**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ**

**Кафедра ядерной физики**

**АНАЛИЗ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ ДЛЯ ЗАДАЧ КОМПЬЮТЕРНОГО ЗРЕНИЯ**

Курсовая работа

Окочутина Александра Васильевича

студента 5 курса,

специальность

ядерные физика и технологии

Научный руководитель:

доктор физико-математических наук,

профессор Чернявская Э.А.

Минск, 2016

**Оглавление**

[**ВВЕДЕНИЕ** 3](#_Toc452378350)

[**ГЛАВА 1 АНАЛИЗ АЛГОРИТМОВ РАСПОЗНАВАНИЯ ЛИЦ** 4](#_Toc452378351)

[**1.1 Общая архитектура процесса распознавания лиц** 4](#_Toc452378352)

[**1.2 Метод гибкого сравнения на графах (Elastic graph matching)** 5](#_Toc452378353)

[**1**.**3 Нейронные сети (Convolution Net, Deep Learning Net)** 7](#_Toc452378354)

[**1.4 Метод главных компонент (Principal Component Analysis)** 8](#_Toc452378355)

[**1.5 Метод Виоллы-Джонса** 10](#_Toc452378356)

[**1.6 Преимущества и недостатки каждого из алгоритмов** 14](#_Toc452378357)

[**1.7 Выбор оптимального алгоритма для программной реализации поставленной задачи** 15](#_Toc452378358)

[**ГЛАВА 2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ БИОМЕТРИЧЕСКОЙ АУТЕНТИФИКАЦИИ ПО ГЕОМЕТРИИ ЛИЦА** 16](#_Toc452378359)

[**2.1 Используемые компоненты.** 16](#_Toc452378360)

[**2.2 Нюансы программной реализации алгоритма.** 17](#_Toc452378361)

[**2.3 Результаты использования построенного алгоритма** 18](#_Toc452378362)

[**Заключение** 19](#_Toc452378363)

[**ПРИЛОЖЕНИЕ** 20](#_Toc452378364)

[**Код программной реализации алгоритма** 20](#_Toc452378365)

[**Список использованных источников** 26](#_Toc452378366)

# **ВВЕДЕНИЕ**

Машинное обучение – обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться. Машинное обучение находится на стыке математической статистики, методов оптимизации, классических математических дисциплин и практических, инженерных дисциплин, но имеет также и собственную специфику, связанную с проблемами вычислительной эффективности и переобучения. Целью методов данного класса является частичная или полная автоматизация решения сложных профессиональных задач в самых разных областях человеческой деятельности.

Спектр приложений:

1. Внедрение алгоритмов распознавания в медицинские учреждения для обработки анализов (автоматическое детектирование циркулирующих опухолевых клеток, детектирование сердечно-сосудистых заболеваний, сегментация МРТ изображений, экспрессия генов).
2. Внедрение алгоритмов машинного обучения в проекты CERN (классификация элементарных частиц, оптимизация сетевых хранилищ GRID).
3. Практическое применение алгоритмов в задачах астрономии (краткосрочное прогнозирование солнечной активности на базе снимков в видимом диапазоне и магнитограмм со спутника SOHO, исследование космических лучей предельно высоких энергий и быстропротекающих процессов в оптическом, рентгеновском и гамма-диапазонах длин волн, происходящих в верхних слоях атмосферы Земли на базе снимков спутника “Ломоносов”).
4. Биометрические системы аутентификации: отпечаток пальца, геометрия лица, аутентификация по почерку и голосу (например, система распознавания пассажиров в Московском метро, российские компании BioLink и Artec Group производят 3d сканеры лица, сканеры отпечатков пальцев и ПО для их обработки).
5. Применение алгоритмов в задачах распознавания речи, печатного текста и рукописного текста (amazon cloud service for recognition, google cloud service for recognition).
6. Атомэнергомаш внедрил алгоритмы машинного обучения для анализа надежности аварийной защиты реактора (Методика основана на многовариантном компьютерном моделировании аварии. Для обработки результатов моделирования используются искусственные нейронные сети.

Спектр применения данной технологии очень широк, поэтому ограничим его рассмотрение исключительно темой компьютерного зрения.

Таким образом цель представленной курсовой работы - анализ алгоритмов, применяющихся для предобработки и подготовки данных и дальнейших распознавания и классификации и сегментации образов на изображении (с использованием библиотеки компьютерного зрения OpenCV).

# **1 АНАЛИЗ АЛГОРИТМОВ РАСПОЗНАВАНИЯ ЛИЦ**

## **1.1 Общая архитектура процесса распознавания лиц**

Несмотря на огромные отличия на этапах вычисления признаков и сравнения их совокупностей между собой, все алгоритмы имеют нечто общее:

архитектуру процесса распознавания лиц.

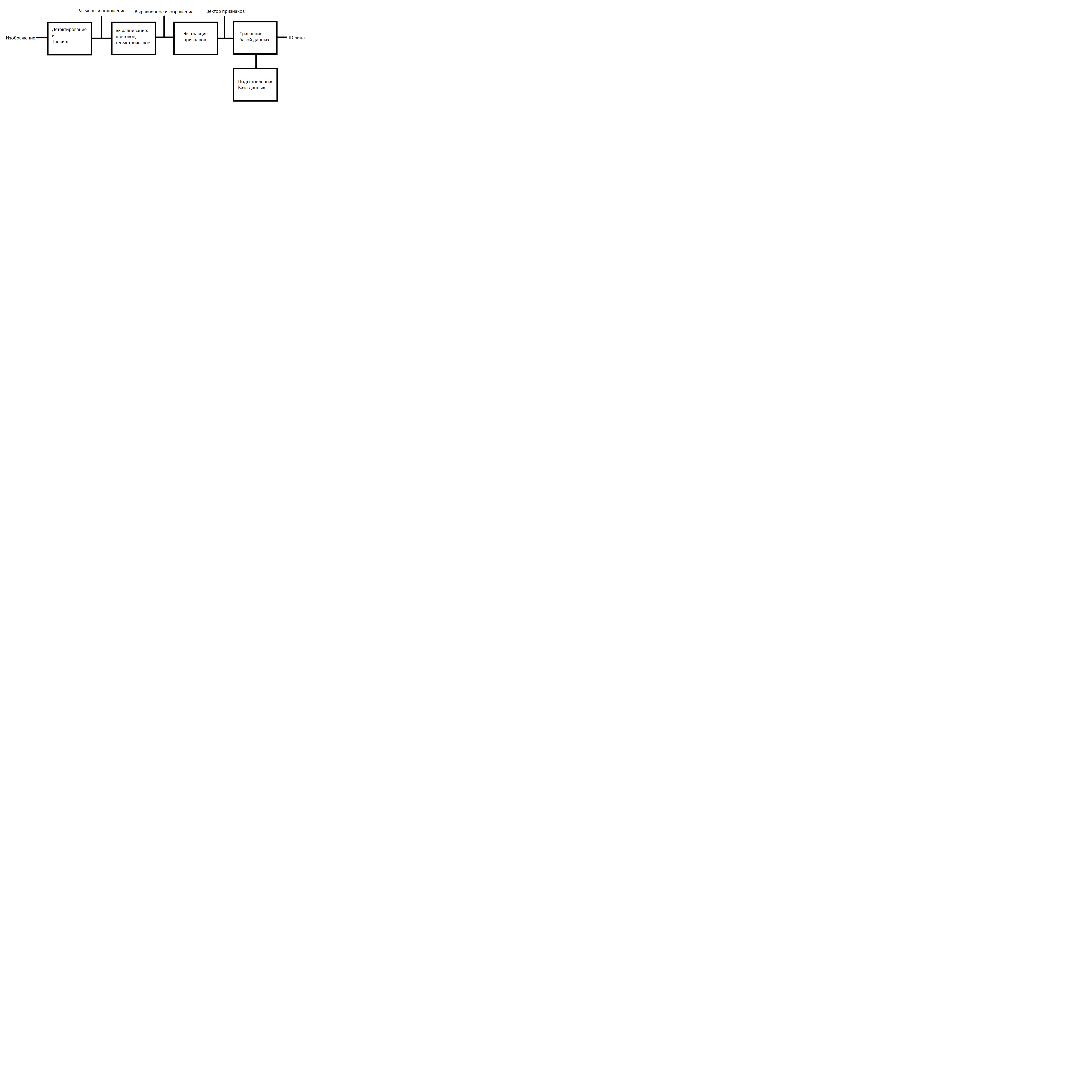


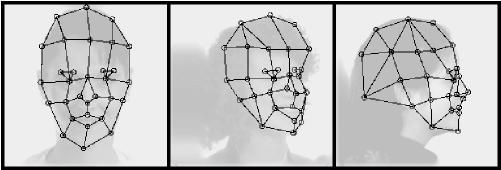
Рисунок 1.1 - Архитектура процесса распознавания лиц.

Согласно рисунку 1.1 на первом этапе процесса распознавания лиц производится детектирование и локализация лица на изображении. Далее на этапе распознавания производится выравнивание изображения лица (геометрическое и яркостное, контрастное), отображение в пространство признаков и непосредственно распознавание – сравнение вычисленных признаков с заложенными в базу данных эталонами.

Теперь можно перейти к непосредственному анализу основных алгоритмов, которые используются для идентификации личности по геометрии лица, на базе которого мы выберем наиболее оптимальные для поставленной задачи с целью их программной реализации.

## 

## **1.2 Метод гибкого сравнения на графах**

Суть **метода**  **гибкого сравнения на графах (Elastic graph matching)** 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.

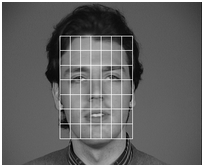


Рисунок 1.2 - Пример структуры графа: прямоугольная решетка и граф на основе антропометрических точек лица (слева на право).

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

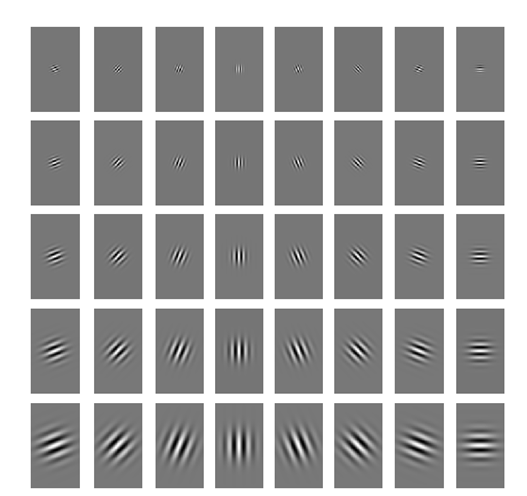


Рисунок 1.3 - Набор фильтров Габора и пример свертки с изображением.

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

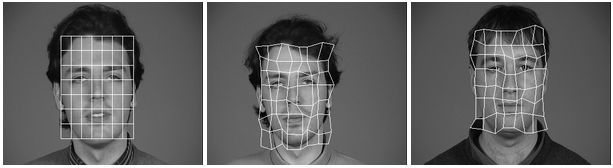


Рисунок 1.4 – Пример деформации графа в виде регулярной решетки

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. основе антропометрических точек лица (слева на право).

## 

## **1**.**3 Типы нейронных сетей для задач распознавания объектов (лиц)**

В настоящее время существует около десятка разновидности нейронных сетей (НС). Одним из самых широко используемых вариантов являться сеть, построенная на многослойном персептроне, которая позволяет классифицировать поданное на вход изображение, после настройки 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.   
Наилучшие результаты в области распознавания лиц (по результатам анализа публикаций) показала Convolutional Neural Network или сверточная нейронная сеть (далее – СНС), которая является логическим развитием идей таких архитектур НС как когнитрона и неокогнитрона. Успех обусловлен возможностью учета двумерной топологии изображения, в отличие от многослойного перцептрона.

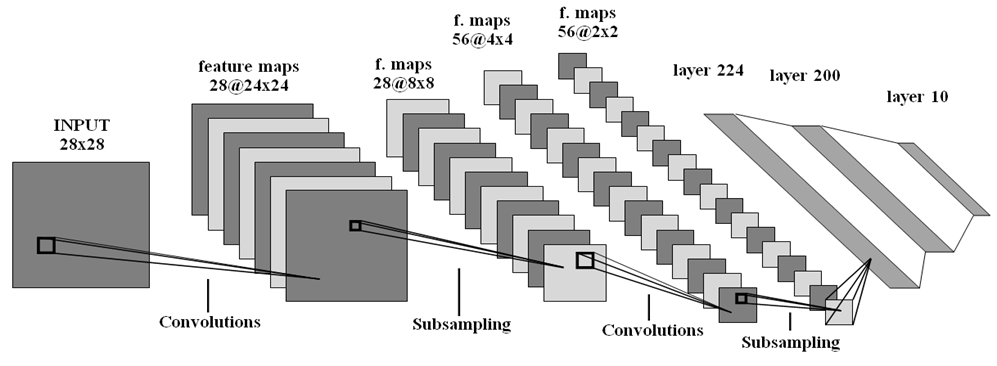


Рисунок 1.5 – Структура свёрточной нейронной сети.\

Отличительными особенностями СНС 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 СНС обеспечивает частичную устойчивость к изменениям масштаба, смещениям, поворотам, смене ракурса и прочим искажениям. Тестирование СНС на базе данных ORL, содержащей изображения лиц с небольшими изменениями освещения, масштаба, пространственных поворотов, положения и различными эмоциями, показало 96% точность распознавания.

Своё логическое развитие СНС получили в глубоком машинном обучении: глубокие нейронные сети (Deep Learning Network), которые заняли ведущее положение в распознавании образов. Благодаря им планка для качества методов компьютерного зрения значительно поднялась.

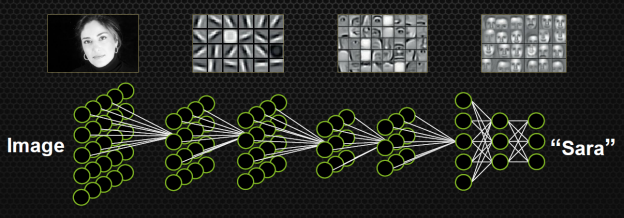


Рисунок 1.6 Структура глубокой нейронной сети.

Главной отличительной особенностью ГНС является то, что уровень абстракции признаков (которые ищет сеть) увеличивается с каждым слоем (в начале яркостные-градиенты, далее мелкие фрагменты лица, далее крупные фрагменты лица).

## **1.4 Метод главных компонент в задаче распознавания лиц**

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.

Суть метода главных компонент сводится к следующему.

Первый этап метода состоит из следующего шага - 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.

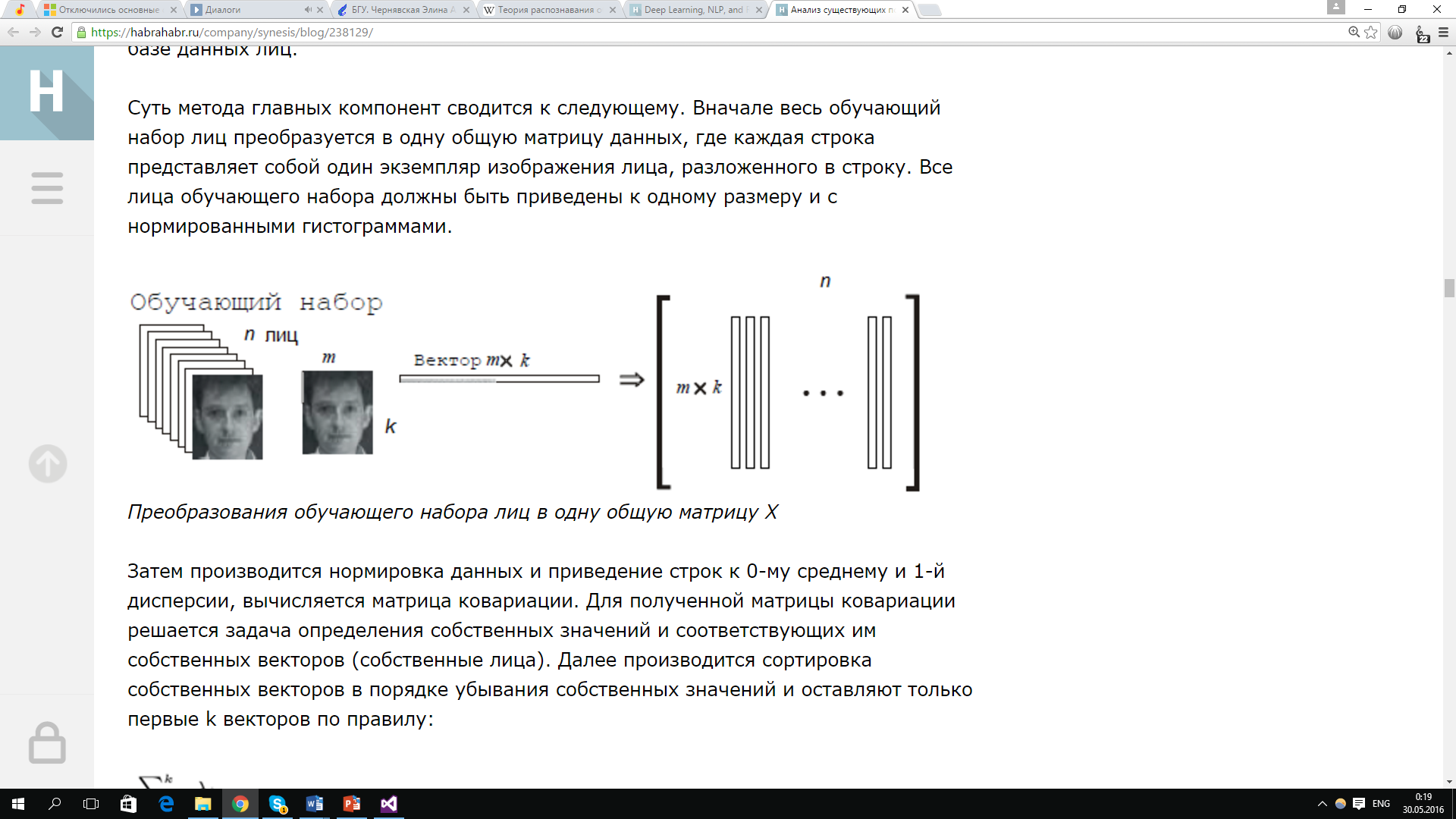


Рисунок 1.7 – Преобразование обучающего набора в матрицу X

Далее = 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).

, где - приведение к нулевому среднему.

, где – приведение к единичной дисперсии.

*,* получение матрицы ковариации.

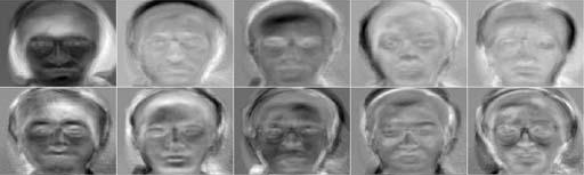
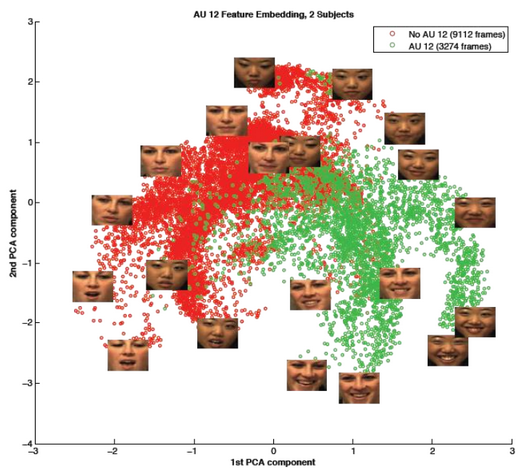


Рисунок 1.8 Графическое представление собственных векторов матрицы ковариации.

Следующий шаг - из найденных собственных векторов производится выборка *к*-собственных векторов (соответствующих наибольшим собственным значениям). Дальнейший этап является проецирование обучающей выборки на базис собственных векторов (на этом шаге происходит сжатие размерности и отображение в пространство признаков). На этапе идентификации происходит приведения к новому лицу, приведения его к единичной дисперсии и проецирования на базис собственных векторов, сравнение с эталонами идет в “сжатом” базисе (решающее правило – евклидово расстояние до эталона должно быть меньше порога). ( Посмотри!!! Немного сумбурно)



б

Рисунок 1.9 – Пример разделения двух лиц в базисе главных компонент.

На рисунке 1.10 представлена блок – схема алгоритма реализации метода главных компонент

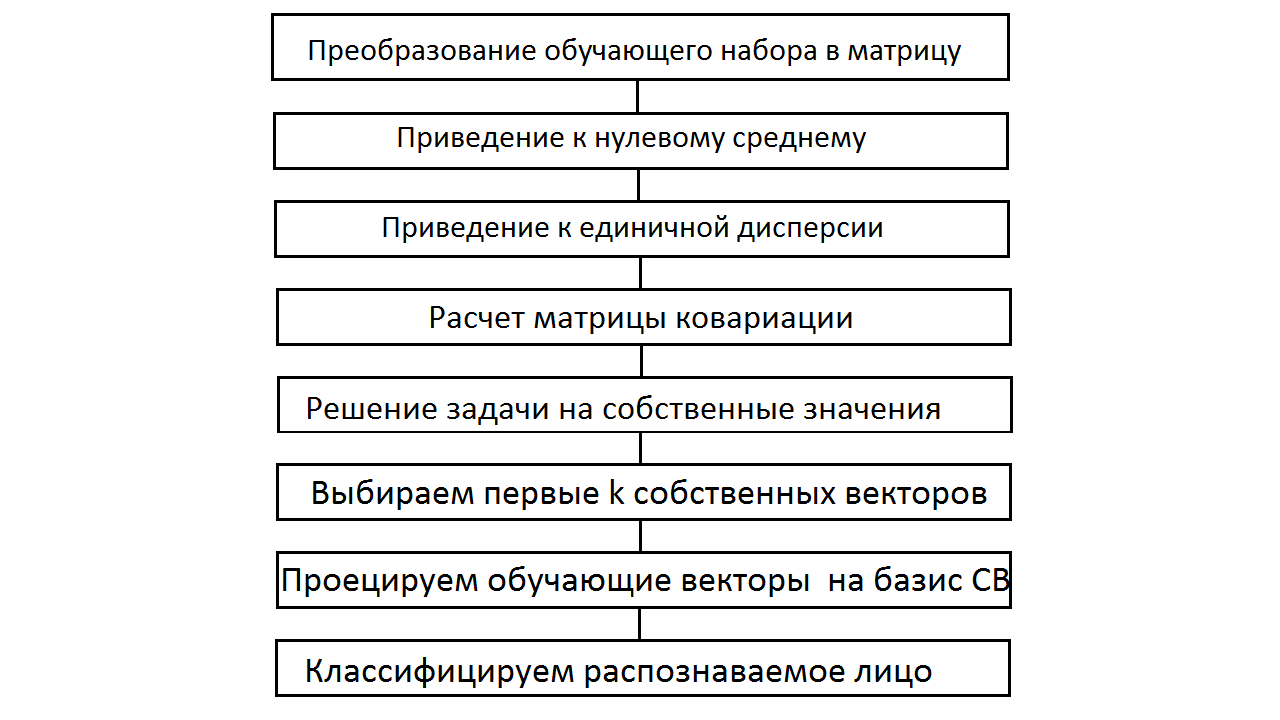


Рисунок 1.10 – Блок схема алгоритма PCA

## **1.5 Метод Виоллы-Джонса**

Рассмотрим основные принципы, на которых основан метод **Виоллы-Джонса**:

* используются изображения в [**интегральном представлении**](http://habrahabr.ru/blogs/algorithm/102919/), что позволяет вычислять быстро необходимые объекты;
* используются [**признаки Хаара**](http://en.wikipedia.org/wiki/Haar-like_features), с помощью которых происходит поиск нужного объекта (в данном контексте, лица и его черт);
* используется [**бустинг**](http://machinelearning.ru/wiki/index.php?title=%D0%91%D1%83%D1%81%D1%82%D0%B8%D0%BD%D0%B3) (от англ. **boost** – улучшение, усиление) для выбора наиболее подходящих признаков для искомого объекта на данной части изображения;
* все признаки поступают на вход [**классификатора**](http://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8), который даёт результат «верно» либо «ложь»;
* используются [**каскады признаков**](http://courses.graphicon.ru/main/vision/2011/lectures/7) для быстрого отбрасывания окон, где не найдено лицо.

В общем виде, задача обнаружения лица и черт лица человека на цифровом изображении выглядит именно так:

* имеется *изображение*, на котором *есть искомые объекты*.

Оно представлено *двумерной матрицей пикселей размером w\*h*, в которой каждый пиксель имеет значение:  
— от 0 до 255, если это черно-белое изображение;  
— от 0 до 2553, если это цветное изображение (компоненты R, G, B).

* в результате своей работы, алгоритм должен определить лица и их черты и *пометить их* – поиск осуществляется в *активной области* изображения *прямоугольными признаками*, с помощью которых и описывается найденное лицо и его черты:
* rectanglei = {x,y,w,h,a},  (1.1)  
  где x, y – координаты центра i-го прямоугольника, w – ширина, h – высота, a – угол наклона прямоугольника к вертикальной оси изображения.

Иными словами, применительно к рисункам и фотографиям используется *подход на основе сканирующего окна (scanning window)*: сканируется изображение окном поиска (так называемое, окно сканирования), а затем применяется классификатор к каждому положению.

***Признаки Хаара***

*Признак* — отображение f: X => Df, где Df — множество допустимых значений признака. Если заданы признаки f1,…,fn, то вектор признаков x = (f1(x),…,fn(x)) называется *признаковым описанием* объекта x ∈ X.

Признаковые описания допустимо отождествлять с самими объектами. При этом множество X = Df1\* …\* Dfn называют признаковым пространством. В стандартном методе Виолы – Джонса используются прямоугольные признаки, изображенные на рисунке ниже, они называются *примитивами Хаара*:

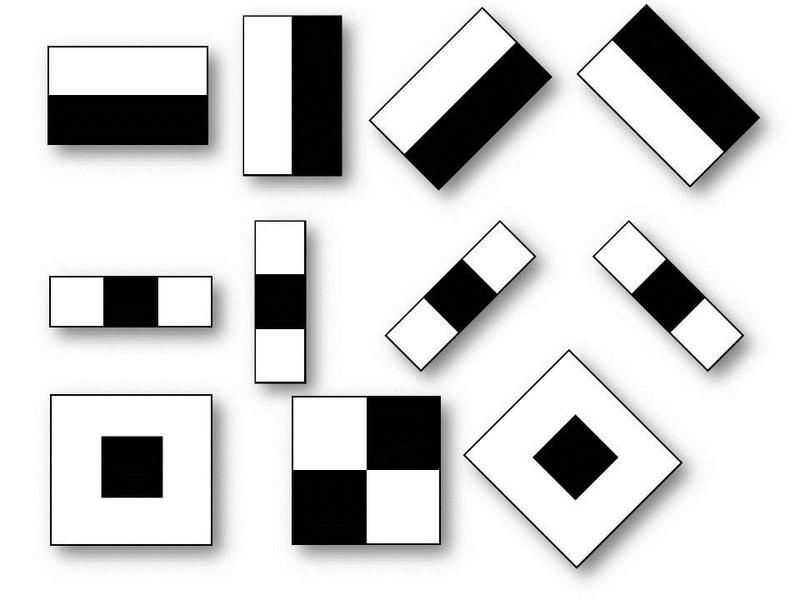


Рисунок 1.11 Признаки Хаара и области которые он ищет.

Вычисляемым значением такого признака будет:  
F = X-Y, (1.5)  
где X – сумма значений яркостей точек закрываемых светлой частью признака, а Y – сумма значений яркостей точек закрываемых темной частью признака. Признаки Хаара дают точечное значение *перепада яркости по оси X и Y соответственно*.

***Алгоритм сканирования*** ***окна*** с признаками выглядит так:

* есть исследуемое изображение, выбрано окно сканирования, выбраны используемые признаки;
* далее окно сканирования начинает последовательно двигаться по изображению с шагом в 1 ячейку окна (допустим, размер самого окна есть 24\*24 ячейки);
* при сканировании изображения в каждом окне вычисляется приблизительно 200 000 вариантов расположения признаков, за счет изменения масштаба признаков и их положения в окне сканирования;
* сканирование производится последовательно для различных масштабов;
* масштабируется не само изображение, а сканирующее окно (изменяется размер ячейки);
* все найденные признаки попадают к классификатору, который «выносит вердикт».

В процессе поиска вычислять все признаки на маломощных настольных ПК просто нереально. Следовательно, классификатор должен реагировать*только на определенное, нужное подмножество* всех *признаков*. Для этого применяется каскадная модель признаков.

***Алгоритм бустинга*** для поиска лиц с моей точки зрения таков:  
1. Определение слабых классификаторов по прямоугольным признакам;  
2. Для каждого перемещения сканирующего окна вычисляется прямоугольный признак на каждом примере;  
3. Выбирается наиболее подходящий порог для каждого признака;  
4. Отбираются лучшие признаки и лучший подходящий порог;  
5. Перевзвешивается выборка.

***Каскадная модель сильных классификаторов***– это по сути то же дерево принятия решений, где каждый узел дерева построен таким образом, чтобы детектировать почти все интересующие образы и отклонять регионы, не являющиеся образами. Помимо этого, узлы дерева размещены таким образом, что чем ближе узел находится к корню дерева, тем из меньшего количества примитивов он состоит и тем самым требует меньшего времени на принятие решения. Данный вид каскадной модели хорошо подходит для обработки изображений, на которых общее количество детектируемых образов мало.  
В этом случае метод может быстрее принять решение о том, что данный регион не содержит образ, и перейти к следующему. Пример каскадной модели сильных классификаторов:

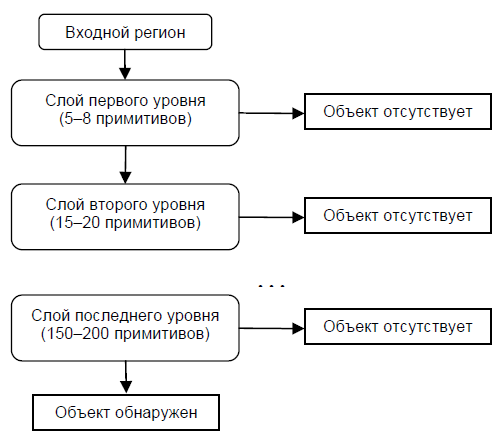


Рисунок 1.12 Пример каскадной модели сильных классификаторов.

## **1.6 Анализ преимущества и недостатков каждого из алгоритмов**

**1.6.1.Метод гибкого сравнения на графах**

***Преимущества:*** В отдельных публикациях указывается 95-97%-ая эффективность распознавания даже при наличии различных эмоциональных выражениях и изменении ракурса лица до 15 градусов.

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

**1.6.2. Нейронные сети (СНС)**

***Преимущества***: Благодаря двумерной топологии СНС обеспечивает частичную устойчивость к изменениям масштаба, смещениям, поворотам, смене ракурса и прочим искажениям.

***Недостатки:*** 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.

**1.6.3. Метод главных компонент.**

***Преимущества:*** Интуитивно понятное, наглядное внутреннее представление данных при работе с изображениями, сам метод классификации подразумевает сжатие размерности, что влияет на быстродействие, математический алгоритм хорошо зарекомендовал себя в практике.

***Недостатки:*** В тех случаях, когда на изображении лица присутствуют значительные изменения в освещенности или выражении лица, эффективность метода значительно падает.

**1.6.4. Метод Виолы-Джонса.**

***Преимущества:*** Обучение классификаторов идет очень медленно, в стандартной реализации невозможно детектирование под произвольным углом, алгоритм наиболее подходит для детектирования малого числа объектов.

***Недостатки:*** Результаты поиска объекта очень быстры(в режиме реального времени), малая вероятность ложного обнаружения лица.

## **1.7 Выбор оптимального алгоритма для программной реализации поставленной задачи**

После анализа преимуществ и недостатков каждого из алгоритма, для поставленной задачи: *программная реализация одного из данных методов, для идентификации личности из базы данных на видеопотоке (в режиме реального времени, в условиях фиксированного фона и фиксированного освещения).*

Была выбрана оптимальная связка двух алгоритмов, на мой взгляд, алгоритма PCA и алгоритма Виолы-Джонса.

##### На этапе детектирования и локализации лица, я буду использовать метод Виолы Джонса (ускоренный метод, и расширенные признаки Хаара). Для непосредственной классификации (ввиду отсутствия перепадов освещения и искажений) я выбрал алгоритм главных компонент.

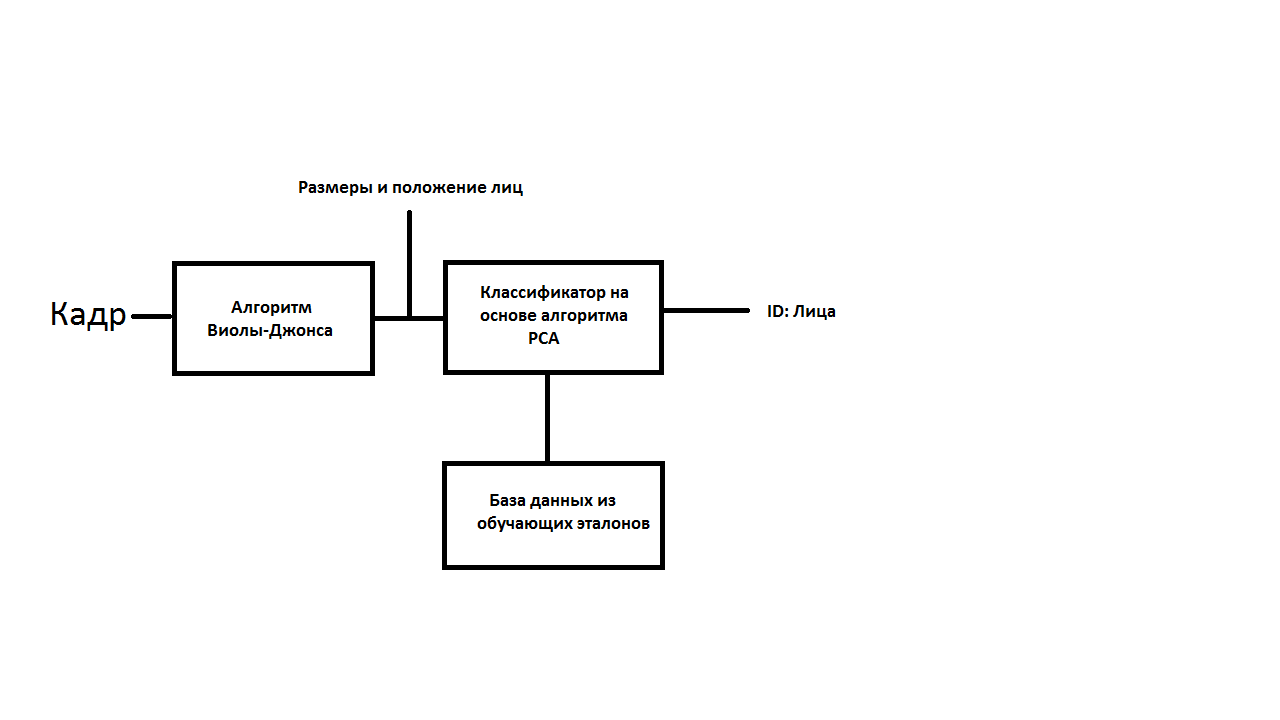


Рисунок 1.13 Структурная схема алгоритма, для реализации поставленной задачи.

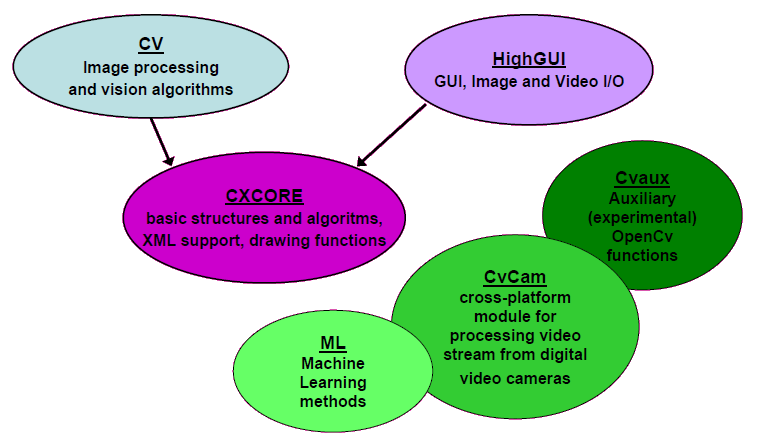
# **2. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ БИОМЕТРИЧЕСКОЙ АУТЕНТИФИКАЦИИ ПО ГЕОМЕТРИИ ЛИЦА**

## **2.1 Используемые компоненты.**

[***The ORL Database of Faces***](http://www.machinelearning.ru/wiki/index.php?title=The_ORL_Database_of_Faces) — база изображений лиц, подготовленная в научно-исследовательской лаборатории компании Olivetti. Содержит 400 фронтальных изображений 40 человек (в различных положениях, и с аксессуарами на лице).

***OpenCV* -** Open Source Computer Vision Library. Это библиотека, которая до 1-й версии разрабатывалась в Центре разработки программного обеспечения Intel (причём, российской командой в Нижнем Новгороде).  
OpenCV написана на языке высокого уровня (C/C++) и содержит алгоритмы для: интерпретации изображений, калибровки камеры по эталону, устранение оптических искажений, определение сходства, анализ перемещения объекта, определение формы объекта и слежение за объектом, 3D-реконструкция, сегментация объекта, распознавание жестов и т.д.  Фактически, OpenCV – это набор типов данных, функций и классов для обработки изображений алгоритмами компьютерного зрения.

Рисунок 2.1 Основные модули, реализованные в библиотеке OpenCV



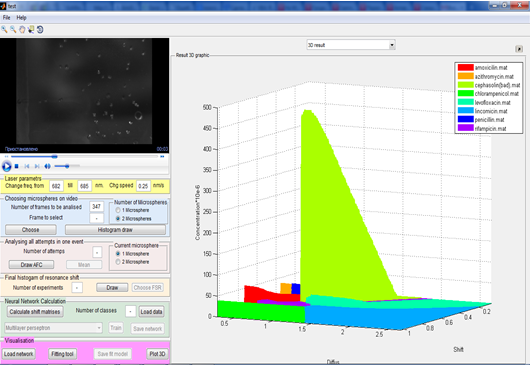
**cxcore** — ядро  
\* содержит базовые структуры данных и алгоритмы:  
— базовые операции над многомерными числовыми массивами  
— матричная алгебра, математические ф-ции, генераторы случайных чисел  
— Запись/восстановление структур данных в/из XML  
— базовые функции 2D графики  
  
**CV** — модуль обработки изображений и компьютерного зрения  
— базовые операции над изображениями (фильтрация, геометрические преобразования, преобразование цветовых пространств и т. д.)  
— анализ изображений (выбор отличительных признаков, морфология, поиск контуров, гистограммы)  
— анализ движения, слежение за объектами  
— обнаружение объектов, в частности лиц  
— калибровка камер, элементы восстановления пространственной структуры  
  
**Highgui** — модуль для ввода/вывода изображений и видео, создания пользовательского интерфейса  
— захват видео с камер и из видео файлов, чтение/запись статических изображений.  
— функции для организации простого UI (все демо приложения используют HighGUI)  
  
**Cvaux** — экспериментальные и устаревшие функции  
— пространств. зрение: стерео калибрация, само калибрация  
— поиск стерео-соответствия, клики в графах  
— нахождение и описание черт лица  
  
**CvCam** — захват видео  
— позволяет осуществлять захват видео с цифровых видео-камер

***Visual Studio Community -*** Бесплатная, полнофункциональная и расширяемая интегрированная среда разработки для создания современных приложений для Windows, Android и iOS, а также веб-приложений и облачных служб.

## **2.2 Программная реализации алгоритма идентификации личности из базы данных в видеопотоке**

Вставить объект (рисунок) и описать основные функции

Я вставила свой объект ( по аналогии введи свой и опиши)



Рассмотрим основные нюансы при реализации алгоритма идентификации личности из базы данных в видеопотоке

1. Захват видеопотока велся с web - камеры ноутбука (использовался API предоставляемый OpenCV).

2.Первый ключевой этап алгоритма локализация лица методом Виолы – Джонса велся также с помощью API предоставляемым OpenCV.

3.База данных представляет собой проиндексированный набор изображений (10 различных человек, по 6 фотографий каждого).

4. Для реализации метода главных компонент был написан собственный API (функции упаковки лиц в матрицу, приведение к нулевому среднему, 1-й дисперсии и т.д.).

5. Проверка работоспособности алгоритма велась как на обучающих изображениях, так и на дополнительных, отсутствующих в обучающем наборе.

## **2.3 Результаты использования построенного алгоритма**

Выполнена программная реализация поставленной задачи: биометрической аутентификации личности по геометрии лица, в режиме реального времени, с фиксированными освещением и фоном.

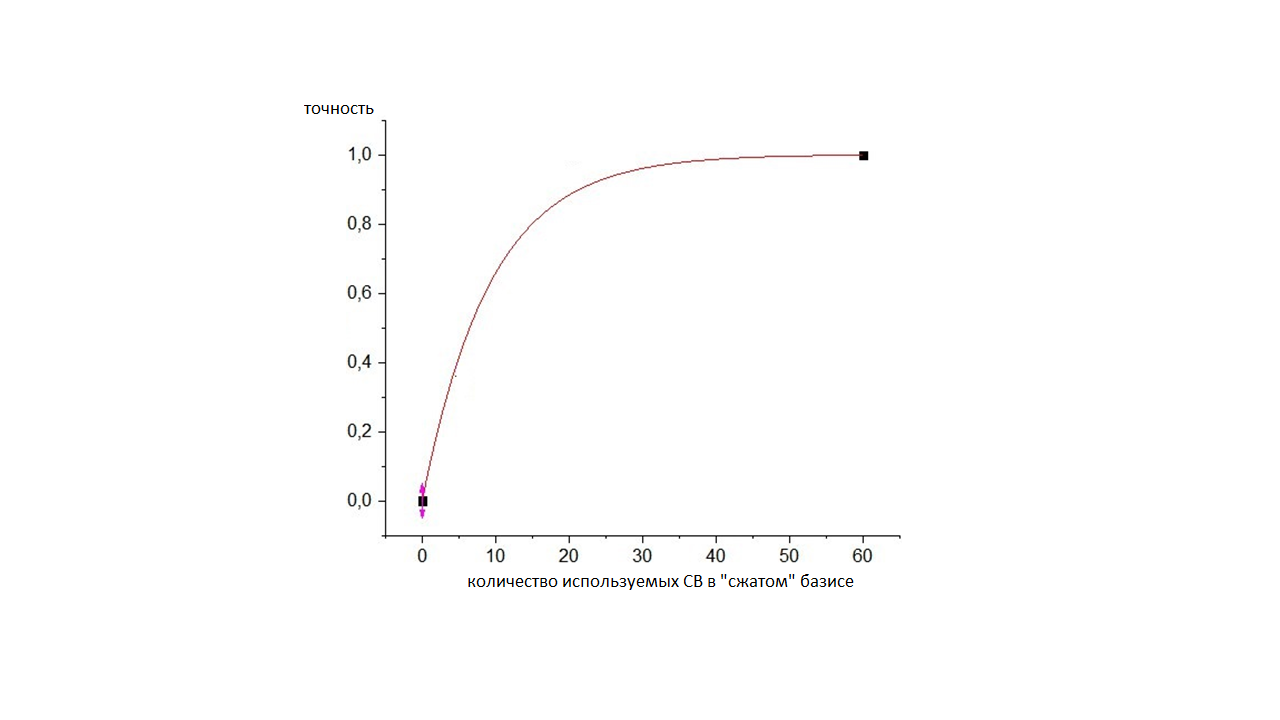


Рисунок 2.2 График зависимости точности распознавания от размерности “сжатого” базиса СВ

# **Заключение**

Выводы по результатам проделанной работы:

* Проведен анализ алгоритмов, применяющихся для распознавания лица на изображении или видеопотоке.
* Был сделан вывод, что наиболее предпочтительным алгоритмом является алгоритм Виолы Джонса в комбинации с алгоритмом главных компонент для решения задачи классификации и идентификации личности из базы данных в видеопотоке
* Точность классификации может быть улучшена за счет увеличения статистических данных.
* Выполнена программная реализация задачи идентификации личности, которая реализуется в режиме реального времени и в динамике (трекинг лица и идентификация).

# **ПРИЛОЖЕНИЕ**

## **Код программной реализации алгоритма**

/\*header files\*/

#include "opencv2/objdetect.hpp"

#include "opencv2/videoio.hpp"

#include "opencv2/highgui.hpp"

#include "opencv2/imgproc.hpp"

#include <math.h>

#include <opencv2/core/core.hpp>

#include <iostream>

#include <stdio.h>

#include <fstream>

/\*Using namespaces std and ComputerVision\*/

using namespace std;

using namespace cv;

// test\_GIT\_Work

/\*\* Global variables \*/

const int Face\_number = 60; //number of all faces in database (in this case 6 photo for each person)

const int Eigen\_number = 60; // number of eigen faces using for rebuild each face

const Size Face\_size(64, 64); // size of face-image

const String face\_cascade\_name = "haarcascade\_frontalface\_alt.xml"; // cascade for front-face detection

const String eyes\_cascade\_name = "haarcascade\_eye\_tree\_eyeglasses.xml"; //cascade for eyes detection

const String window\_name = "Capture - Face detection"; // window for image from webcam after detection

const String window\_name2 = "Face\_Classification"; // window for image from webcam after classification of detected faces

bool is\_capture = false; // true - webcam is on; false - webcam is off;

bool is\_detection = false; // true - start detection; false - stop detection;

int scale\_factor = 110; // scale for scan-window of haar-detector

int min\_neighbours = 3; // min number of positive features in one face for detection

int treshold = 50; // min distance between two vector in PCA-classificator

CascadeClassifier face\_cascade;

CascadeClassifier eyes\_cascade;

VideoCapture capture;

Mat frame;

// structure for function Haar\_Detection

struct Weights\_And\_Faces

{

Mat weights;

vector<Rect> faces;

};

/\*\* Function Headers \*/

void callbackButton\_detection(int state, void\* userdata); // button for start/stop detection

void callbackButton\_capture(int state, void\* userdata); // butto for start/stop capture

void Capture\_and\_Display(Mat average\_face, vector<Mat> eigen\_faces, Mat weight);

void Detect\_And\_Clissify(Mat average\_face, vector<Mat> eigen\_faces, Mat weight);

Mat Norm\_Dispersion(Mat average\_face, Mat matrix\_of\_faces);

Weights\_And\_Faces Haar\_Detection(Mat frame, Mat averageface, vector<Mat> eigen\_faces); // function which result is mat of weights\_of\_eigenfaces for each face and vector\_of\_position of each face on image

vector<Mat> read\_faces(void); // функция чтения лиц из корневого каталога с именами (1.jpg,2,jpg, etc) в вектор матриц

Mat Average\_face(vector<Mat> faces); // функция подсчета матрицы среднего лица в матрицу

vector<Mat> normaize\_64F\_faces(vector<Mat> faces, Mat average\_face);// функция нормализация матриц лиц(нормализ. лицо = лицо - среднее)

Mat Matrix\_of\_faces\_calculate(vector<Mat> normalize\_faces); // функция конвертации вектора лиц 64\*64 в матрицу 4096\*30 для дальнейших расчетов

vector<Mat> eigenfaces\_calculate(Mat matrix\_of\_faces, Mat eigenVec);// расчет собственных лиц размером 64\*64 и запись в вектор, используя матрицу лиц и собственные вектора матрицы ковариации

Mat weights\_of\_eigenfaces(vector<Mat> eigen\_faces, Mat matrix\_of\_faces);// подсчет матрицы весов собственных лиц для каждого лица из базы данных, Eigen\_number\*Face\_number

Mat weights\_of\_test\_face(vector<Mat> eigen\_faces, Mat matrix\_of\_faces);

/\*\* @function main \*/

int main(void)

{

namedWindow(window\_name, CV\_WINDOW\_NORMAL);

namedWindow(window\_name2, CV\_WINDOW\_NORMAL);

if (!face\_cascade.load(face\_cascade\_name)){ printf("--(!)Error loading face cascade\n"); return -1; };

if (!eyes\_cascade.load(eyes\_cascade\_name)){ printf("--(!)Error loading eyes cascade\n"); return -1; };

cvCreateButton("capture", callbackButton\_capture, NULL, CV\_PUSH\_BUTTON, 0);

cvCreateButton("detection", callbackButton\_detection, NULL, CV\_PUSH\_BUTTON, 0);

cvCreateTrackbar("scale\_factor \* 100", NULL, &scale\_factor, 200, NULL);

cvCreateTrackbar("min\_haar\_neighbours", NULL, &min\_neighbours, 10, NULL);

cvCreateTrackbar("treshold", NULL, &treshold, 200, NULL);

//LOCAL VARIABLES

Mat eigenVal; // собственные значения матрицы ковариации

Mat eigenVec(Size(Face\_number, Face\_number), CV\_64FC1); // собственные вектора матрицы ковариации

vector<Mat> normalize\_faces(Face\_number); // вектор матриц нормализованных лиц 64\*64

vector<Mat> faces(Face\_number); // вектор матри лиц из базы данных 64\*64

vector<Mat> eigen\_faces(Face\_number);// вектор матриц собственных лиц 64\*64

Mat average\_face = Mat::zeros(Face\_size, CV\_64FC1);//матрица усредненного лица 64\*64

Mat matrix\_of\_faces(Face\_size.height\*Face\_size.width, Face\_number, CV\_64FC1);// матрица лиц(располагаются по столбцам) 4096\*30

Mat matrix\_of\_faces\_transpose(Face\_number, Face\_size.height\*Face\_size.width, CV\_64FC1);// транспонированная матрица лиц 30\*4096

Mat covar\_matrix(Face\_number, Face\_number, CV\_64FC1);//марица ковариации 30\*30 (матрица\_лиц\_транспонированная \* матрицу\_лиц)

Mat weight = Mat::zeros(Eigen\_number, Face\_number, CV\_64FC1);//веса собственных лиц (вектор отдельного человека распологается в столбце, нормирован от -1 до 1)

Mat weight\_of\_test\_face = Mat::zeros(Eigen\_number, 1, CV\_64FC1);

// START PCA

faces = read\_faces();//чтение лиц из корневого каталога

average\_face = Average\_face(faces);//подсчет среднего лица

normalize\_faces = normaize\_64F\_faces(faces, average\_face);// нормализация вектора лиц

matrix\_of\_faces = Matrix\_of\_faces\_calculate(normalize\_faces);// составление матрицы лиц (каждое лицо столбец матрицы)

matrix\_of\_faces = Norm\_Dispersion(average\_face,matrix\_of\_faces);

transpose(matrix\_of\_faces, matrix\_of\_faces\_transpose);//транспонирование матрицы

covar\_matrix = matrix\_of\_faces\_transpose \* matrix\_of\_faces;// подсчет матрицы ковариации

eigen(covar\_matrix, eigenVal, eigenVec);//расчет собственных значений и собственных лиц матрицы ковариации средствами OPENCV

eigen\_faces = eigenfaces\_calculate(matrix\_of\_faces\_transpose, eigenVec); // расчет собственных лиц

weight = weights\_of\_eigenfaces(eigen\_faces, matrix\_of\_faces); // нахождение весов собственных лиц , для каждого лица из базы (веса относящиеся к одному лицу располагаются в столбце, нормированы[-1,1])

system("cls");

cout << "INFORMATION for user" << "\n" << "press ctrl + p for open Control Panel(for program control)" << "\n" << "press esc for close program " << "\n";

Capture\_and\_Display(average\_face, eigen\_faces, weight); // ctrl+p for Control Panel, esc for stop program

return 0;

}

/\* Function Definitions \*/

/////////////////////////////////////////////////////////////////////////////////

//////////// функция подсчета матрицы среднего лица в матрицу ///////////////

/////////////////////////////////////////////////////////////////////////////////

Mat Average\_face(vector<Mat> faces)

{

Mat result = Mat::zeros(Face\_size, CV\_64FC1);

for (int i = 0; i < Face\_number; i++){

add(result, faces[i], result);

}

result = result / Face\_number;

return result;

}

///////////////////////////////////////////////////////////////////////////////////////

// функция конвертации вектора лиц 64\*64 в матрицу 4096\*30 для дальнейших расчетов //

///////////////////////////////////////////////////////////////////////////////////////

Mat Matrix\_of\_faces\_calculate(vector<Mat> normalize\_faces)

{

Mat matrix\_of\_faces(Face\_size.width\*Face\_size.height, Face\_number, CV\_64FC1);

Mat temp\_mat(Face\_size.width\*Face\_size.height, 1, CV\_64FC1);

for (int i = 0; i < Face\_number; i++){

temp\_mat.data = normalize\_faces[i].data;

temp\_mat.col(0).copyTo(matrix\_of\_faces.col(i));

}

return matrix\_of\_faces;

}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// расчет собственных лиц размером 64 \* 64 и запись в вектор, используя матрицу лиц и собственные вектора матрицы ковариации //

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

vector<Mat> eigenfaces\_calculate(Mat matrix\_of\_faces, Mat eigenVec)

{

Mat temp\_eigenfaces(4096, 1, CV\_64FC1);

Mat temp\_mat(64, 64, CV\_64FC1);

vector<Mat> eigen\_faces(Face\_number);

for (int i = 0; i < Face\_number; i++) {

temp\_eigenfaces = eigenVec.row( i) \* matrix\_of\_faces;

temp\_mat.data = temp\_eigenfaces.data;

temp\_mat.copyTo(eigen\_faces[i]);

eigen\_faces[i] = eigen\_faces[i] /sqrt(norm(eigen\_faces[i], NORM\_L2));

//normalize(eigen\_faces[i], eigen\_faces[i], 0, 255, NORM\_MINMAX, CV\_64FC1);

}

return eigen\_faces;

}

///////////////////////////////////////////////////////////////////////////////////////////

// функция чтения лиц из корневого каталога с именами (1.jpg,2,jpg, etc) в вектор матриц //

///////////////////////////////////////////////////////////////////////////////////////////

vector<Mat> read\_faces(void) {

int counter = 0;

String face\_name;

Mat m1, m2;

vector<Mat> faces(Face\_number);

while (true) {

face\_name = "(" + to\_string(counter)+ ")" + ".jpg";

//face\_name = to\_string(counter + 1) + ".jpg";

if (counter >= Face\_number) { break; }

faces[counter] = imread(face\_name, 0);

equalizeHist(faces[counter], faces[counter]);

resize(faces[counter], faces[counter], Face\_size);

counter++;

}

for (int j = 0; j < Face\_number; j++) {

faces[j].convertTo(faces[j], CV\_64FC1);

}

return faces;

}

//////////////////////////////////////////////////////////////////////

// функция нормализация матриц лиц(нормализ. лицо = лицо - среднее) //

//////////////////////////////////////////////////////////////////////

vector<Mat> normaize\_64F\_faces(vector<Mat> faces, Mat average\_face) {

vector<Mat> normalize\_faces(Face\_number);

for (int j = 0; j < Face\_number; j++) {

normalize\_faces[j] = faces[j] - average\_face;

}

return normalize\_faces;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////

// подсчет матрицы весов собственных лиц для каждого лица из базы данных, Eigen\_number\*Face\_number //

//////////////////////////////////////////////////////////////////////////////////////////////////////

Mat weights\_of\_eigenfaces(vector<Mat> eigen\_faces, Mat matrix\_of\_faces){

Mat temp\_eigen = Mat::zeros(1, Face\_size.width\*Face\_size.height, CV\_64FC1);

Mat eigen\_tra = Mat::zeros(Eigen\_number, Face\_size.width\*Face\_size.height, CV\_64FC1);

Mat weight = Mat::zeros(Eigen\_number, Face\_number, CV\_64FC1);

for (int i = 0; i < Eigen\_number; i++){

temp\_eigen.data = eigen\_faces[i].data;

temp\_eigen.copyTo(eigen\_tra.row(i));

}

weight = eigen\_tra \* matrix\_of\_faces;

return weight;

}

Mat weights\_of\_test\_face(vector<Mat> eigen\_faces, Mat matrix\_of\_faces){

Mat temp\_eigen = Mat::zeros(1, Face\_size.width\*Face\_size.height, CV\_64FC1);

Mat eigen\_tra = Mat::zeros(Eigen\_number, Face\_size.width\*Face\_size.height, CV\_64FC1);

Mat weight = Mat::zeros(Eigen\_number, 1, CV\_64FC1);

for (int i = 0; i < Eigen\_number; i++){

temp\_eigen.data = eigen\_faces[i].data;

temp\_eigen.copyTo(eigen\_tra.row(i));

}

weight = eigen\_tra \* matrix\_of\_faces;

return weight;

}

/\*\* @function Haar\_Detection \*/

Weights\_And\_Faces Haar\_Detection(Mat frame, Mat averageface, vector<Mat> eigen\_faces)

{

Weights\_And\_Faces w\_a\_f;

std::vector<Rect> faces;

Mat frame\_gray,m1,m2;

Mat mat\_of\_face;

cvtColor(frame, frame\_gray, COLOR\_BGR2GRAY);

equalizeHist(frame\_gray, frame\_gray);

//-- Detect faces

double scale\_factor\_local = (double)((double)scale\_factor / 100.0);

face\_cascade.detectMultiScale(frame\_gray, faces, scale\_factor\_local, min\_neighbours, 0 | CASCADE\_SCALE\_IMAGE, Size(10, 112));

Mat weight = Mat::zeros(Eigen\_number, faces.size(), CV\_64FC1);

w\_a\_f.faces = faces;

for (size\_t i = 0; i < faces.size(); i++)

{

rectangle(frame, faces[i], Scalar(0, 255, 0), 3, 8);

Mat faceROI = frame\_gray(faces[i]);

Mat temp\_face = Mat::zeros(4096, 1, CV\_64FC1);

resize(faceROI, faceROI, Face\_size);

faceROI.convertTo(faceROI, CV\_64FC1);

faceROI = faceROI - averageface;

temp\_face.data = faceROI.data;

temp\_face.copyTo(mat\_of\_face);

weights\_of\_test\_face(eigen\_faces, mat\_of\_face).copyTo(weight.col(i));

}

weight.copyTo(w\_a\_f.weights);

imshow(window\_name, frame);

return w\_a\_f;

}

void Capture\_and\_Display(Mat average\_face, vector<Mat> eigen\_faces, Mat weight){

while (true){

switch (is\_capture){

case true:{

if (!capture.isOpened()){ capture.open(0); }

capture.read(frame);

switch (is\_detection)

{

case true:{

Detect\_And\_Clissify(average\_face, eigen\_faces, weight);

break;

}

case false:{

imshow(window\_name, frame);

imshow(window\_name2, 0);

break;

}

}

break;

}

case false:{

if (capture.isOpened()){

capture.release();

frame = 0;

putText(frame, "Webcam Off", Point(20, (int)frame.rows / 2), FONT\_HERSHEY\_DUPLEX, 3, Scalar(255, 255, 255), 2, 8);

imshow(window\_name, frame);

imshow(window\_name2, frame);

}

break;

}

}

char c = waitKey(10);

if (c == 27) { break; }

}

}

void Detect\_And\_Clissify(Mat average\_face, vector<Mat> eigen\_faces, Mat weight){

Weights\_And\_Faces w\_a\_f;

Mat face\_class;

int num2[10] = { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 };

double distance[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

double min\_distance[10] = { 10.0e+15, 10.0e+15, 10.0e+15, 10.0e+15, 10.0e+15, 10.0e+15, 10.0e+15, 10.0e+15, 10.0e+15, 10.0e+15 };

int j = 0;

w\_a\_f = Haar\_Detection(frame, average\_face, eigen\_faces);

w\_a\_f.weights.copyTo(face\_class);

for (int l = 0; l < face\_class.cols; l++){

j = 0;

for (int i = 0; i < Face\_number; i++){

if ((i % 6 == 0) && (i != 0)){ j++; };

distance[l] = norm(weight.col(i), face\_class.col(l), NORM\_L2);

if (distance[l] < min\_distance[l]){ num2[l] = j; min\_distance[l] = distance[l]; }

}

}

for (int i = 0; i < w\_a\_f.faces.size(); i++){

if ((distance[i] / 1.e+1 < treshold) && (num2[i] != 3) && (num2[i] != 0) ){

frame.convertTo(frame, CV\_8UC1);

Mat check\_good = imread("check\_good.png", 1);

cout << distance[i] / 1.e+1 << "-" << "Face from Data\_Base" << "\n";

resize(check\_good, check\_good, Size((int)w\_a\_f.faces[i].width /5, (int)w\_a\_f.faces[i].height / 5));

putText(frame, to\_string(num2[i]), Point(w\_a\_f.faces[i].x + (int)w\_a\_f.faces[i].width / 2 - 20, w\_a\_f.faces[i].y + (int)w\_a\_f.faces[i].height - 10), FONT\_HERSHEY\_SCRIPT\_SIMPLEX, 5, Scalar(0, 255, 0), 5, 8);

check\_good.copyTo(frame(Rect(w\_a\_f.faces[i].x, w\_a\_f.faces[i].y, (int)w\_a\_f.faces[i].width / 5, (int)w\_a\_f.faces[i].height / 5)));

rectangle(frame, w\_a\_f.faces[i], Scalar(0, 255, 0), 5, 8, 0);

}

else

{

Mat check\_bad = imread("check\_bad.png", 1);

frame.convertTo(frame, CV\_8UC1);

cout << distance[i] / 1.e+1 << "-" << "Unknown face" << "\n";

resize(check\_bad, check\_bad, Size((int)w\_a\_f.faces[i].width / 5, (int)w\_a\_f.faces[i].height / 5));

check\_bad.copyTo(frame(Rect(w\_a\_f.faces[i].x, w\_a\_f.faces[i].y, (int)w\_a\_f.faces[i].width / 5, (int)w\_a\_f.faces[i].height / 5)));

rectangle(frame, w\_a\_f.faces[i], Scalar(0, 0, 255), 5, 8, 0);

}

}

imshow(window\_name2, frame);

}

void callbackButton\_capture(int state, void\* userdata){

is\_capture = !is\_capture;

}

void callbackButton\_detection(int state, void\* userdata){

is\_detection = !is\_detection;

}

Mat Norm\_Dispersion(Mat average\_face,Mat matrix\_of\_faces){

Mat Average\_Column = Mat::zeros(Face\_size.width\*Face\_size.height, 1, CV\_64FC1);

Mat temp\_mat(Face\_size.width\*Face\_size.height, 1, CV\_64FC1);

Mat Dipersion\_Correct = Mat::zeros(Face\_size.width\*Face\_size.height,1, CV\_64FC1);

temp\_mat.data = average\_face.data;

temp\_mat.col(0).copyTo(Average\_Column.col(0));

for (int j = 0; j < Face\_size.width\*Face\_size.height; j++){

for(int i = 0; i < Face\_number; i++){

Dipersion\_Correct.row(j) = Dipersion\_Correct.row(j) + (matrix\_of\_faces.row(j).col(i) - Average\_Column.row(j))\*(matrix\_of\_faces.row(j).col(i) - Average\_Column.row(j));

}

}

Dipersion\_Correct = Dipersion\_Correct / (Face\_number - 1);

for (int i = 0; i < Face\_number; i++){

matrix\_of\_faces.col(i) = matrix\_of\_faces.col(i) / Dipersion\_Correct;

}

return matrix\_of\_faces;}

# **Список использованных источников**

1. https://habrahabr.ru/company/synesis/blog/238129/

2. https://habrahabr.ru/post/133826/

3. http://robocraft.ru/page/opencv/

4. <https://msdn.microsoft.com/>

5. <http://stackoverflow.com/questions/tagged/opencv>

6. <https://geektimes.ru/post/68870/>

8. https://habrahabr.ru/post/146236/

9. https://habrahabr.ru/post/208090/  
  
Список использованных источников

1. Gоrоdеtskу, М. L. Ultimate Q of optical microsphere resonators / М. L. Gоrоdеtskу, A. A. Savchenkov, V. S. Ilсhеnkо // Орtics Lеtters. – 1996. – Vol. 21, № 7. – P. 453-455.
2. Whispering-gallery-mode electro-optic modulator and photonic microwave receiver / V. S. Ilchenko [et al.] // J. Opt. Soc. Am. – 2003. –Vol.20. – P. 333–342.
3. High-Q measurements of fused-silica microspheres in the near infrared / D. W. Vеrnооу [et al.]. // Орtics Lеtters. – 1998. – Vol. 23, № 4. – P. 247-249.
4. Gоrоdеtskу, М. L. Thermal nonlinear effects in optical whispering-gallery microresonators / М. L. Gоrоdеtskу, V. S. Ilсhеnkо // Lаsеr Рhуsics. – 1992. – Vol. 2. – P. 1004-1009.
5. C-реактивный белок //Столица Медикл. Медицинские справочники. [Electronic resource]. – 2011. – Mode of access http://smed.ru/guides/66001/#article. – Date of access: 13.03. 2011.