASM является не распознавание лиц, а точная локализация лица и антропометрических точек на изображении для дальнейшей обработки.

Практически во всех алгоритмах обязательным этапом, предваряющим классификацию, является выравнивание, под которым понимается выравнивание изображения лица во фронтальное положение относительно камеры или приведение совокупности лиц (например, в обучающей выборке для обучения классификатора) к единой системе координат. Для реализации этого этапа необходима локализация на изображении характерных для всех лиц антропометрических точек – чаще всего это центры зрачков или уголки глаз. Разные исследователи выделяют разные группы таких точек. В целях сокращения вычислительных затрат для систем реального времени разработчики выделяют не более 10 таких точек.

Модели ААМ и ASM как раз и предназначены для того, чтобы точно локализовать эти антропометрические точки на изображении лица.

2.2.6. Выводы.

Основные проблемы:

- a) Проблема освещенности;
- b) Проблема положения головы;
- c) Проблема нестабильности участков лица (эмоции, взросление).

Данные проблемы по-отдельности возможно решить с помощью простых и эффективных алгоритмов, расписанных выше. Но в совокупности все – только сверточная нейросеть (CNN).

2.2. Нейросеть

Нейронная сеть — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы. После разработки алгоритмов обучения получаемые модели стали использовать в практических целях: в задачах прогнозирования, для распознавания образов, в задачах управления и др.

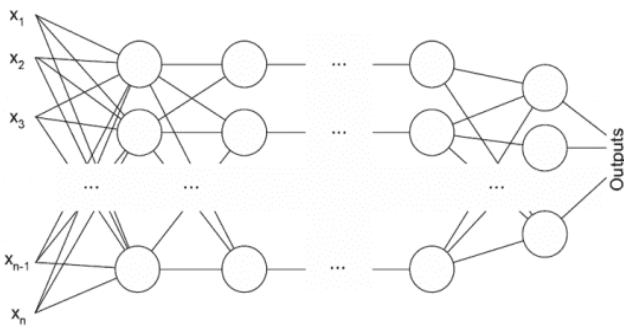
НС представляет собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Такие процессоры обычно довольно просты (особенно в сравнении с процессорами, используемыми в персональных компьютерах). Каждый процессор подобной сети имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим процессорам. И, тем не менее, будучи соединёнными в достаточно большую сеть с управляемым взаимодействием, такие по отдельности простые процессоры вместе способны выполнять довольно сложные задачи.

- С точки зрения машинного обучения, нейронная сеть представляет собой частный случай методов распознавания образов, дискриминантного анализа, методов кластеризации и т. п.
- С точки зрения математики, обучение нейронных сетей — это многопараметрическая задача нелинейной оптимизации.
- С точки зрения кибернетики, нейронная сеть используется в задачах адаптивного управления и как алгоритмы для робототехники.
- С точки зрения развития вычислительной техники и программирования, нейронная сеть — способ решения проблемы эффективного параллелизма.
- С точки зрения искусственного интеллекта, НС является основой философского течения коннекционизма и основным направлением в структурном подходе по изучению возможности построения (моделирования) естественного интеллекта с помощью компьютерных алгоритмов.

Нейронные сети не программируются в привычном смысле этого слова, они обучаются. Возможность обучения — одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически обучение заключается в нахождении коэффициентов связей между нейронами. В процессе обучения нейронная сеть способна выявлять сложные зависимости между входными данными и выходными, а также выполнять обобщение. Это значит, что в случае успешного обучения сеть сможет вернуть верный результат на основании данных, которые отсутствовали в обучающей выборке, а также неполных и/или «зашумленных», частично искажённых данных.

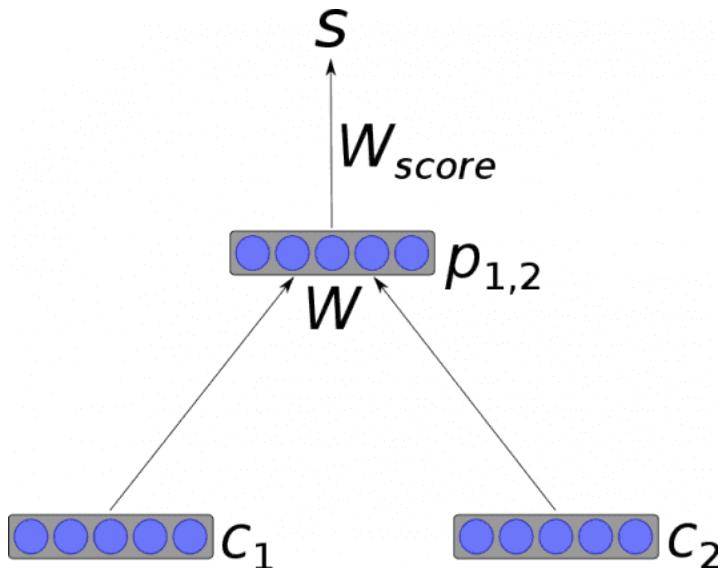
2.2.1 Типы нейросетей и задачи.

Многослойный перцептрон



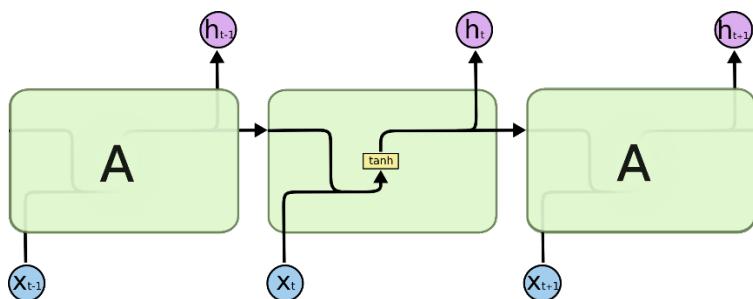
Многослойный перцептрон состоит из 3 или более слоев. Он использует нелинейную функцию активации, часто тангенциальную или логистическую, которая позволяет классифицировать линейно неразделимые данные. Каждый узел в слое соединен с каждый узлом в последующем слое, что делает сеть полностью связанной. Такая архитектура находит применение в задачах распознавания речи и машинном переводе.

Рекурсивная нейронная сеть



Рекурсивная нейронная сеть — тип глубокой нейронной сети, сформированный при применении одних и тех же наборов весов рекурсивно над структурой, чтобы сделать скалярное или структурированное предсказание над входной структурой переменного размера через активацию структуры в топологическом порядке. В простейшей архитектуре нелинейность, такая как тангенциальная функция активации, и матрица весов, разделяемая всей сетью, используются для объединения узлов в родительские объекты.

Рекуррентная нейронная сеть



Рекуррентная нейронная сеть, в отличие от прямой нейронной сети, является вариантом рекурсивной НС, в которой связи между нейронами — направленные циклы. Последнее означает, что выходная информация зависит не только от текущего входа, но также от состояний нейрона на предыдущем шаге. Такая память позволяет пользователям решать задачи NLP: распознание рукописного текста или речи.

LSTM

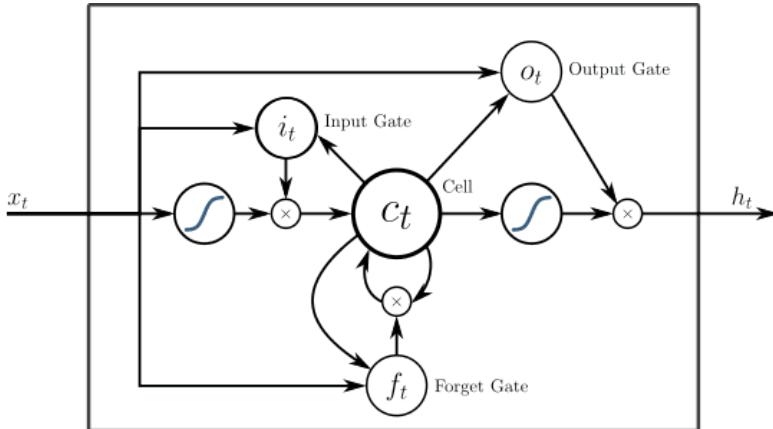


Рисунок 21. LSTM блок с входным, выходным и гейтом забывания

Сеть долгой краткосрочной памяти (LSTM) — разновидность архитектуры рекуррентной нейросети, созданная для более точного моделирования временных последовательностей и их долгосрочных зависимостей, чем традиционная рекуррентная сеть. LSTM-сеть не использует функцию активации в рекуррентных компонентах, сохраненные значения не модифицируются, а градиент не стремится исчезнуть во время тренировки. Часто LSTM применяется в блоках по несколько элементов. Эти блоки

состоят из 3 или 4 затворов (например, входного, выходного, гейта забывания), которые контролируют построение информационного потока по логистической функции. Модель часто используется для автоматической морфологической разметки. Apple, Amazon, Google, Microsoft и другие компании внедрили в продукты LSTM-сети как фундаментальный элемент.

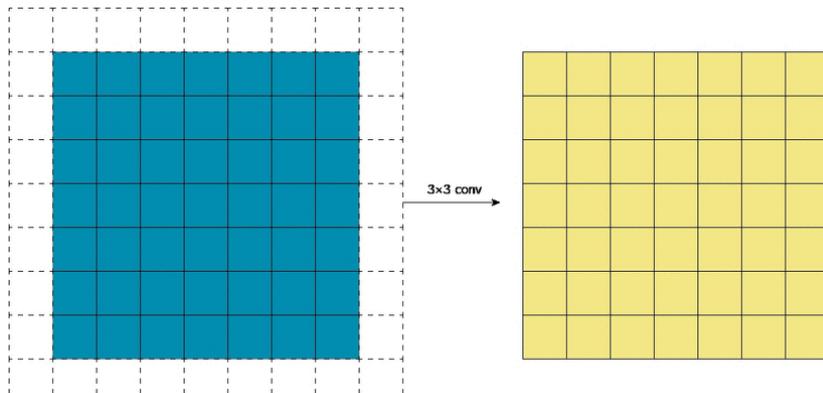
Sequence-to-sequence

Sequence-to-sequence модели состоят из двух рекуррентных сетей: кодировщика, который обрабатывает входные данные, и декодера, который осуществляет вывод. Такие модели часто используются в вопросно-ответных системах, чат-ботах и машинном переводе.

Неглубокие (shallow) нейронные сети

Неглубокие модели, как и глубокие нейронные сети, тоже популярные и полезные инструменты. Например, word2vec — группа неглубоких двухслойных моделей, которая используется для создания векторных представлений слов (word embeddings). Word2vec принимает на входе большой корпус текста и создает векторное пространство. Каждому слову в этом корпусе приписывается соответствующий вектор в этом пространстве. Отличительное свойство — слова из общих текстов в корпусе расположены близко друг к другу в векторном пространстве.

Сверточная нейронная сеть

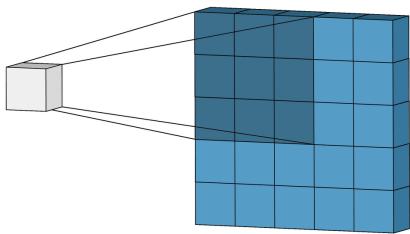


операция позволяет сети быть глубже с меньшим количеством параметров. Сверточные сети показывают выдающиеся результаты в приложениях к картинкам и речи.

Сверточная нейронная сеть (CNN) содержит один или более объединенных или соединенных сверточных слоев. CNN использует вариацию многослойного перцептрона, рассмотренного выше. Сверточные слои используют операцию свертки для входных данных и передают результат в следующий слой. Эта

2.2.2. Convolutional Neural Network.

Двумерная сверточная нейронная сеть



Двумерная свертка (2D convolution) — это довольно простая операция: начинаем с ядра, представляющего из себя матрицу весов. Ядро “скользит” над двумерным изображением, поэлементно выполняя операцию умножения с той частью входных данных, над которой оно сейчас находится, и затем суммирует все полученные значения в один выходной пиксель. Ядро повторяет эту процедуру с каждой локацией, над которой оно “скользит”, преобразуя двумерную матрицу в другую все еще двумерную матрицу признаков. Признаки на выходе являются взвешенными суммами (где веса являются

значениями самого ядра) признаков на входе, расположенных примерно в том же месте, что и выходной пиксель на входном слое.

Независимо от того, попадает ли входной признак в “примерно то же место”, он определяется в зависимости от того, находится он в зоне ядра, создающего выходные данные, или нет. Это значит, что размер ядра сверточной нейронной сети определяет количество признаков, которые будут объединены для получения нового признака на выходе.

В примере, приведенном сбоку, мы имеем $5 \times 5 = 25$ признаков на входе и $3 \times 3 = 9$ признаков на выходе. Для стандартного слоя (standard fully connected layer) мы бы имели весовую матрицу $25 \times 9 = 225$ параметров, а каждый выходной признак являлся бы взвешенной суммой всех признаков на входе. Свертка позволяет произвести такую операцию с 9-ю параметрами, ведь каждый признак на выходе получается анализом не каждого признака на входе, а только одного входного, находящегося в “примерно том же месте”.

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

Часто используемые техники

Стоит взглянуть на две техники, которые часто применяются в сверточных нейронных сетях: *Padding* и *Striding*.

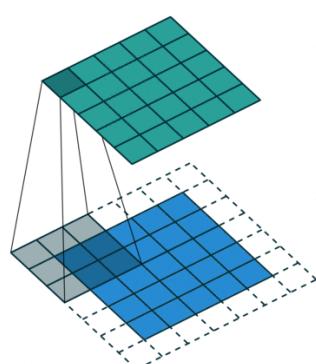


Рисунок 22. Добавление fake-пикселей по краям

Padding

В процессе скольжения края по существу обрезаются, преобразуя матрицу признаков размером 5×5 в матрицу 3×3 . Крайние пиксели никогда не оказываются в центре ядра, потому что тогда ядру не над чем будет скользить за краем. Это совсем не идеальный вариант, так как мы хотим, чтобы размер на выходе равнялся входному.

Padding добавляет к краям поддельные (fake) пиксели (обычно нулевого значения, вследствие этого к ним применяется термин “нулевое дополнение” — “zero padding”). Таким образом, ядро при проскальзывании позволяет неподдельным пикселям оказываться в своем центре, а затем распространяется на поддельные пиксели за пределами края, создавая выходную матрицу того же размера, что и входная.

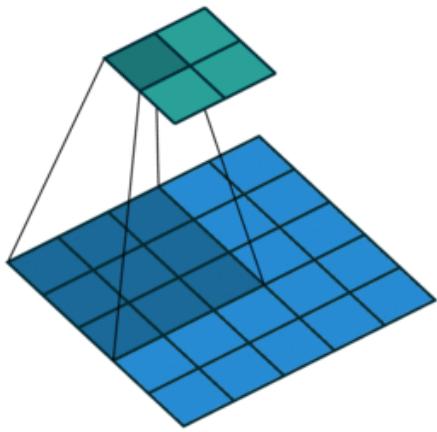


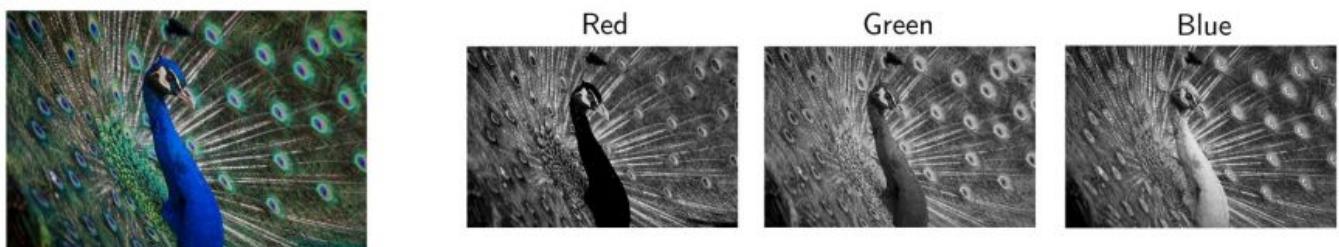
Рисунок 23. Свертка с шагом 2

Пролет является стандартной сверткой. Шаг 2 означает, что пролеты совершаются через каждые два пикселя, пропуская все другие пролеты в процессе и уменьшая их количество примерно в 2 раза, шаг 3 означает пропуск 3-х пикселей, сокращая количество в 3 раза и т.д.

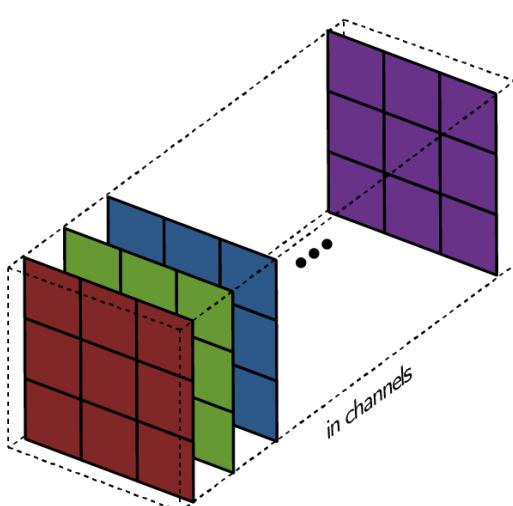
Более современные сети, такие как архитектуры ResNet, полностью отказываются от субдискритизирующих слоев во внутренних слоях в пользу чередующихся сверток, когда необходимо уменьшить размер на выходе.

Многоканальная версия сверточной нейронной сети

Приведенные выше диаграммы касаются только случая, когда изображение имеет один входной канал. На практике большинство входных изображений имеют 3 канала, и чем глубже вы в сети, тем больше это число.



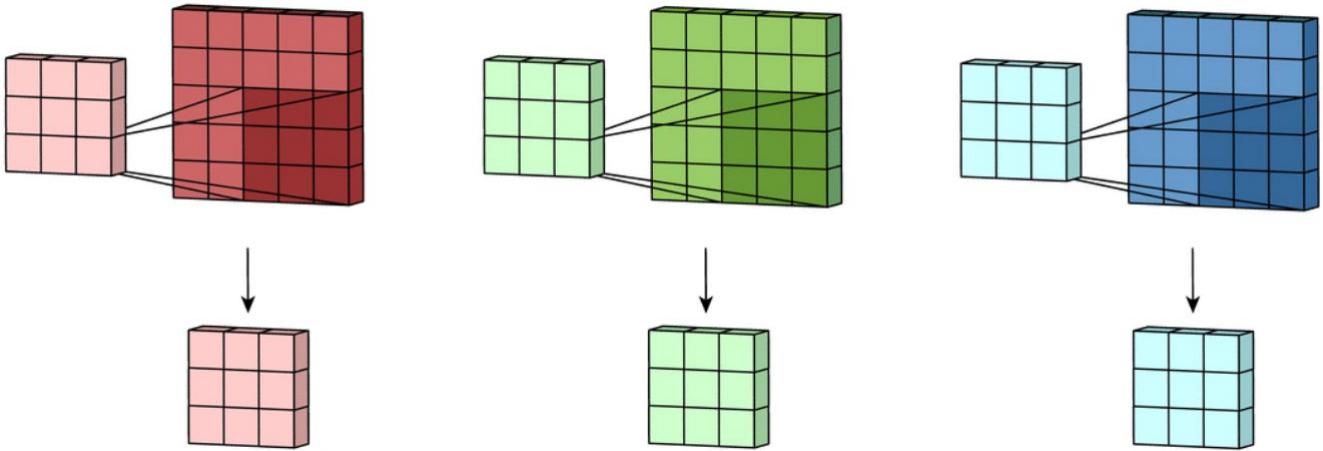
В большинстве случаев мы имеем дело с изображениями RGB с тремя каналами.



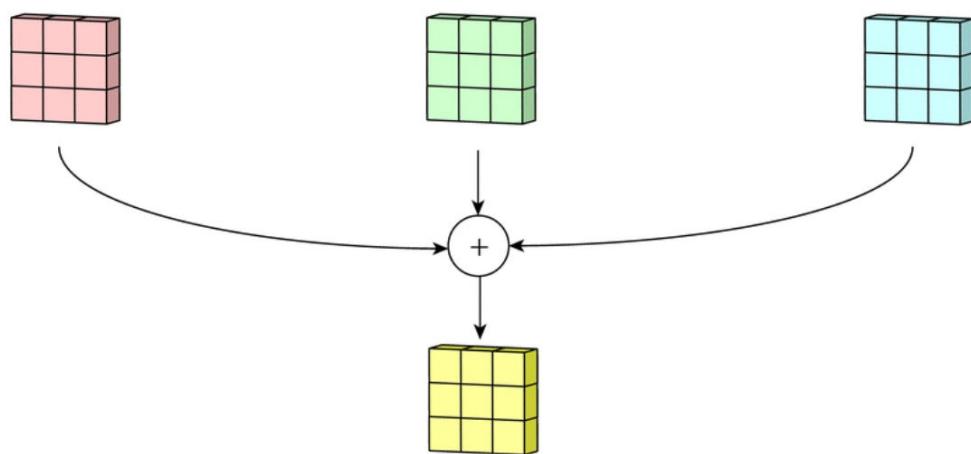
Каждый фильтр на самом деле представляет собой коллекцию ядер, причем для каждого отдельного входного канала этого слоя есть одно ядро, и каждое ядро уникально. Каждый фильтр в сверточном слое создает только один выходной канал и делают они это так: каждое из ядер фильтра «скользит» по их соответствующим входным каналам, создавая обработанную версию каждого из них. Некоторые ядра могут иметь больший вес, чем другие, для того чтобы уделять больше внимания определенным входным каналам (например, фильтр может задать красному каналу ядра больший вес, чем другим каналам, и, следовательно, больше реагировать на различия в образах из красного канала).

Striding

Striding. Часто бывает, что при работе со сверточным слоем, нужно получить выходные данные меньшего размера, чем входные. Это обычно необходимо в сверточных нейронных сетях, где размер пространственных размеров уменьшается при увеличении количества каналов. Один из способов достижения этого — использование субдискритизирующих слоев (pooling layer), например, принимать среднее/максимальное значение каждой ветки размером 2×2 , чтобы уменьшить все пространственные размеры в два раза. Еще один способ добиться этого — использовать stride (шаг). Идея stride заключается в том, чтобы пропустить некоторые области, над которыми скользит ядро. Шаг 1 означает, что берутся пролеты через пиксель, то есть по факту каждый

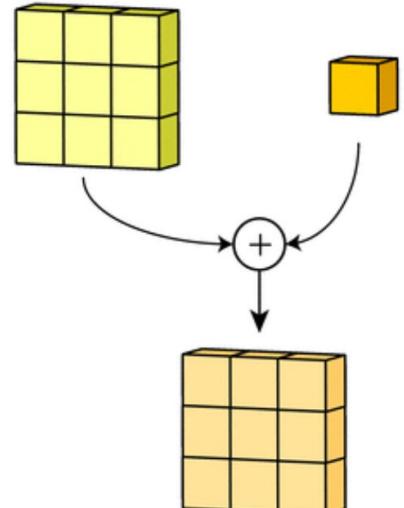


Затем каждая из обработанных в канале версий суммируется вместе для формирования одного канала. Ядра каждого фильтра генерируют одну версию каждого канала, а фильтр в целом создает один общий выходной канал:



Наконец, каждый выходной файл имеет свое смещение. Смещение добавляется к выходному каналу для создания конечного выходного канала.

Результат для любого количества фильтров идентичен: каждый фильтр обрабатывает вход со своим отличающимся от других набором ядер и скалярным смещением по описанному выше процессу, создавая один выходной канал. Затем они объединяются вместе для получения общего выхода, причем количество выходных каналов равно числу фильтров. При этом обычно применяется нелинейность перед передачей входа другому слою свертки, который затем повторяет этот процесс.



Параметры в сверточной нейронной сети

Рисунок 24. Смещение

Свертка — это по-прежнему линейное преобразование.

Даже с уже описанной механикой работы сверточного слоя, все еще сложно связать это с нейронной сетью прямого распространения (feed-forward network), и это все еще не объясняет, почему свертки масштабируются и работают намного лучше с изображениями.

Предположим, что у нас есть вход 4×4 , и мы хотим преобразовать его в сетку 2×2 . Если бы мы использовали feed-forward network, мы бы переделали вход 4×4 в вектор длиной 16 и передали его

через полносвязный слой с 16 входами и 4 выходами. Можно было бы визуализировать весовую матрицу W для слоя по типу:

$$\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} & w_{1,6} & w_{1,7} & w_{1,8} & w_{1,9} & w_{1,10} & w_{1,11} & w_{1,12} & w_{1,13} & w_{1,14} & w_{1,15} & w_{1,16} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} & w_{2,6} & w_{2,7} & w_{2,8} & w_{2,9} & w_{2,10} & w_{2,11} & w_{2,12} & w_{2,13} & w_{2,14} & w_{2,15} & w_{2,16} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} & w_{3,6} & w_{3,7} & w_{3,8} & w_{3,9} & w_{3,10} & w_{3,11} & w_{3,12} & w_{3,13} & w_{3,14} & w_{3,15} & w_{3,16} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} & w_{4,6} & w_{4,7} & w_{4,8} & w_{4,9} & w_{4,10} & w_{4,11} & w_{4,12} & w_{4,13} & w_{4,14} & w_{4,15} & w_{4,16} \end{bmatrix}$$

И хотя сверточные операции с ядрами могут вначале показаться немного странными, это по-прежнему линейные преобразования с эквивалентной матрицей перехода.

Если мы использовали ядро K размера 3 на видоизмененным входом размера $4*4$, чтобы получить выход $2*2$, эквивалентная матрица перехода будет выглядеть так:

$$\begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} & 0 \\ 0 & 0 & 0 & 0 & 0 & k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix}$$

Примечание: в то время как приведенная матрица является эквивалентной матрицей перехода, фактическая операция обычно реализуется как совсем иное [матричное умножение](#)

Свертка, в целом, все еще является линейным преобразованием, но в то же время она также представляет собой совершенно иной вид преобразования. Для матрицы с 64 элементами существует всего 9 параметров, которые повторно используются несколько раз. Каждый выходной узел получает только определенное количество входов (те, что находятся внутри ядра). Нет никакого взаимодействия с другими входами, так как вес для них равен 0.

Полезно представлять сверточные операции как hard prior для весовых матриц – предопределенные параметры сети (использование предобученной модели). Например, когда вы используете предварительно обработанную модель для классификации изображений, вы используете предварительные параметры сети как prior как экстрактор образов для вашего окончательного полносвязного слоя.

Вам не нужно оптимизировать все 64 параметра, потому большинство из них установлено на ноль (и они останутся такими), а остальные мы преобразуем в общие параметры и в результате получим только 9 параметров для оптимизации. Эта эффективность имеет значение, потому что, когда вы переходите от 784 входов MNIST к реальным изображениям $224*224*3$, это более 150 000 входов. Слой, пытающийся вдвое уменьшить вход до 75 000 входных значений, по-прежнему потребует более 10 миллиардов параметров. Для сравнения ResNet-50 имеет около 25 миллионов параметров. Таким образом, фиксирование некоторых параметров равными нулю и их связывание повышает эффективность, но в отличие от случая с предобученными моделями, где мы знаем, что prior работает грамотно, потому что он хорошо работает с большим общим набором изображений, откуда мы знаем, что это будет работать хоть сколько-то хорошо в нашем случае?

Ответ заключается в комбинациях образов, изучаемых параметрами за счет prior.

Локальные особенности

- Ядра объединяют пиксели только из небольшой локальной области для формирования выхода. То есть выходные признаки видят только входные признаки из небольшой локальной области;
- Ядро применяется глобально по всему изображению для создания матрицы выходных значений.

Таким образом, с backpropagation (метод обратного распространения ошибки), идущим во всех направлениях от узлов классификации сети, ядра имеют интересную задачу изучения весов для создания признаков только из локального набора входов. Кроме того, поскольку само ядро

применяется по всему изображению, признаки, которые изучает ядро, должны быть достаточно общими, чтобы поступать из любой части изображения.

Пиксели, однако, всегда отображаются в последовательном порядке, а соседние пиксели влияют на пиксель рядом, например, если все соседние пиксели красные, довольно вероятно, что пиксель рядом также красный. Если есть отклонения, это интересная аномалия, которая может быть преобразована в признак, и все это можно обнаружить при сравнении пикселя со своими соседями, с другими пикселями в своей местности.

Эта идея — то, на чем были основаны более ранние методы извлечения признаков компьютерным зрением. Например, для обнаружения граней можно использовать фильтр обнаружения граней Sobel — ядро с фиксированными параметрами, действующее точно так же, как стандартная одноканальная свертка:

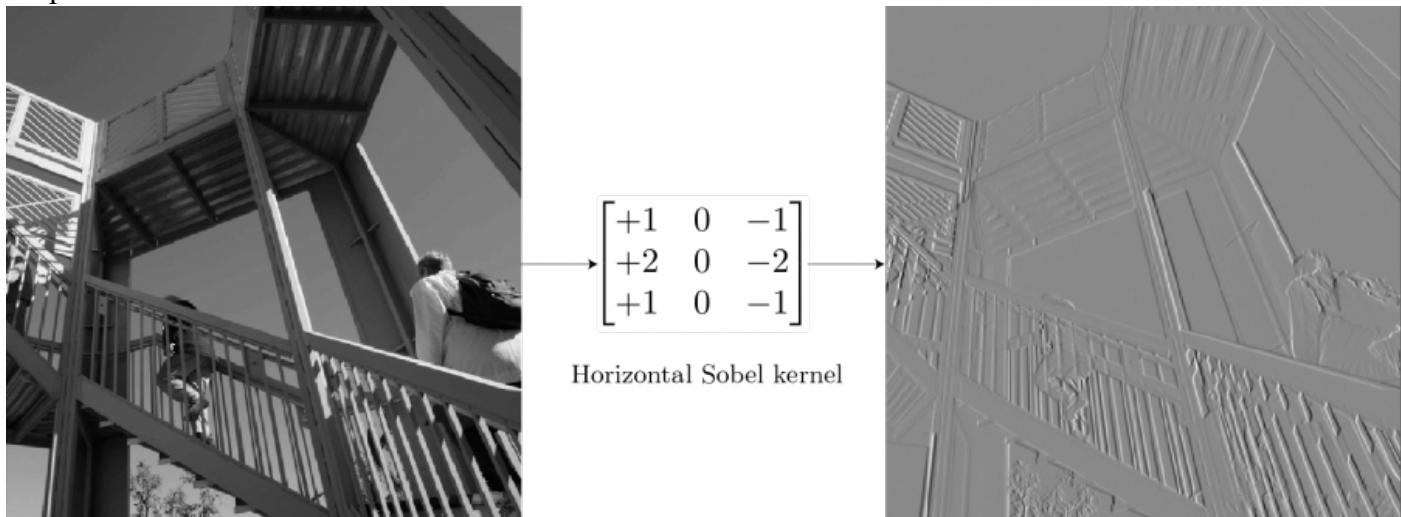


Рисунок 25. Применение ядра, детектирующего грани

Для сетки, не содержащей граней (например, неба на заднем фоне), большинство пикселей имеют одинаковое значение, поэтому общий вывод ядра в этой точке равен 0. Для сетки с вертикальными гранями существует разница между пикселями слева и справа от грани, и ядро вычисляет эту ненулевую разницу, находя ребра. Ядро за раз работает только с сетками 3*3, обнаруживая аномалии в определенных местах, но применения ко всему изображению достаточно для обнаружения определенного признака в любом месте изображения!

Но могут ли полезные ядра быть изучены? Для ранних слоев, работающих с необработанными пикселями, мы могли бы ожидать детекторы признаков низкого уровня, таких как ребра, линии и т.д. Один из самых мощных инструментов для интерпретируемости моделей — визуализация признаков с помощью оптимизации. Идея в корне проста: оптимизируйте изображение (обычно инициализированное случайным шумом), чтобы активировать фильтр как можно сильнее. Такой способ интуитивно понятен: если оптимизированное изображение полностью заполнено гранями, то это убедительное доказательство того, что фильтр активирован и занят поиском. Используя это, мы можем заглянуть в изученные фильтры, и результаты будут ошеломляющими:

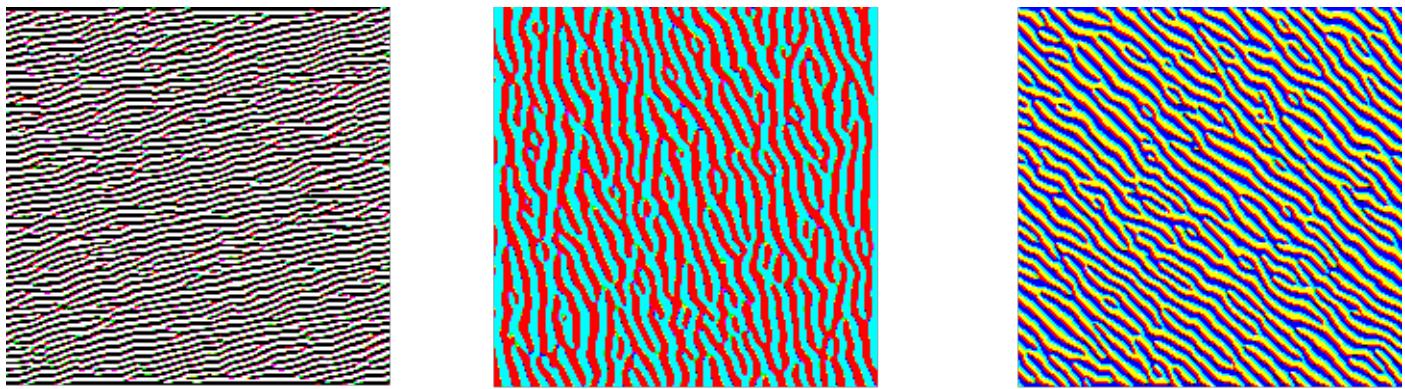


Рисунок 26. Визуализация признаков для 3 каналов после первого сверточного слоя

Стоит обратить внимание, что, хотя они обнаруживают разные типы ребер, они все еще являются низкоуровневыми детекторами

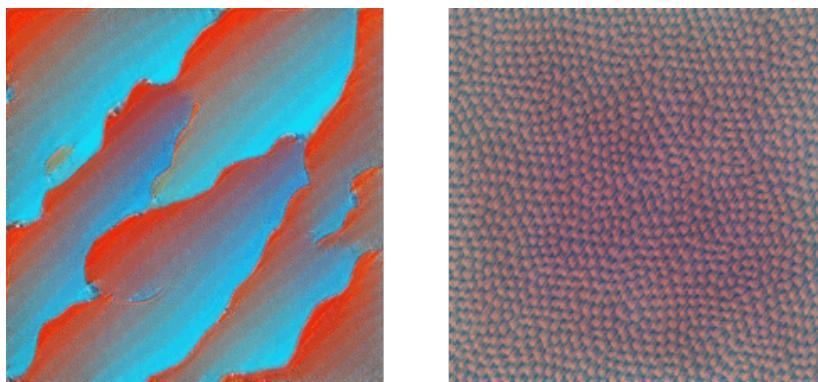


Рисунок 27. После 2-й и 3-й свертки

Важно обратить внимание на то, что конвертированные изображения остаются изображениями. Выход, получаемый от небольшой сетки пикселей в верхнем левом углу, будет тоже расположен в верхнем левом углу. Таким образом, можно применять один слой поверх другого (как два слева на картинке) для извлечения более глубоких признаков, которые мы визуализируем.

Тем не менее, как бы глубоки ни заходили наши детекторы признаков, без каких-либо дальнейших изменений они все равно будут работать на очень маленьких участках изображения. Независимо от того, насколько глубоки ваши детекторы, вы не сможете обнаружить лица в сетке 3×3 . И вот здесь возникает идея рецептивного поля (receptive field).

Рецептивные поля

Существенной особенностью архитектур сверточной нейронной сети является то, что размеры ввода становятся все меньше и меньше от начала до конца сети, а количество каналов становится больше. Это, как упоминалось ранее, часто делается с помощью strides или pooling layers. Местность определяет, какие входные данные из предыдущего уровня будут на выходе следующего. Receptive field определяет, какую область исходного входа получает выход.

Идея strided convolution состоит в том, что мы обрабатываем пролеты только на фиксированном расстоянии друг от друга и пропускаем те, что посередине. С другой точки зрения, мы оставляем только выходы на определенном расстоянии друг от друга, удаляя остальные.

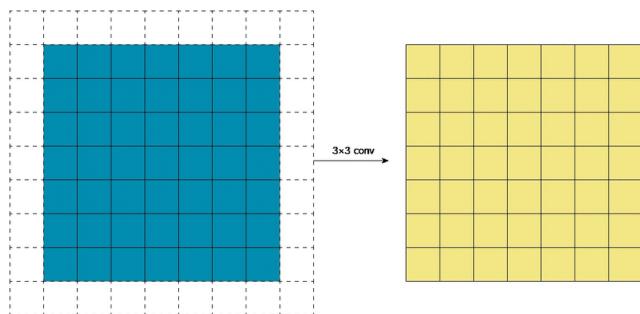
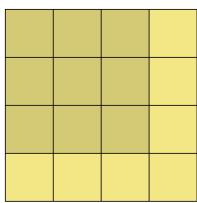
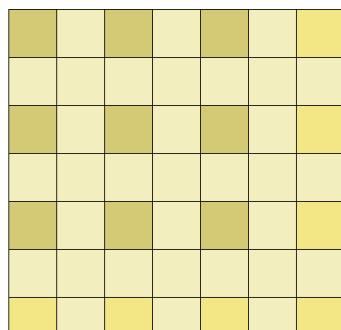


Рисунок 28. Свртка 3×3 , шаг 2

Затем мы применяем нелинейность к выходным данным, затем накладываем еще один новый слой свертки сверху. Здесь все становится интересным. Даже если бы мы применили ядро того же размера (3×3), имеющее одну и ту же локальную область, к выходу strided convolution, ядро имело бы более эффективное receptive field.



Operation on the true strided output

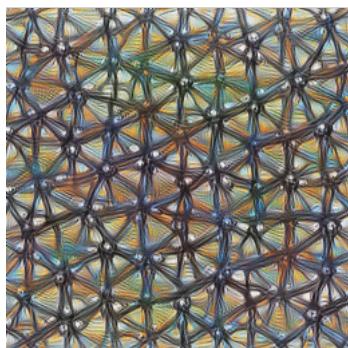


Operation on the output with the removed pixels

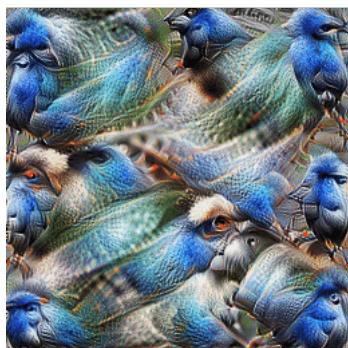
Это связано с тем, что выход strided слоя по-прежнему представляет из себя одно и то же изображение. Это не столько обрезка, сколько изменение размера, только теперь каждый отдельный пиксель на выходе является «представителем» большей площади (другие пиксели которой были отброшены) из того же местоположения исходного ввода. Поэтому, когда ядро следующего слоя применяется к выходу, оно работает с пикселями, собранными из большей области.



mixed3a, channel 31



mixed4a, channel 11



mixed5a, channel 14

Рисунок 29. Визуализация усложнения после добавления слоев

Обнаруживая низкоуровневые признаки и используя их для обнаружения признаков более высокого уровня по мере улучшения своей визуальной иерархии, она в конечном итоге может обнаруживать целые визуальные концепции, такие как лица, птицы, деревья и т.д., именно это делает их такими мощными и эффективными для изображений.

2.2.3. Обзор популярных моделей и выбор лучшей.

При разработке системы распознавания мы экспериментировали с несколькими архитектурами:

- [Wide ResNet](#)
- [Inception-ResNet](#)
- [Light CNN](#)

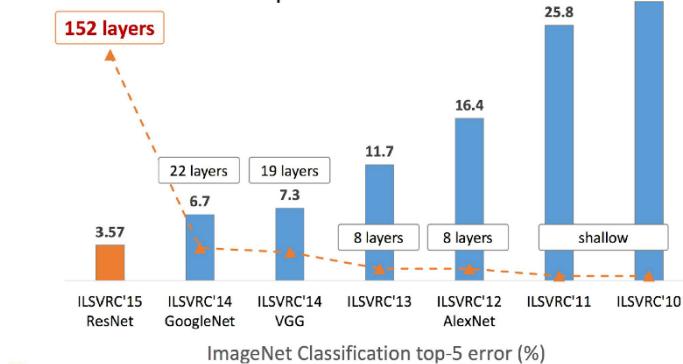
Немного лучше остальных себя показала Inception-ResNet. Далее рассмотрим более подробно архитектуру каждой сети и ошибки на разных датасетах.

Models	lfw accuracy	Training dataset	Complexity
Inception ResNet v1	0.9905	CASIA-WebFace	★★★
Inception ResNet v1	0.9965	VGGFace2	★★★★★
Light CNN	0.9933	CASIA-WebFace	★★★★★
Wide ResNet	0.9856	VGGFace2	★

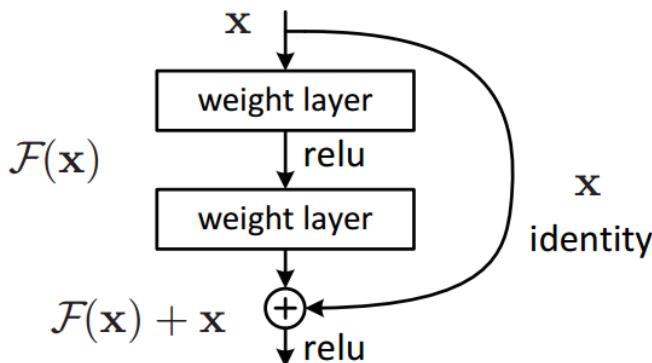
*для сравнения, стоит заметить, что датасет VGGFace2 содержит 3,3М лиц, что на порядок больше датасета CASIA-WebFace, содержащего всего около полумиллиона лиц.

2.2.3.1. Wide ResNet.

Revolution of Depth



Residual Networks, сокращённо ResNet (можно перевести как «остаточные сети»), появились в конце 2015 года и победили в ImageNet 2015. Их изобретатели — команда из азиатского подразделения Microsoft Research. Они смогли построить и успешно обучать сети очень большой глубины, т. е. с большим количеством слоёв. Сравнение победителей разных лет по этому параметру.

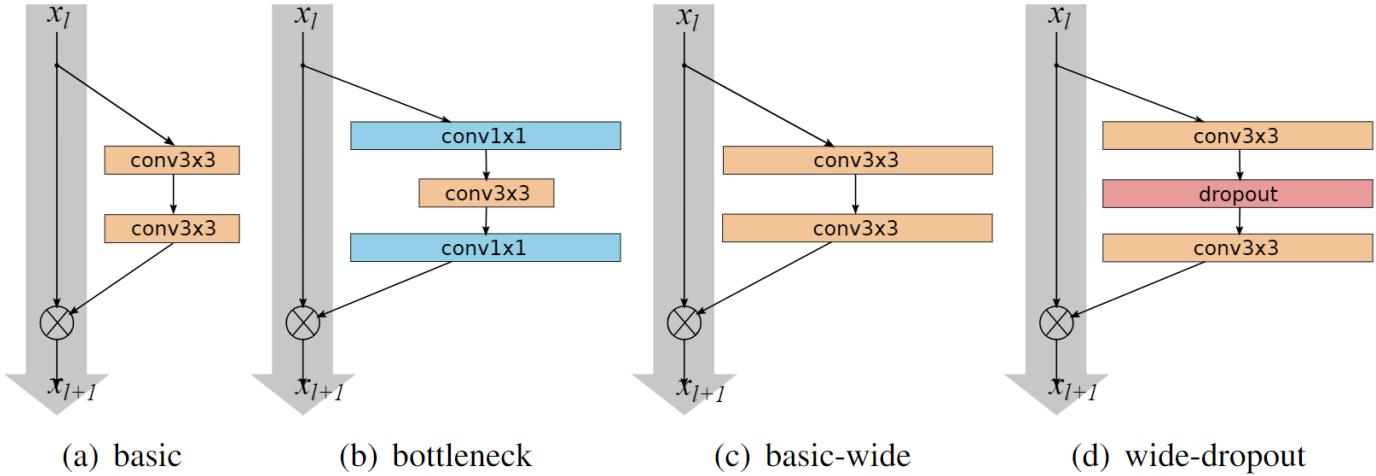


Основной элемент ResNet — Residual-блок (остаточный блок) с shortcut-соединением, через которое данные проходят без изменений. Res-блок представляет собой несколько свёрточных слоёв с активациями, которые преобразуют входной сигнал x в $\mathcal{F}(x)$. Shortcut-соединение — это тождественное преобразование $x \rightarrow x$.

В результате такой конструкции Res-блок учит, как входной сигнал x отличается от $\mathcal{F}(x)$. Поэтому если на некотором слое сеть уже достаточно хорошо аппроксимировала исходную функцию, порождающую данные, то на дальнейших слоях оптимизатор может в Res-блоках делать веса близкими к нулю, и сигнал будет почти без изменений проходить по shortcut-соединению. В некотором смысле можно сказать, что CNN сама определяет свою глубину.

Как показала практика, последние Res-блоки вносят малый вклад в формирование конечного результата работы всей сети, поэтому простое увеличение количества блоков не даёт ожидаемого результата. Из этих соображений возникла идея увеличивать не глубину, а ширину Residual-блока, т. е. количество фильтров в свёртках.

Появившаяся в 2016 году Wide Residual Network делает именно это.
Ниже представлены разные популярные модернизации Res-блока.

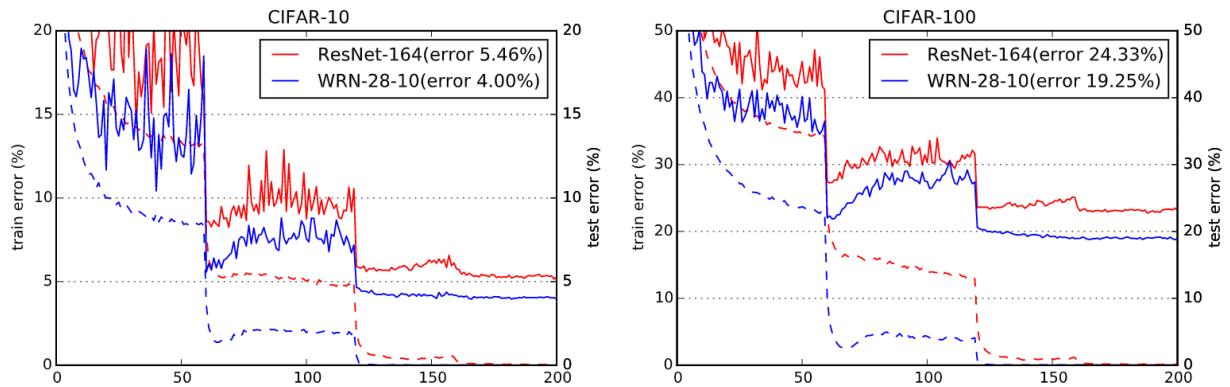


Чтобы увеличить эффективность вычислений и уменьшить количество параметров для больших сетей, таких как ResNet-50/101/152, авторы применили подход, который появился в модели [Network In Network](#), а затем был внедрен в [Inception](#)-модель. Идея заключается в использовании 1×1 свёртки для уменьшения числа каналов перед более дорогой операцией 3×3 свёртки, а затем в восстановлении исходного числа каналов посредством ещё одной 1×1 свёртки. Res-блок с таким трюком называется bottleneck.

Ниже представлены тесты и сравнения разных архитектур. Параметр k отвечает за количество фильтров. В обычном ResNet используется параметр $k = 1$. В Wide ResNet параметр $k > 1$.

	depth- k	# params	CIFAR-10	CIFAR-100
NIN			8.81	35.67
DSN			8.22	34.57
FitNet			8.39	35.04
Highway			7.72	32.39
ELU			6.55	24.28
original-ResNet	110	1.7M	6.43	25.16
	1202	10.2M	7.93	27.82
stoc-depth	110	1.7M	5.23	24.58
	1202	10.2M	4.91	-
pre-act-ResNet	110	1.7M	6.37	-
	164	1.7M	5.46	24.33
	1001	10.2M	4.92(4.64)	22.71
WRN (ours)	40-4	8.9M	4.53	21.18
	16-8	11.0M	4.27	20.43
	28-10	36.5M	4.00	19.25

Рисунок 30. Эффект добавления bottleneck в Wide ResNet



Далее был использован wide-dropout блок. Он добавляет эффект регуляризации.

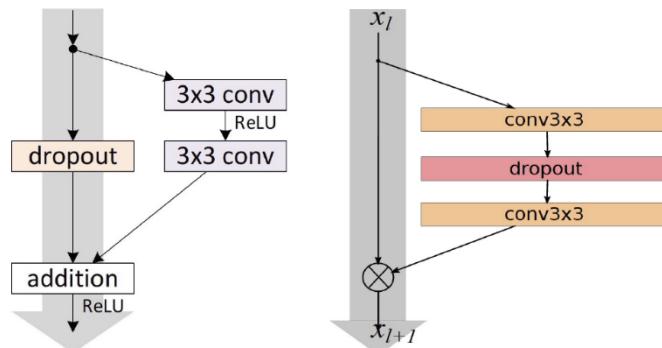
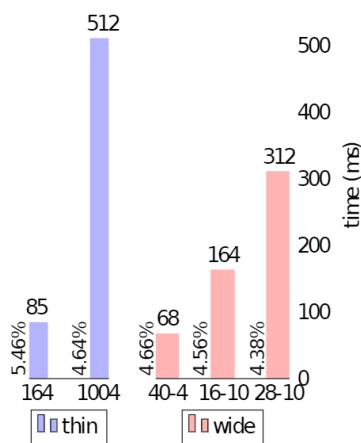
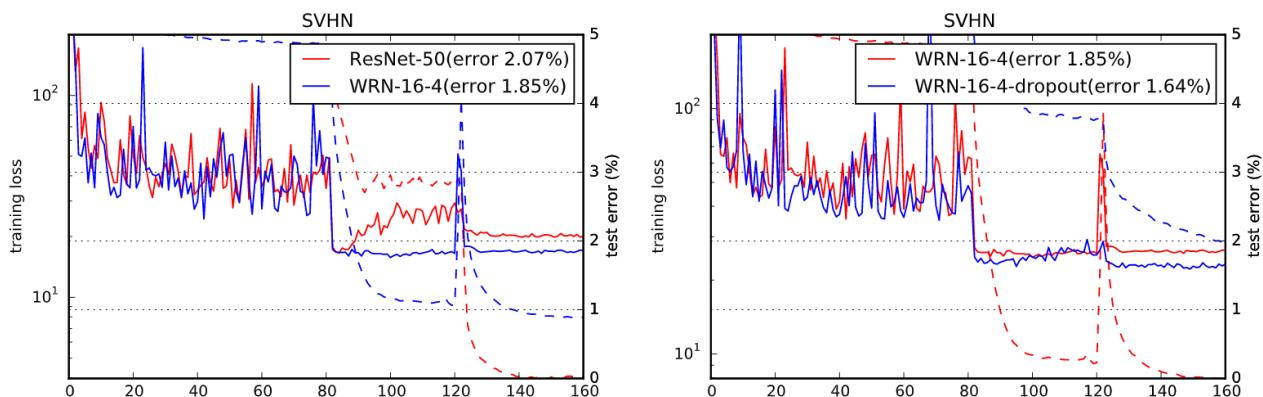


Рисунок 31. ResNet и Wide ResNet

Ниже представлены тесты и сравнения разных архитектур.

depth	k	dropout	CIFAR-10	CIFAR-100	SVHN
16	4		5.02	24.03	1.85
16	4	✓	5.24	23.91	1.64
28	10		4.00	19.25	-
28	10	✓	3.89	18.85	-
52	1		6.43	29.89	2.08
52	1	✓	6.28	29.78	1.70

Рисунок 32. Эффект добавления dropout в Wide ResNet



Вывод:

Исходя из научных исследований, широкие сети всего с 16 слоями могут значительно превосходить 1000-слойные глубокие сети на CIFAR, а 50-слойные превосходят 152-слойные на ImageNet. Из этого следует, что основная мощность остаточных сетей находится в остаточных блоках, а не в экстремальной глубине, как утверждалось ранее. Кроме того, широкие остаточные сети в несколько раз быстрее обучаются.

Полная архитектура сети:

<https://github.com/szagoruyko/wide-residual-networks>

2.2.3.2. Inception-ResNet.

Особенность Inception блока заключается в “разветвлении” на несколько разных блоков. Вот пример Inception блока.

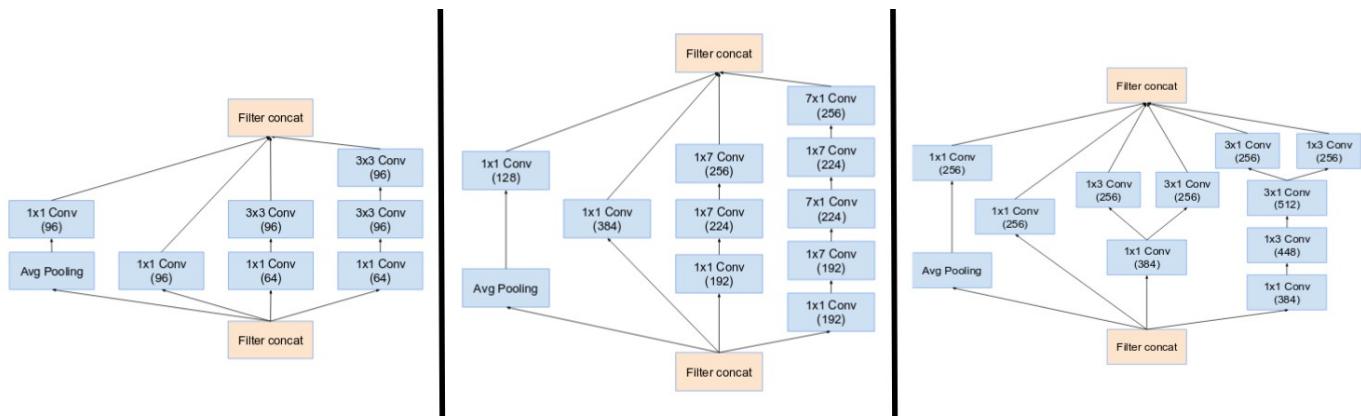


Рисунок 33. Inception блок

Кроме этого, в сети данного типа присутствуют специальные “Reduction blocks”, которые используются для изменения ширины и высоты сетки. Данное новшество можно наблюдать только в сетях последних версий.

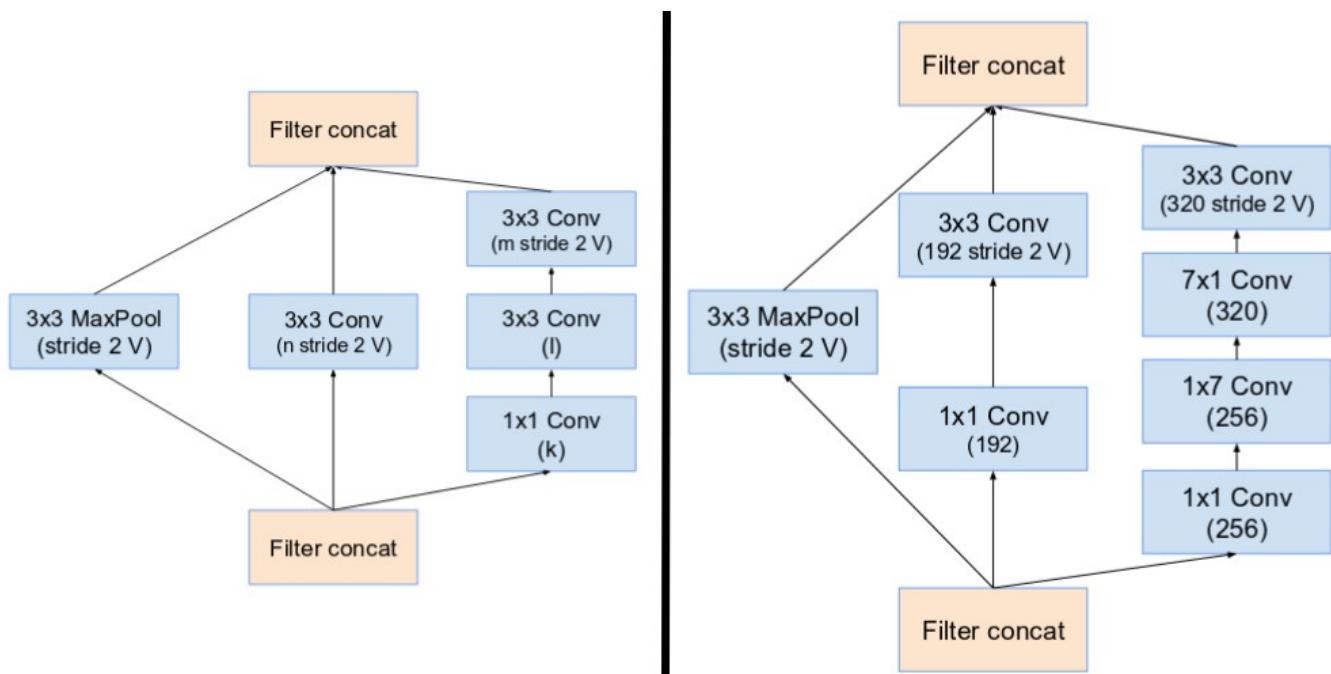


Рисунок 34. Reduction blocks

Уникальность Inception-ResNet заключается в совмещении технологий блоков типа Inception и Wide ResNet, про которые мы писали выше.

Основные особенности:

- Inception-ResNet блоки основаны на методе использования bottleneck-block и wide-block. Помимо этого, внедрены методы 1x7 и 7x1 последовательной свертки, а также 1x3 и 3x1 последовательной свертки из Inception V4.
- Операция объединения внутри основных начальных модулей была заменена в пользу остаточных соединений. Однако эти операции все ещё остались в reduction блоках.
- “Reduction” блок такой же, как и у Inception V4.

- Сети с остаточными глубинными блоками приводили к тому, что сеть "умирала", если число фильтров превышало 1000. Таким образом, для повышения стабильности авторы масштабировали остаточные активации на величину от 0,1 до 0,3.
- Модели типа Inception-ResNet способны достичь лучшей точности за меньшее количество эпох.

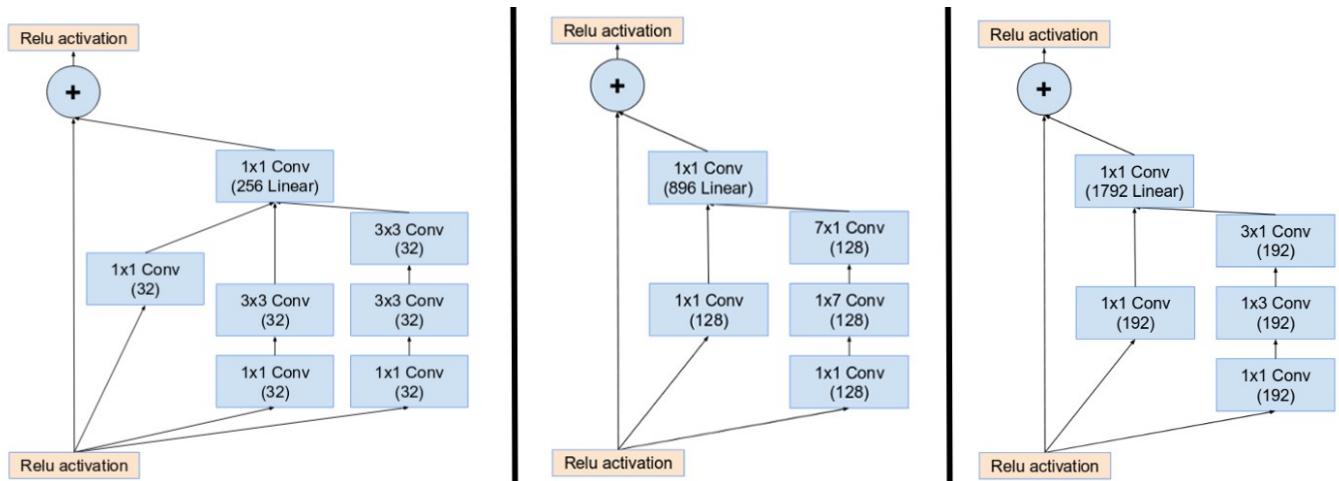
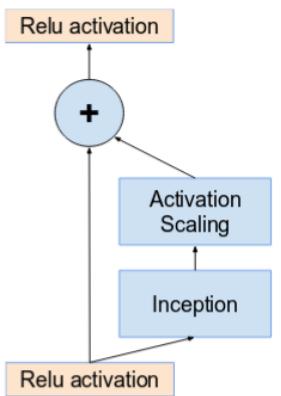


Рисунок 36. Блоки типа A,B,C в Inception-ResNet

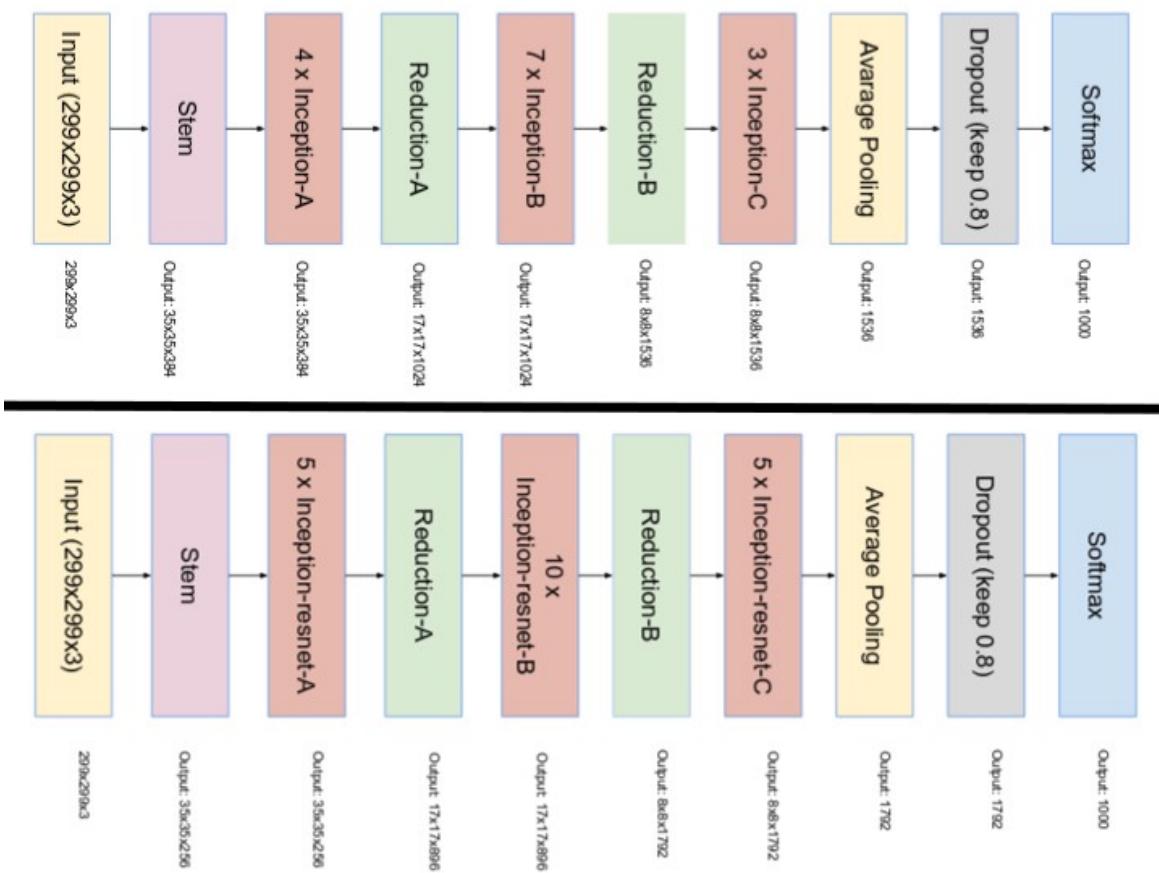
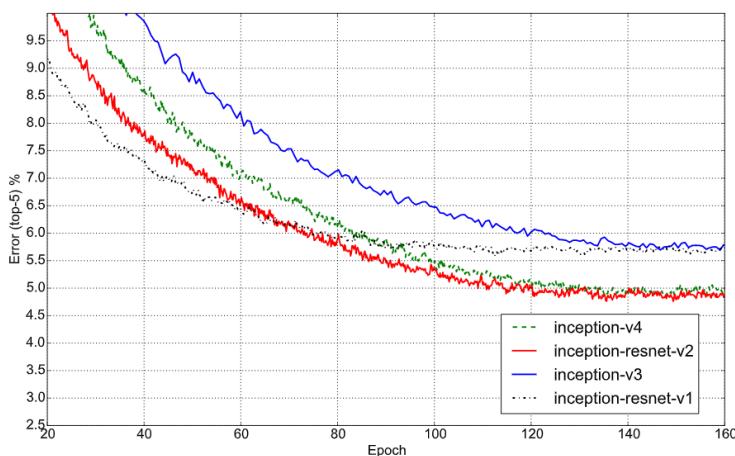


Рисунок 35. Архитектура Inception V4 и Inception-ResNet

Как видим, структура сети никак не изменилась; изменились только блоки. Inception-ResNet V1 и V2 отличаются только количеством фильтров в каждом блоке: в первой версии в каждом блоке количество фильтров не увеличивается при последующей свертке, в то время как во второй версии в каждом блоке количество фильтров растет с шагом 32 при последующей свертке.

Ниже представлены тесты и сравнение разных архитектур на датасете ILSVRC 2012.

Network	Top-1 Error	Top-5 Error
BN-Inception	25.2%	7.8%
Inception-v3	21.2%	5.6%
Inception-ResNet-v1	21.3%	5.5%
Inception-v4	20.0%	5.0%
Inception-ResNet-v2	19.9%	4.9%



Вывод:

Inception-ResNet легче своих предшественников, а точность данного типа модели увеличивается за счет увеличения количества фильтров в каждом блоке. Одна из ключевых особенностей данной сети является совмещение в себе мощных блоков Inception и эффективных Res-блоков.

2.2.3.3. Light CNN.

Одна из ключевых особенностей данного типа моделей – легкость и хорошая точность относительно своих конкурентов.

MFM-слой

Датасеты лица часто содержат зашумленные сигналы. Чтобы CNN давала качественные предсказания, нужно должным образом обработать данные сигналы. Активация линейного блока (ReLU) разделяет шумовые и информативные сигналы порогом (или смещением) активации одного нейрона. Обычно, данный порог является 0 (x , если $x > 0$ и 0, если $x < 0$). Однако, этот порог может привести к потере некоторой информации, особенно для первых нескольких слоев свертки, поскольку эти слои похожи на фильтры Габора (про них описывалось выше). Для решения этой проблемы были предложены новые блоки – LReLU, PReLU и ELU.

Предложим новую функцию активации, которая в одном слое свертки будет иметь следующие интересные характеристики:

1) поскольку крупномасштабный набор данных часто содержит различные шумы разных типов, мы ожидаем, что шумовые сигналы и информационные сигналы можно разделить, при этом не потерять не ту, не другую информацию;

- 2) при наличии горизонтального края или линии на изображении возбуждается нейрон, соответствующий горизонтальной информации, а нейрон, соответствующий вертикальной информации, тормозится; аналогично наоборот;
 3) торможение обучения одного нейрона не зависит от параметров, а значит в значительной степени не зависит от обучающей выборки.

Для достижения вышеуказанных характеристик была предложена операция Max-Feature-Map (MFM), которая основана на активации Maxout. Однако, задачи активации MFM и Maxout немного отличаются.

- Maxout стремится аппроксимировать произвольную выпуклую функцию с помощью достаточного количества скрытых нейронов. Чем больше нейронов используется, тем лучше получаются результаты аппроксимации. Как правило, масштаб сети Maxout больше, чем у сети ReLU.
- MFM прибегает к функции \max для подавления активации небольшого числа нейронов, так что модели CNN на основе MFM являются легкими и надежными.

Хотя MFM и Maxout all используют функцию \max для активации нейронов, MFM нельзя рассматривать как аппроксимацию выпуклой функции. Существует два типа MFM операций: MFM 2/1 и MFM 3/2.

Ниже показано математическое представление MFM-активаций:

$$\hat{x}_{ij}^k = \max(x_{ij}^k, x_{ij}^{k+N}, x_{ij}^{k+2N}) \quad \left\{ \begin{array}{l} \hat{x}_{ij}^{k_1} = \max(x_{ij}^k, x_{ij}^{k+N}, x_{ij}^{k+2N}) \\ \hat{x}_{ij}^{k_2} = \text{median}(x_{ij}^k, x_{ij}^{k+N}, x_{ij}^{k+2N}) \end{array} \right.$$

Рисунок 38. MFM 2/1

Рисунок 37. MFM 3/2

Ниже наглядно представлены ReLU, Maxout, MFM 2/1 и MFM 3/2 активации:

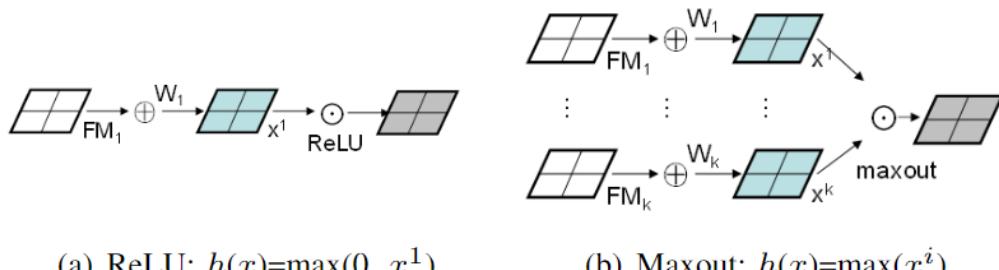
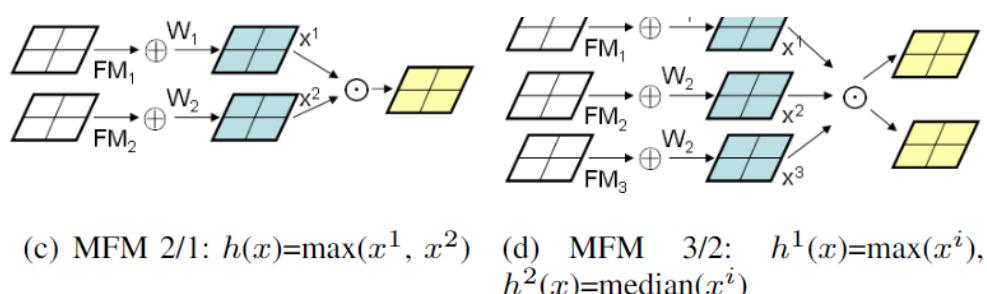


Рисунок 39. Сравнение точности на датасете LFW с использованием разных функций активаций



Method	Accuracy	Rank-1	DIR@FAR=1%
ReLU [26]	98.30	88.58	67.56
PReLU [28]	98.17	88.30	66.30
ELU [29]	97.70	84.70	62.09
MFM 2/1	98.80	93.80	84.40
MFM 3/2	98.83	94.97	88.59

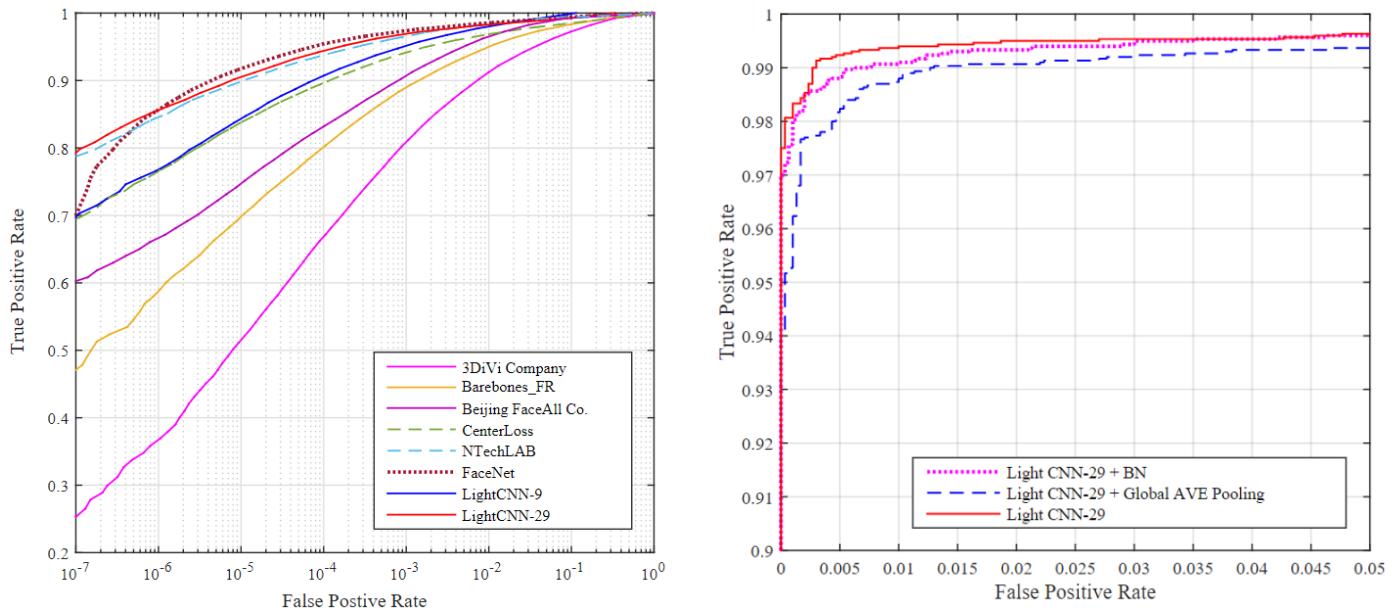
Ниже представлены тесты и сравнение разных архитектур на датасете LFW. Обучались модели на датасете CASIA-WebFace и MS-Celeb-1M.

Method	#Net	Acc on LFW	VR@FAR=0	Protocol	Rank-1	DIR@FAR=1%	Acc on YTF
DeepFace [5]	7	97.35	46.33	unrestricted	64.90	44.50	91.40
Web-Scale [14]	4	98.37	-	unrestricted	82.50	61.90	-
DeepID2+ [16]	25	99.47	69.36	unrestricted	95.00	80.70	93.20
WebFace [6]	1	97.73	-	unrestricted	-	-	90.60
FaceNet [8]	1	99.63	-	unrestricted	-	-	95.10
SeetaFace [36]	1	98.62	-	unrestricted	92.79	68.13	-
VGG [7]	1	97.27	52.40	unsupervised	74.10	52.01	92.80
CenterLoss [37]	1	98.70	61.40	unsupervised	94.05	69.97	94.90
Light CNN-4	1	97.97	79.20	unsupervised	88.79	68.03	90.72
Light CNN-9	1	98.80	94.97	unsupervised	93.80	84.40	93.40
Light CNN-29	1	99.33	97.50	unsupervised	97.33	93.62	95.54

Ниже представлена архитектура лучшей модели Light CNN-29:

Type	Filter Size /Stride, Pad	Output Size	#Params
Conv1	$5 \times 5/1, 2$	$128 \times 128 \times 96$	2.4K
MFM1	-	$128 \times 128 \times 48$	-
Pool1	$2 \times 2/2$	$64 \times 64 \times 48$	-
Conv2_x	$\begin{bmatrix} 3 \times 3/1, 1 \\ 3 \times 3/1, 1 \end{bmatrix} \times 1$	$64 \times 64 \times 48$	82K
Conv2a	$1 \times 1/1$	$64 \times 64 \times 96$	4.6K
MFM2a	-	$64 \times 64 \times 48$	-
Conv2	$3 \times 3/1, 1$	$64 \times 64 \times 192$	165K
MFM2	-	$64 \times 64 \times 96$	-
Pool2	$2 \times 2/2$	$32 \times 32 \times 96$	-
Conv3_x	$\begin{bmatrix} 3 \times 3/1, 1 \\ 3 \times 3/1, 1 \end{bmatrix} \times 2$	$32 \times 32 \times 96$	662K
Conv3a	$1 \times 1/1$	$32 \times 32 \times 192$	18K
MFM3a	-	$32 \times 32 \times 96$	-
Conv3	$3 \times 3/1, 1$	$32 \times 32 \times 384$	331K
MFM3	-	$32 \times 32 \times 192$	-
Pool3	$2 \times 2/2$	$16 \times 16 \times 192$	-
Conv4_x	$\begin{bmatrix} 3 \times 3/1, 1 \\ 3 \times 3/1, 1 \end{bmatrix} \times 3$	$16 \times 16 \times 192$	3981K
Conv4a	$1 \times 1/1$	$16 \times 16 \times 384$	73K
MFM4a	-	$16 \times 16 \times 192$	-
Conv4	$3 \times 3/1, 1$	$16 \times 16 \times 256$	442K
MFM4	-	$16 \times 16 \times 128$	-
Conv5_x	$\begin{bmatrix} 3 \times 3/1, 1 \\ 3 \times 3/1, 1 \end{bmatrix} \times 4$	$16 \times 16 \times 128$	2356K
Conv5a	$1 \times 1/1$	$16 \times 16 \times 256$	32K
MFM5a	-	$16 \times 16 \times 128$	-
Conv5	$3 \times 3/1, 1$	$16 \times 16 \times 256$	294K
MFM5	-	$16 \times 16 \times 128$	-
Pool4	$2 \times 2/2$	$8 \times 8 \times 128$	-
fc1	-	512	4,194K
MFM_fc1	-	256	-
Total	-	-	12,637K

ROC-кривая на датасете MegaFace и LFW соответственно:



Вывод:

Light CNN – одна из наилучших архитектур по критерию эффективности. Модель сама по себе легкая: быстро обучается, даёт быстрое предсказание и занимает мало места в памяти устройства; обладает относительно высокой точностью. Если сравнивать с конкурентом, ResNet архитектурой, очевидным победителем выйдет Light CNN. Но, к сожалению, с увеличением количества фильтров или глубины (количество слоев), как в случае с Inception-ResNet, точность модели поднять не удается выше её предела. Отсюда следует вывод, что высокая точность и относительная легкость модели обеспечиваются за счет ее уникальной архитектуры.

2.2.4. Реализация с помощью MTCNN + Inception-ResNet-V1.

В реальной задаче, помимо модели идентификации, нужно использовать инструменты для обнаружения лица. Таким образом мы сможем облегчить задачу нейросети-идентификатора и увеличить точность предсказания, отправив только лицо человека, а не всю картинку целиком с посторонним “шумом”. Существует множество различных алгоритмов, про них было описано выше, но самым эффективным, что не странно, является нейросеть.

Идея

Человеческое лицо – это уникальное искусство природы. У него есть два глаза с бровями, один нос, один рот и уникальная структура скелета лица, которая влияет на структуру щек, челюсти и лба. Как можно сравнивать двух людей по их лицам? На самом деле наш мозг натренирован естественным образом: он может **определить местоположение лица** и заведомо знает, какие области лица являются общими для всех человеческих лиц и их учитывать не стоит; а вот остальные участки мозг быстро обрабатывает и по ним идентифицирует личность: **прогоняет образы через память и выдает наиболее похожий результат**. Таким образом мы узнаем знакомых нам людей по их лицу. Но, к сожалению, мозг можно обмануть. Если показать лицо очень похожего человека, то мозг в большинстве случаев скажет вам: “Да, я его знаю!” Это явление связано с тем, что лица людей уникальны и вероятность повстречать или познакомиться с человеком, чье лицо сильно напоминает вашего друга, очень мала, а значит мозгу просто не на чем обучаться в таком случае. Но это уже другая интересная тема для обсуждения, которую мы затрагивать не будем.

Аналогичное явление происходит в модели MTCNN. Но её можно натренировать таким образом, чтобы она могла различать похожих людей, в отличии от мозга человека. Это зависит от датасета, на котором происходит обучение. Давайте более подробно изучим принцип работы данного алгоритма.

Рассмотрим 3 главных этапа:

- [Обрезка и выравнивание лица](#)
- [Генерация образа \(embedding\)](#)
- [Сравнение образов](#)

I этап.

Для выполнения первого этапа создадим 3 легкие нейросети с разной архитектурой.

- 1) Нейросеть P-Net.

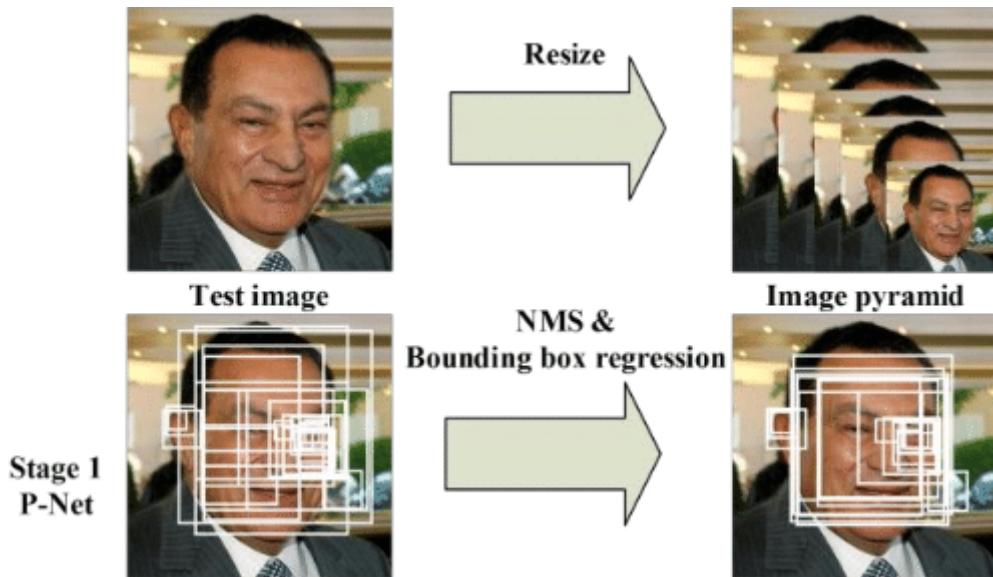


Рисунок 40. Первичный поиск лица с помощью Proposal Network

- 2) Нейросеть R-net.

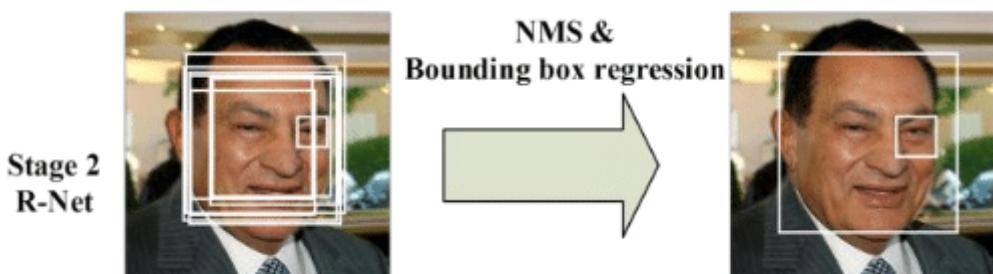


Рисунок 41. Обработка первичного поиска и уточнение с помощью Refinement Network

- 3) Нейросеть O-net.

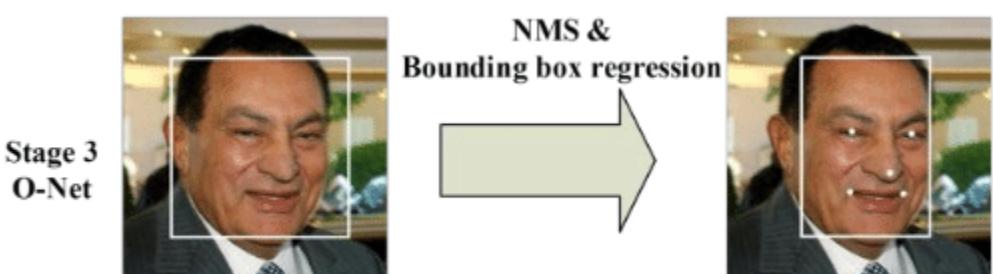
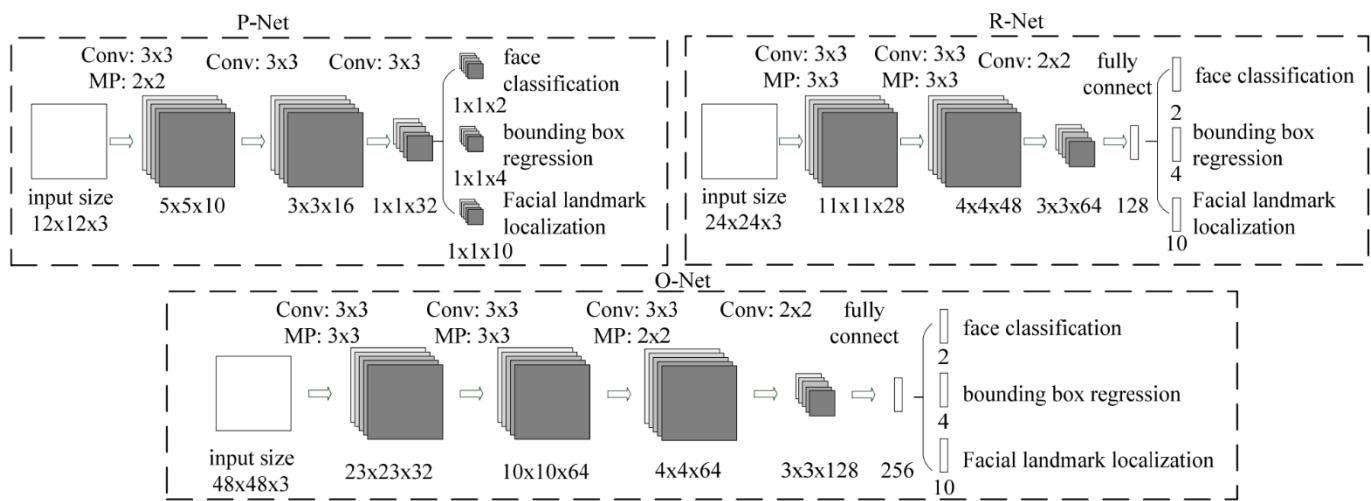


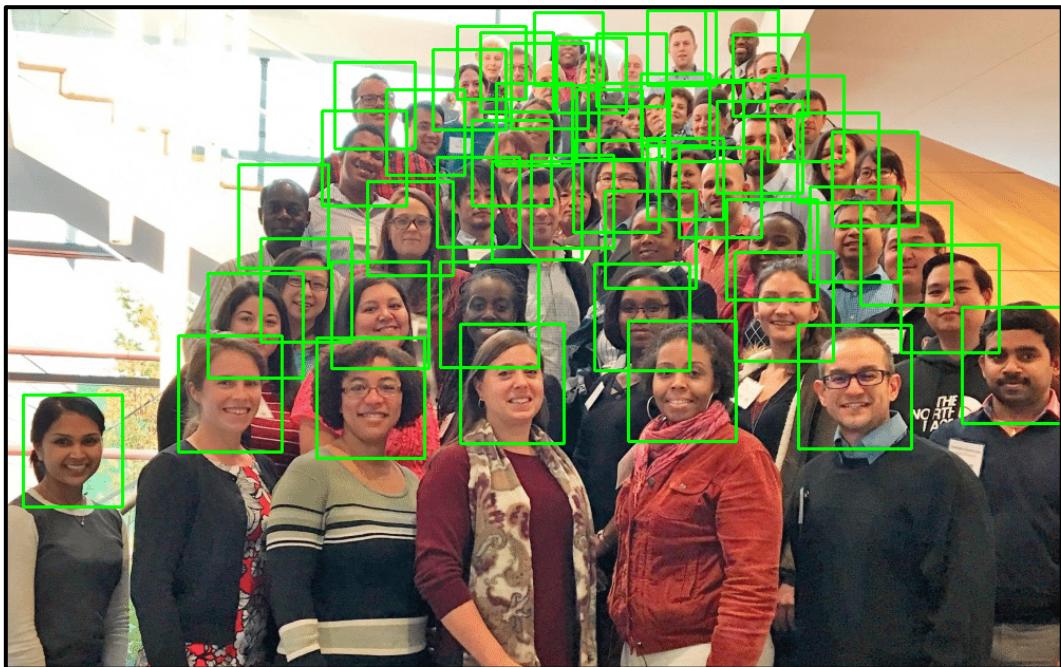
Рисунок 42. Последний этап: создание окончательной ограничительной рамки и идентификация точек с помощью Output Network

Этот метод распознавания лиц имеет преимущество при различных условиях освещения, вариациях позы лица и визуальных вариациях лица.

Ниже представлена архитектура каждой нейросети:



Пример распознавания лиц на фото:



II этап.

Для реализации второго этапа используется предобученная модель на датасете VGGFace2 архитектуры Inception-ResNet-V1.

После нахождения и обрезания лица, обработанная фотография отправляется в нейросеть-идентификатор. Нейросеть “чудным образом” обрабатывает лицо. На выходе мы получаем 512-мерный embedding-вектор, содержащий в себе уникальные признаки.

III этап.

Для реализации третьего этапа использовалась евклидова метрика. С помощью неё сравнивалось два вектора, после чего делался вывод, похожи люди или нет.

2.3. Сравнение с помощью евклидовой метрики

Евклидова метрика (евклидово расстояние) — метрика в евклидовом пространстве — расстояние между двумя точками евклидова пространства, вычисляемое по теореме Пифагора.

Для точек $p = (p_1, \dots, p_n)$ и $q = (q_1, \dots, q_n)$ евклидово расстояние определяется следующим образом:

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}.$$

Евклидова метрика — наиболее естественная функция расстояния, возникающая в геометрии, отражающая интуитивные свойства расстояния между точками. При этом существуют и другие метрики в евклидовых пространствах, применяемые как в геометрии, так и в приложениях; параметрическое расстояние Минковского обобщает ряд таких метрик, при параметре со значением 2 оно обращается в евклидову метрику, но в нашем случае нет нужны использовать эту метрику.

3. Хранение данных

Надежное хранение данных и эффективное их использование – непростая задача. Проблемы возникают, когда повышаются объемы информации. Современные системы хранения данных – это сложные программно-аппаратные комплексы, каждый из которых специально разрабатывается под нужды конкретного пользователя.

3.1. Виды СУБД и их особенности

СУБД — это комплекс языковых и программных средств, предназначенный для создания, ведения и совместного использования БД многими пользователями. Обычно СУБД различают по используемой модели данных. Так, СУБД, базирующиеся на использовании реляционной модели данных, называют реляционными СУБД. Системы управления базами данных помогают отсортировать информацию, а также связать базы данных между собой, при этом предоставив отчет об изменениях и зарегистрированных событиях.

При сравнении различных популярных баз данных, следует учитывать, удобна ли для пользователя и масштабируема ли данная конкретная СУБД, а также убедиться, что она будет хорошо интегрироваться с другими продуктами, которые уже используются. Кроме того, во время выбора следует принять во внимание стоимость системы и поддержки, предоставляемой разработчиком.

Существует несколько популярных СУБД, как платных, так и бесплатных, которые можно рекомендовать для хранения данных. Далее будут рассмотрены самые популярные бесплатные реляционные СУБД.

Реляционные системы реализуют реляционную модель работы с данными, которая определяет всю хранимую информацию как набор связанных записей и атрибутов в таблице.

СУБД такого типа используют структуры (таблицы) для хранения и работы с данными. Каждый столбец (атрибут) содержит свой тип информации. Каждая запись в базе данных, обладающая уникальным ключом, передаётся в строку таблицы, и её атрибуты отображаются в столбцах таблицы.

3.1.1. SQLite.

SQLite – изумительная библиотека, встраиваемая в приложение, которое её использует. Будучи файловой БД, она предоставляет отличный набор инструментов для более простой (в сравнении с серверными БД) обработки любых видов данных.

Когда приложение использует SQLite, их связь производится с помощью функциональных и прямых вызовов файлов, содержащих данные (например, баз данных SQLite), а не интерфейса, что повышает скорость и производительность операций.

Поддерживаемые типы данных:

- **NULL:** NULL-значение.
- **INTEGER:** целое со знаком, хранящееся в 1, 2, 3, 4, 6, или 8 байтах.
- **REAL:** число с плавающей запятой, хранящееся в 8-байтовом формате IEEE.
- **TEXT:** текстовая строка с кодировкой UTF-8, UTF-16BE или UTF-16LE.

- **BLOB:** тип данных, хранящийся точно в таком же виде, в каком и был получен.

Преимущества

- **Файловая:** вся база данных хранится в одном файле, что облегчает перемещение.
- **Стандартизированная:** SQLite использует SQL; некоторые функции опущены (RIGHT OUTER JOIN или FOR EACH STATEMENT), однако, есть и некоторые новые.
- **Отлично подходит для разработки и даже тестирования:** во время этапа разработки большинству требуется масштабируемое решение. SQLite, со своим богатым набором функций, может предоставить более чем достаточный функционал, при этом будучи достаточно простой для работы с одним файлом и связанной сишной библиотекой.

Недостатки

- **Отсутствие пользовательского управления:** продвинутые БД предоставляют пользователям возможность управлять связями в таблицах в соответствии с привилегиями, но у SQLite такой функции нет.
- **Невозможность дополнительной настройки:** опять-таки, SQLite нельзя сделать более производительной, поковырявшись в настройках — так уж она устроена.

Когда стоит использовать SQLite

- **Встроенные приложения:** все портируемые не предназначенные для масштабирования приложения — например, локальные однопользовательские приложения, мобильные приложения или игры.
- **Система доступа к дисковой памяти:** в большинстве случаев приложения, часто производящие прямые операции чтения/записи на диск, можно перевести на SQLite для повышения производительности.
- **Тестирование:** отлично подойдёт для большинства приложений, частью функционала которых является тестирование бизнес-логики.

Когда не стоит использовать SQLite

- **Многопользовательские приложения:** если вы работаете над приложением, доступом к БД в котором будут одновременно пользоваться несколько человек, лучше выбрать полнофункциональную РСУБД — например, MySQL.
- **Приложения, записывающие большие объёмы данных:** одним из ограничений SQLite являются операции записи. Эта РСУБД допускает единовременное исполнение лишь одной операции записи.

3.1.2. MySQL.

MySQL — это самая популярная из всех крупных серверных БД. Разобраться в ней очень просто, да и в сети о ней можно найти большое количество информации. Хотя MySQL и не пытается полностью реализовать SQL-стандарты, она предлагает широкий функционал. Приложения общаются с базой данных через процесс-демон.

Поддерживаемые типы данных

- **TINYINT:** очень маленькое целое.
- **SMALLINT:** маленькое целое.
- **MEDIUMINT:** целое среднего размера.
- **INT или INTEGER:** целое нормального размера.
- **BIGINT:** большое целое.
- **FLOAT:** знаковое число с плавающей запятой одинарной точности.
- **DOUBLE, DOUBLE PRECISION, REAL:** знаковое число с плавающей запятой двойной точности.
- **DECIMAL, NUMERIC:** знаковое число с плавающей запятой.
- **DATE:** дата.
- **DATETIME:** комбинация даты и времени.
- **TIMESTAMP:** отметка времени.
- **TIME:** время.
- **YEAR:** год в формате YY или YYYY.
- **CHAR:** строка фиксированного размера, дополняемая справа пробелами до максимальной длины.
- **VARCHAR:** строка переменной длины.
- **TINYBLOB, TINYTEXT:** BLOB- или TEXT-столбец длиной максимум $2^{8} - 1$ символов.
- **BLOB, TEXT:** BLOB- или TEXT-столбец длиной максимум $2^{16} - 1$ символов.
- **MEDIUMBLOB, MEDIUMTEXT:** BLOB- или TEXT-столбец длиной максимум $2^{24} - 1$ символов.
- **LONGBLOB, LONGTEXT:** BLOB- или TEXT-столбец длиной максимум $2^{32} - 1$ символов.
- **ENUM:** перечисление.
- **SET:** множества

Преимущества

- **Простота:** MySQL легко устанавливается. Существует много сторонних инструментов, включая визуальные, облегчающих начало работы с БД.
- **Много функций:** MySQL поддерживает большую часть функционала SQL.
- **Безопасность:** в MySQL встроено много функций безопасности.
- **Мощность и масштабируемость:** MySQL может работать с действительно большими объёмами данных, и неплохо походит для масштабируемых приложений.
- **Скорость:** пренебрежение некоторыми стандартами позволяет MySQL работать производительнее, местами срезая на поворотах.

Недостатки

- **Известные ограничения:** по определению, MySQL не может сделать всё, что угодно, и в ней присутствуют определённые ограничения функциональности.
- **Вопросы надёжности:** некоторые операции реализованы менее надёжно, чем в других РСУБД.
- **Застой в разработке:** хотя MySQL и является open-source продуктом, работа над ней сильно заторможена. Тем не менее, существует несколько БД, полностью основанных на MySQL (например, MariaDB). Кстати, подробнее о родстве MariaDB и MySQL можно из нашего интервью с создателем обеих РСУБД — Джеймсом Боттомли.

Когда стоит использовать MySQL

- **Распределённые операции:** когда вам нужен функционал больший, чем может предоставить SQLite, стоит использовать MySQL.
- **Высокая безопасность:** функции безопасности MySQL предоставляют надёжную защиту доступа и использования данных.
- **Веб-сайты и приложения:** большая часть веб-ресурсов вполне может работать с MySQL, несмотря на ограничения. Этот инструмент весьма гибок и прост в обращении, что только на руку в длительной перспективе.
- **Кастомные решения:** если вы работаете над очень специфичным продуктом, MySQL подстроится под ваши потребности благодаря широкому спектру настроек и режимов работы.

Когда не стоит использовать MySQL

- **SQL-совместимость:** поскольку MySQL не пытается полностью реализовать стандарты SQL, она не является полностью совместимой с SQL. Из-за этого могут возникнуть проблемы при интеграции с другими РСУБД.
- **Конкурентность:** хотя MySQL неплохо справляется с операциями чтения, одновременные операции чтения-записи могут вызвать проблемы.
- **Недостаток функций:** в зависимости от выбора движка MySQL может недоставать некоторых функций.

3.1.3. PostgreSQL.

PostgreSQL – это самая продвинутая РСУБД, ориентирующаяся в первую очередь на полное соответствие стандартам и расширяемость. PostgreSQL, или Postgres, пытается полностью соответствовать SQL-стандартам ANSI/ISO. PostgreSQL отличается от других РСУБД тем, что обладает объектно-ориентированным функционалом, в том числе полной поддержкой концепта ACID (Atomicity, Consistency, Isolation, Durability).

Будучи основанным на мощной технологии Postgres отлично справляется с одновременной обработкой нескольких заданий. Поддержка конкурентности реализована с использованием MVCC (Multiversion Concurrency Control), что также обеспечивает совместимость с ACID. Хотя эта РСУБД не так популярна, как MySQL, существует много сторонних инструментов и библиотек для облегчения работы с PostgreSQL.

Поддерживаемые типы данных

- **bigint:** знаковое 8-байтное целое.
- **bigserial:** автоматически инкрементируемое 8-битное целое.
- **bit [(n)]:** битовая строка фиксированной длины.
- **bit varying [(n)]:** битовая строка переменной длины.
- **boolean:** булевская величина.
- **box:** прямоугольник на плоскости.
- **bytea:** бинарные данные.
- **character varying [(n)]:** строка символов фиксированной длины.
- **character [(n)]:** строка символов переменной длины.

- **cidr:** сетевой адрес IPv4 или IPv6.
- **circle:** круг на плоскости.
- **date:** календарная дата.
- **double precision:** число с плавающей запятой двойной точности.
- **inet:** адрес хоста IPv4 или IPv6.
- **integer:** знаковое 4-байтное целое.
- **interval [fields] [(p)]:** временной промежуток.
- **line:** бесконечная прямая на плоскости.
- **lseg:** отрезок на плоскости.
- **macaddr:** MAC-адрес.
- **money:** денежная величина.
- **path:** геометрический путь на плоскости.
- **point:** геометрическая точка на плоскости.
- **polygon:** многоугольник на плоскости.
- **real:** число с плавающей запятой одинарной точности.
- **smallint:** знаковое 2-байтное целое.
- **serial:** автоматически инкрементируемое 4-битное целое.
- **text:** строка символов переменной длины.
- **time [(p)] [without time zone]:** время суток (без часового пояса).
- **time [(p)] with time zone:** время суток (с часовым поясом).
- **timestamp [(p)] [without time zone]:** дата и время (без часового пояса).
- **timestamp [(p)] with time zone:** дата и время (с часовым поясом).
- **tsquery:** запрос текстового поиска.
- **tsvector:** документ текстового поиска.
- **txid_snapshot:** снэпшот ID пользовательской транзакции.
- **uuid:** уникальный идентификатор.
- **xml:** XML-данные.

Преимущества

- **Полная SQL-совместимость.**
- **Сообщество:** PostgreSQL поддерживается опытным сообществом 24/7.
- **Поддержка сторонними организациями:** несмотря на очень продвинутые функции, PostgreSQL используется в многих инструментах, связанных с РСУБД.
- **Расширяемость:** PostgreSQL можно программно расширить за счёт хранимых процедур.
- **Объектно-ориентированность:** PostgreSQL — не только реляционная, но и объектно-ориентированная СУБД.

Недостатки

- **Производительность:** В простых операциях чтения PostgreSQL может уступать своим соперникам.
- **Популярность:** из-за своей сложности инструмент не очень популярен.
- **Хостинг:** из-за вышеперечисленных факторов проблематично найти подходящего провайдера.

Когда стоит использовать PostgreSQL

- **Целостность данных:** если приоритет стоит на надёжность и целостность данных, PostgreSQL — лучший выбор.
- **Сложные процедуры:** если ваша БД должна выполнять сложные процедуры, стоит выбрать PostgreSQL в силу её расширяемости.
- **Интеграция:** если в будущем вам предстоит перемещать всю базу на другое решение, меньше всего проблем возникнет с PostgreSQL.

Когда не стоит использовать PostgreSQL

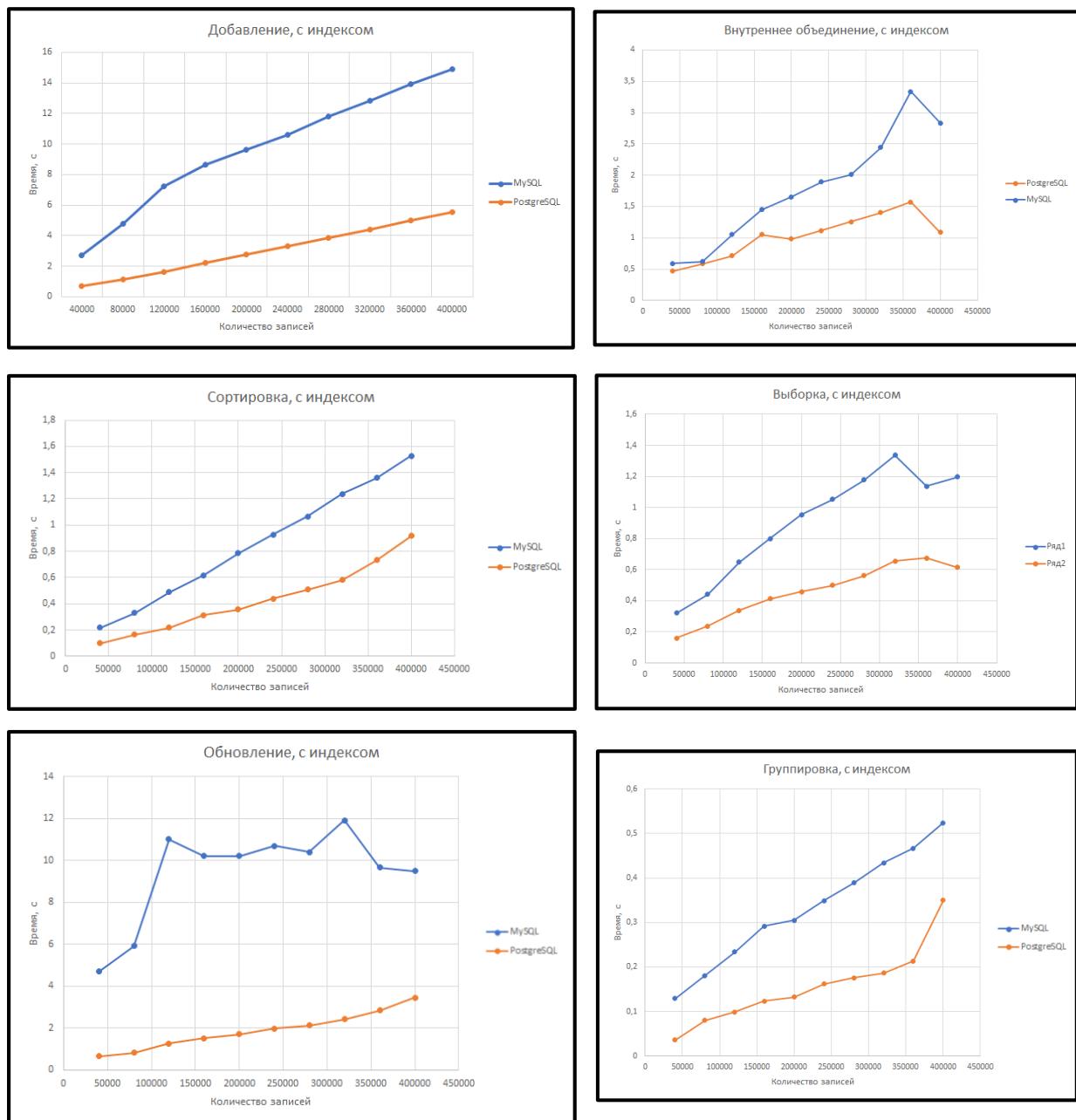
- **Простые ситуации:** если вам не требуется повышенная надёжность, поддержка ACID.

3.2. Сравнение СУБД

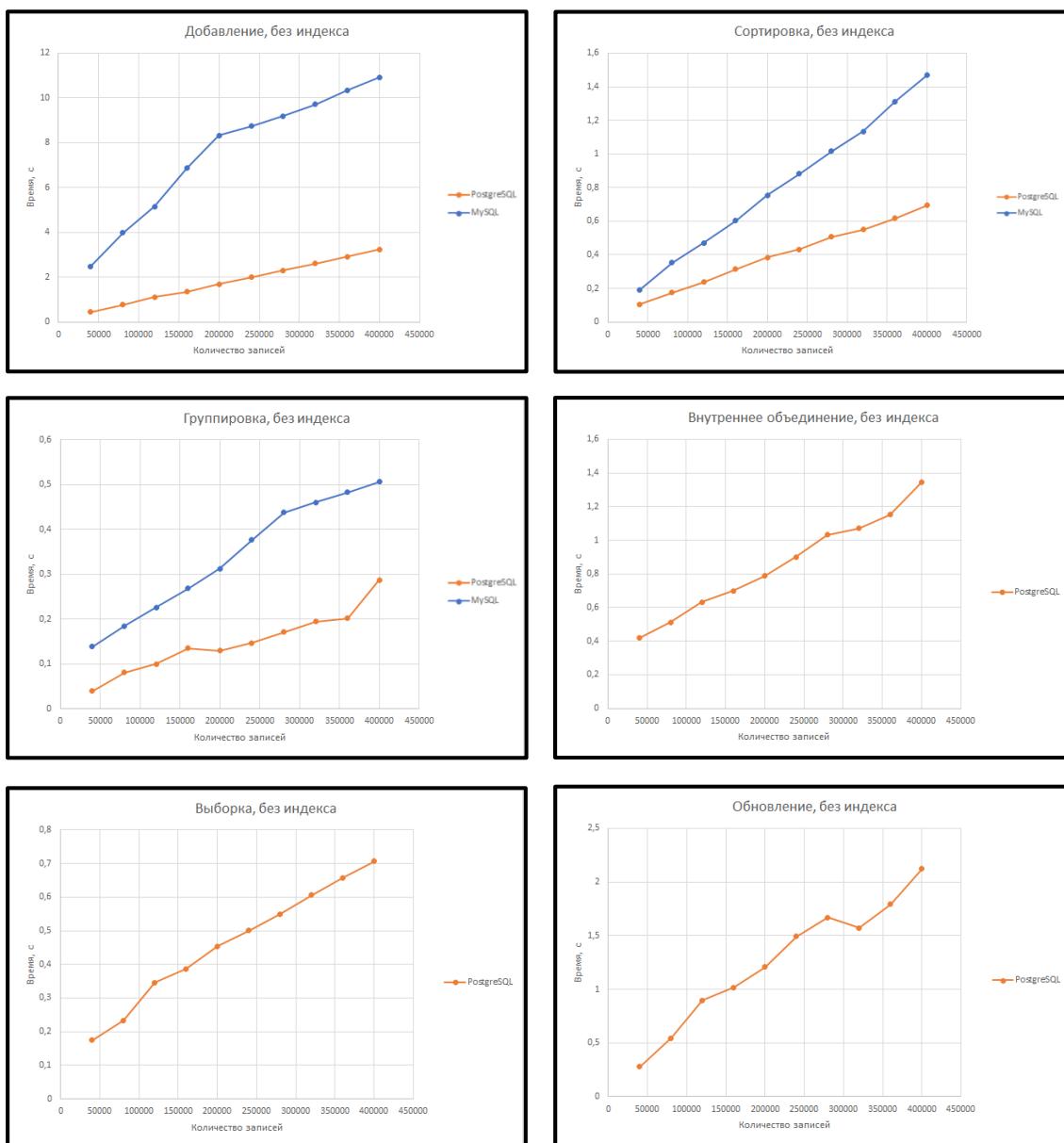
Выбор пал на две СУБД – MySQL и PostgreSQL.

В приоритете стояла скорость работы базы данных с большим количеством строк (более 5М) и производительность (считывание вектора, его преобразование и расчет дистанции между целевым вектором). Для этого были произведены различные синтетические тесты.

С использованием индексации.



Без использования индексации.



3.3. Выбор СУБД

Исходя из результатов тестов выше, окончательно была выбрана СУБД PostgreSQL. Она обладает всеми необходимыми возможностями и высокой скоростью считывания данных.

Основные плюсы, которыми обладает PostgreSQL:

- Поддержка большого количества типов данных. Массивы (в том числе и многомерные), JSON, XML, INET и прочие. Кроме того, можно самому создать свой тип для удобства использования.
- Подготовленные запросы. Кеширование данных для каждого подготовленного оператора.
- Производительность.
- Огромный размер базы данных:

Максимальный размер базы данных	Неограничен
Максимальный размер таблицы	32 ТВ
Максимальный размер строки	1.6 ТВ
Максимальный размер поля	1 GB
Максимальное количество строк в таблице	Неограничено
Максимальное количество столбцов в таблице	250-1600 в зависимости от типа столбца
Максимальное количество индексов в таблице	Неограничено

- Целостность данных. PostgreSQL стремится соответствовать стандарту ANSI-SQL:2008, отвечает требованиям ACID (атомарность, согласованность, изолированность и надежность) и известен своей ссылочной и транзакционной целостностью.

4. Выдача результата

После сбора и обработки данных необходимо продемонстрировать результат. Для этой цели удобно использовать чат-бот (специальная программа, работающая внутри мессенджера и имеющая возможность автоматического взаимодействия с пользователем, получения и отправки данных). Платформой для создания такого бота был выбран Telegram, так как этот мессенджер один из самых популярных и имеет достаточно подробную документацию для разработки.

4.1. Работа с Telegram-api

Боты – это специальные аккаунты в Telegram. Они не требуют номер телефона при создании, и пользователи могут взаимодействовать с ним по средствам обычных сообщений. Ответы бота контролируются за счет HTTPS запросов к специальному API.

4.1.1. Авторизация.

Самый простой способ создания и управления ботом в Telegram это взаимодействие с опять же ботом @BotFather. У бота есть два имени: первое отображается в чатах и его можно будет сменить в дальнейшем, а второе это уникальное имя (username), которое должно заканчиваться на bot, содержать только цифры, латинские буквы и символ нижнего подчеркивания, в то же время его длина должна быть от 5 до 32 символов.

После настройки этих двух имен присыпается токен для работы с API. Для упрощения работы с ним существует HTTP-интерфейс Bot API, а также специальные подмодули, одним из которых является telegram.ext. Они и будут использоваться для взаимодействия с API. Если необходимо, токен можно получить заново с помощью команды /token (команда сгенерирует новый токен).

Для отправки запросов и получения ответов из Telegram необходимо использовать такие сервисы как VPN или proxy. Они позволяют получить доступ к Telegram за счет того, что выступают некими посредниками между серверами мессенджера и нами.

VPN – virtual private network. За счет него IP-адрес всех запросов меняется на другой, в большинстве случаев это адрес из другой страны. Подключение в то же время зашифрованное и конфиденциальное, то есть считается достаточно безопасным из-за используемых продвинутых мер по ее осуществлению.

Основной минус бесплатных VPN состоит в том, что при его использовании скорость передачи данных значительно снижается и соответственно скорость передачи со сторонними сайтами, такими как vk.com в данном случае также снижается.

При использовании proxy передаваемый трафик представляется как трафик от определенного proxy-сервера. Иногда совершаемые действия могут записываться. Один из самых распространенных видов таких серверов это открытые proxy-серверы. Такими являются HTTP-proxy и SOCKS-proxy.

Первые proxy вида HTTP бывают трех видов:

- 1) Прозрачные – используют настоящий IP адрес. Используются очень редко и в основном для перенаправления на другой proxy-сервер.

- 2) Анонимные - подменяют IP адрес, но передают информацию о том, что используется proxy.
- 3) Элитные – по сравнению с анонимными скрывает то, что используется proxy.

Также существует модификация HTTPS-proxy, которая дополнительно использует шифрование данных. Но из-за этого она работает медленнее оригинала.

В то же время, HTTP-proxy обрабатывают только http запросы и зачастую работают быстрее VPN.

Второй вид proxy поддерживает разный тип трафика, из-за этого и работает медленнее HTTP-proxy. SOCKS-proxy пропускают трафик в чистом виде, не раскрывая http заголовков. Это позволяет получить доступ к серверу, который при этом не сможет узнать исходный IP-адрес и используется ли proxy или нет. Но существуют те же проблемы с безопасностью что и у HTTP-proxy.

У всех этих посредников передачи данных есть свои плюсы и минусы. При выборе одного из них нет гарантии что он будет на самом деле работать быстрее. Во-первых, сервера могут находиться достаточно далеко от источника и получателя сигнала, что замедлит работу. Во-вторых, нагрузка даже на один и тот же сервер может различаться в разное время, что может повлиять не только на скорость передачи данных, но и на само подключение к нему. Соответственно, если нет большой необходимости в шифровании передаваемого трафика возможно использование proxy или VPN, а при значительном снижении скорости достаточно переключиться на другой сервер передачи.

4.2.2. Интерфейс.

С помощью @BotFather можно настроить также внешние параметры для бота такие как имя (/setname), описание бота (/setdescription), информация о боте (/setabouttext), фотография профиля бота (/setuserpic), возможные для обработки команды (/setcommands).

Для кастомизации клавиатуры можно использовать объект ReplyKeyboardMarkup. С его помощью можно создать кнопки, при нажатии на которые отправится соответствующая команда серверу.

Для нахождения человека достаточно отправить фотографию в сообщениях боту, поэтому кнопка для этого ни к чему. Достаточно будет кнопки для получения информации о боте, кнопка для получения информации о возможных командах и кнопки для отправки сообщения разработчикам.

Рассмотрим получения обратной связи от пользователя. Этот механизм очень важен для дальнейшего усовершенствования проекта, поэтому необходимо команда, с помощью которой пользователь сможет отправлять свое мнение о нем или писать о возникших проблемах, а также возможность получения их и отправки ответа. Для этого можно создать режим доступа администратора. Основными отличиями такого доступа будут являться такие особенности:

- 1) Изначальный доступ осуществляется по паролю и существует до тех пор, пока id человека существует в списке администраторов (id автоматически добавляется в список при правильно введенном пароле).
- 2) Существуют дополнительные команды, с помощью которых можно добавить данные новых пользователей из «ВКонтакте», посмотреть историю запросов и определить происходит ли сейчас поиск человека.

- 3) Можно получить и отправить ответ на отзыв пользователя. Причем ответ пересыпается пользователю после ответа прямо на это сообщение боту администратора.

4.2. Обработка запроса/фотографии

С помощью MTCNN на фотографии, полученной от пользователя, находится лицо, и его изображение отправляется в нейросеть для дальнейшей обработки. Далее происходит преобразование фотографии в вектор уникальных признаков. Полученный вектор сравнивается с другими векторами из базы данных с помощью евклидовой метрики. Находятся 10 наиболее похожих на человека с фотографии людей (тех, у которых наименьшее расстояние между векторами) и отправляются в сообщении пользователю.

5. Реализация

5.1. Краткое описание

Class Auth.

Содержит 2 типа авторизации:

- **Client_credentials_flow** – по токену приложения. Минимальный доступ.
- **ImplicitFlow** – авторизация через пользовательский аккаунт соц. сети с возможностью прохождения двухэтапной аутентификации. Полный доступ.

Class ParsePageVK.

Осуществляет парсинг страницы пользователя. Алгоритм проверяет последние 30 фотографий пользователя из 2 альбомов: “Фотографии со страницы” и “Фотографии на стене”. Также присутствуют ограничения: не более 10 найденных лиц для одного пользователя, не более 5 лиц на одной фотографии и не менее 15 фотографий в ОЗУ устройства перед сохранением, что ускоряет работу парсера и не сильно нагружает компьютер. Все параметры можно менять.

В связи с относительно недавним нововведением VK на закрытие страницы, возникла периодическая проблема с доступом к альбомам. Для этого был придуман способ, позволяющий проверить “аватарку” пользователя, и если на ней найдено лицо, добавляет фото в базу данных.

После получения фотографии, поиска лиц с помощью MTCNN и преобразования каждого в вектор, данные помещаются в СУБД на хранение.

Class DataBase.

Подключает к базе данных; содержит функции, отвечающие за сохранение вектора, поиска по базе данных, создание и удаление таблицы. Реализована многопоточность для поиска.

Class FinderVk.

Преобразует target-фото (отправленное пользователем) в вектор и вызывает поиск по базе данных.

Class ResetDB.

Пересоздает таблицу.

Class TelegramBot.

Подключает telegram-бота. Реализована очередь для поиска по базе данных и работы парсера на основе многопоточности.

Файл private.py.

Хранит токены, логины, пароли и прочую засекреченную информацию.

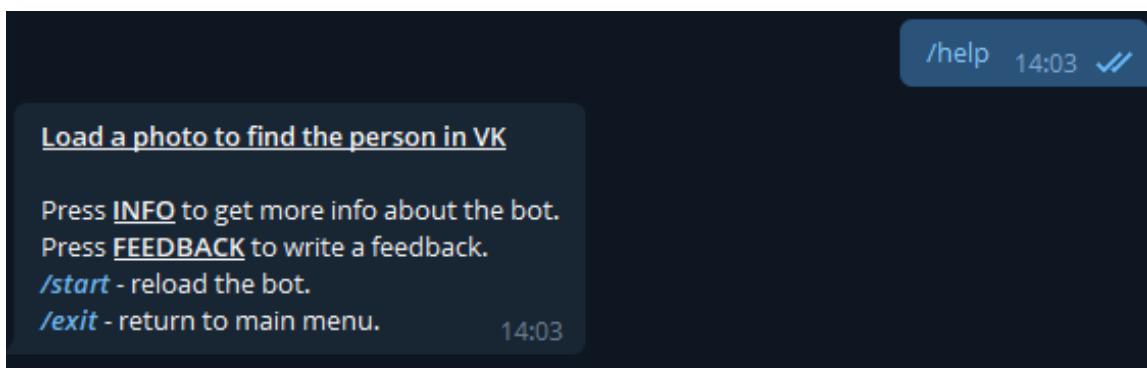
5.2. Telegram bot

Интерфейс.

Подсказки при вводе “/”:



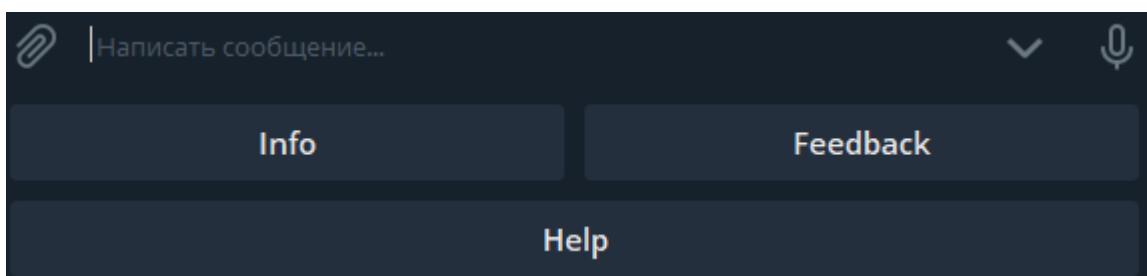
Ответ на команду “/help”:

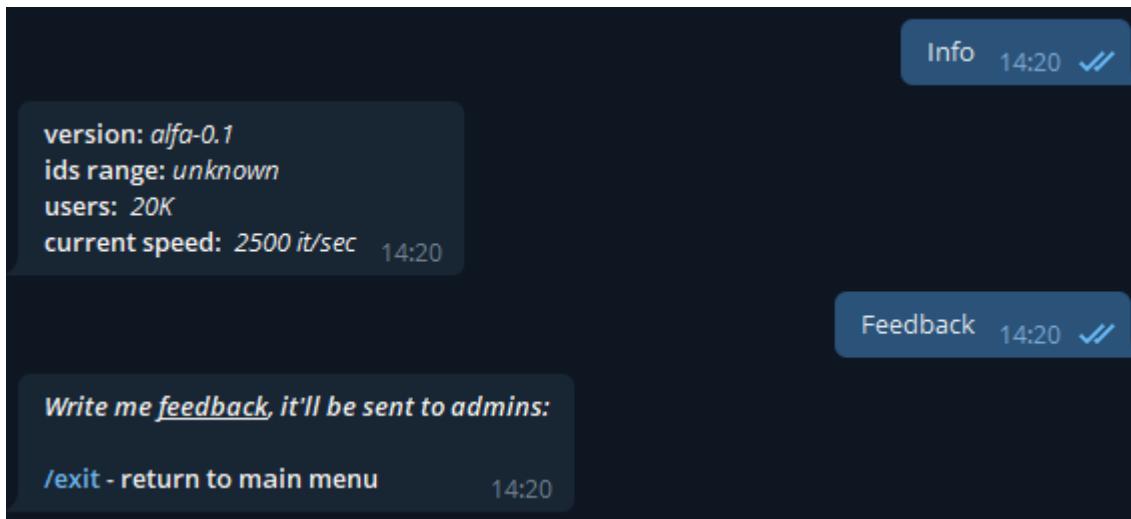


Ответ на команду “/start”:

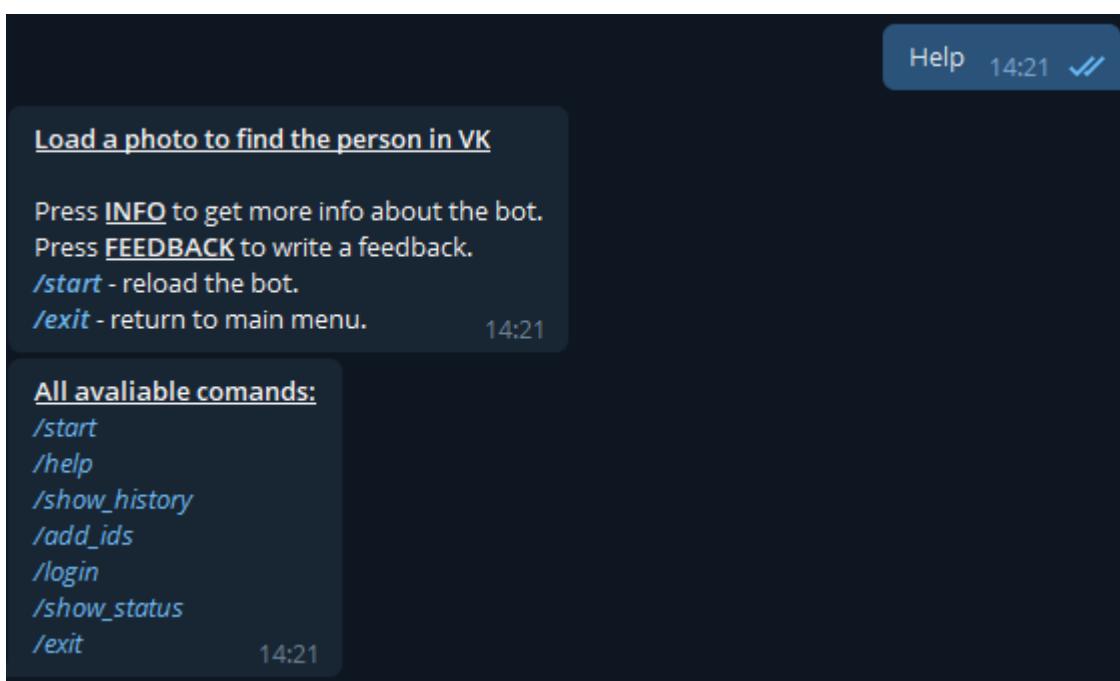


Использование кнопок:





Расширенное меню администратора:



Выдача результатов:

Telegram-бот выводит 10 наиболее релевантных результатов в порядке убывания: лучшая фотография человека и ссылку на страницу.

Литература

- VK-api:
 - <https://vk.com/dev/methods>
 - https://github.com/python273/vk_api
- Способы распознавания лиц
 - <https://habr.com/ru/company/synesis/blog/238129/>
- NeuralNetworks:
 - <https://neurohive.io/ru/osnovy-data-science/7-arhitektur-nejronnyh-setej-nlp/>
 - https://ru.wikipedia.org/wiki/Нейронная_сеть
 - CNN
 - <https://habr.com/post/309508/>
 - <https://neurohive.io/ru/osnovy-data-science/glubokaya-svertochnaja-nejronnaja-set/>
 - <https://arxiv.org/abs/1708.05234>
 - <https://habr.com/ru/company/odnoklassniki/blog/350566/>
 - <https://habr.com/ru/company/synesis/blog/238129/>
 - <https://api-2d3d-cad.com/python-face-detection/>
 - LightCNN
 - <https://github.com/AlfredXiangWu/LightCNN>
 - <https://arxiv.org/pdf/1511.02683.pdf>
 - <https://www.arxiv-vanity.com/papers/1511.02683/>
 - Inception-ResNet
 - <https://arxiv.org/pdf/1602.07261.pdf>
 - Wide ResNet.
 - <https://arxiv.org/pdf/1605.07146.pdf>
 - <https://github.com/szagoruyko/wide-residual-networks>
 - MTCNN
 - <https://www.pytorials.com/face-detection-matching-using-facenet/>
 - <https://github.com/ipazc/mtcnn>
 - СУБД:
 - <http://drach.pro/blog/hi-tech/item/145-/>
 - <https://tproger.ru/translations/sqlite-mysql-postgresql-comparison/>
 - <https://habr.com/ru/post/282764/>
 - <https://habr.com/ru/post/317394/>
 - А. Ю. Васильев «Работа с PostgreSQL: настройка и масштабирование. 3-е издание»
 - Евклидова метрика:
 - https://en.wikipedia.org/wiki/Euclidean_distance
 - Telegram-api:
 - <https://python-telegram-bot.readthedocs.io/en/stable/>