Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра вычислительных систем
Допустить к защите
зав. кафедрой д.т.н. доцент
_____ *Мамойленко С.Н.*

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Исследование алгоритмов распознавания лиц

Пояснительная записка

Студент Кокуев Т.Н. /……/

Факультет ИВТ Группа ИВ-221

Руководитель к.т.н. доцент Молдованова О.В. / ……..……/

Новосибирск 2016 г.

Содержание

1 ВВЕДЕНИЕ

Распознавание лиц — довольно легкая задача для человека. В ходе экспериментов было показано, что дети возрастом 1-3 суток могут различать знакомые лица. Но это задача не такая простая для компьютера. Какие признаки использовать, внешние (прическа, форма головы) или внутренние (глаза, нос, рот)? Как анализировать изображение и как его кодировать?

Первый подход, который пришел на ум ученым, базируется на геометрических признаках лица. И одна из первых таких автоматизированных систем распознавания лиц была описана в [4], в которой маркированные точки (позиция глаз, ушей, носа) были использованы для построения вектора признаков (расстояние между точками, угол между ними). Само распознавание выполняло вычисления по нахождению евклидового расстояния между признак-векторами входного и эталонного изображениями. Такой метод устойчив к изменениям света, но не мог достаточно точно определять маркированные точки. Эксперименты на большом наборе данных показали, что одних геометрических признаков недостаточно для распознавания лиц.

Метод Eigenfaces, описанный в [5], использовал целостный подход к распознаванию лиц. Основным подходом этого метода является сжатие информации исходного изображения без существенных потерь информативности с помощью метода главных компонент (PCA — Principle Component Analysis).

Для того чтобы избежать многомерности, стали описывать признаки только окрестностей изображения, эти признаки должны быть более устойчивы к преградам, свету и малому размеру образца. К таким алгоритмам относятся: Gabor Wavelets, Discrete Cosinus Transform, Local Binary Patterns.

2 ПОСТАНОВКА ЗАДАЧИ

2.1 Цели

При выполнении бакалаврской работы были поставлены следующие цели:

- Протестировать алгоритмы распознавания на точность и время работы;
- Написать программу для распознавания лиц из видеопотока.

2.2 Этапы выполнения работы

- Детектирование лица;
- Отслеживание лица;
- Выравнивание лица;
- Распознавание лица.

2.3 Используемые средства для выполнения работы

2.3.1 Язык программирования Python

Причины выбора этого языка:

- Он не связан с какой-либо одной операционной системой или машиной;
- Этот язык поддерживает основные парадигмы программирования, которые нужны для выполнения данной работы (ООП, Функциональное программирование, Процедурное программирование);
- Легко читаемый код;
- Множество полезных библиотек.

2.3.2 Библиотека NumPy

Библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых и математических функций для операций с этими массивами, которые работают достаточно быстро за счет использования вставок на языках: C, C++ и Fortran.

2.3.3 Библиотека OpenCV

Библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом.

Используемые средства из этой библиотеки в данной работе:

- Методы распознавания лиц (EigenFaces, FisherFaces, LBPH);
- Алгоритм Лукаса-Канаде для отслеживания лиц;
- Алгоритм Виолы-Джонса для детектирования лиц;
- И другие разные функции для преобразования изображений.

2.3.4 Библиотека Dlib

Библиотека алгоритмов машинного обучения и разных дополнительных вспомогательных инструментов. Из этой библиотеки используется готовая обученная модель AAM (Active Appearance Model) для нахождения лицевых точек.

2.3.5 Библиотека OpenFace

Открытая библиотека для распознавания лиц, использующая глубокую сверточную нейронную сеть, которая основана на технологии FaceNet.

3 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

3.1 Метод Виолы-Джонса для обнаружения лиц

Метод Виолы-Джонса – алгоритм, позволяющий обнаруживать объекты на изображениях в реальном времени. Предложен в 2001 году Paul Viola и Michael Jones. Хотя алгоритм может распознавать различные классы изображений, основной задачей при его создании было обнаружение лиц. Обучение классификаторов идет очень медленно, но результаты поиска лица очень быстры. Также этот детектор обладает крайне низкой вероятностью ложного обнаружения лица. Алгоритм хорошо работает и распознает черты лица под небольшим углом, примерно до 30 градусов. При угле наклона больше 30 градусов процент обнаружений резко падает [11].

3.1.1 Основные принципы

- используются изображения в интегральном представлении, что позволяет вычислять быстро необходимые объекты;
- используются признаки Хаара, с помощью которых происходит поиск нужного объекта;
- используется бустинг для выбора наиболее подходящих признаков для искомого объекта на данной части изображения;
- используются каскады признаков для быстрого отбрасывания окон, где не найдено лицо.

3.1.2 Интегральное представление изображения

Для того, чтобы производить какие-либо действия с данными, используется интегральное представление изображений в методе Виолы-Джонса. Интегральное представление позволяет быстро рассчитывать суммарную яркость произвольного прямоугольника на данном изображении, причем какой бы прямоугольник не был, время расчета неизменно. Интегральное представление изображения - это матрица, совпадающая по размерам с исходным изображением. В каждом элементе ее хранится сумма интенсивностей всех пикселей, находящихся левее и выше данного элемента. Элементы матрицы рассчитываются по следующей формуле:

$$L(x,y) = \sum_{i=0,j=0}^{i \le x, j \le y} I(i,j), \tag{3.1}$$

где $I(i, j)$ — яркость пикселя исходного изображения.

Каждый элемент матрицы $L(x,y)$ представляет собой сумму пикселей в прямоугольнике от $(0,0)$ до (x,y), т.е. значение каждого пикселя (x,y) равно сумме значений всех пикселов левее и выше данного пикселя (x,y). Расчет матрицы занимает линейное время, пропорциональное числу пикселей в изображении, поэтому интегральное изображение просчитывается за один проход.

Расчет матрицы возможен по формуле:

$$L(x,y) = I(x,y) - L(x-1,y-1) + L(x,y-1) + L(x-1,y) \quad (3.2)$$

3.1.3 Признаки Хаара

Признак - отображение $f: X => D_f$, где D_f — множество допустимых значений признака. Если заданы признаки $f_1,\ldots,f_n$, то вектор признаков $x = (f_1(x),\ldots,f_n(x))$ называется признаковым описанием объекта $x \in X$ Признаковые описания допустимо отождествлять с самими объектами. При этом множество $X = D_{f1} * \ldots * D_{fn}$ называют признаковым пространством.

Признаки делятся на следующие типы в зависимости от множества D_f:

- бинарный признак, $D_f = \{0,1\}$;
- номинальный признак: D_f — конечное множество;
- порядковый признак: D_f — конечное упорядоченное множество;
- количественный признак: D_f — множество действительных чисел.

Примеры признаков Хаара показаны на рисунке 3.1.

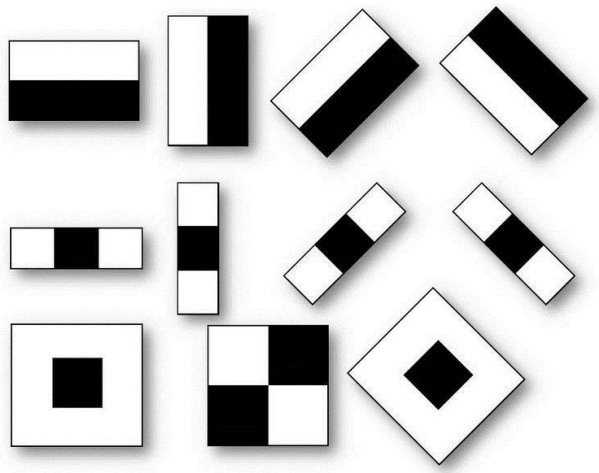

Рисунок 3.1 – Признаки Хаара

Значение признака равняется разности суммы пикселей в белом прямоугольнике и в черном прямоугольнике. Для их вычисления используется понятие интегрального изображения, рассмотренное выше. Но для задачи обнаружения лица, многие признаки бесполезны. Как же выбрать полезные признаки из количества 16000+ ? С помощью AdaBoost.

AdaBoost (сокращение от Adaptive Boosting) — алгоритм усиления классификаторов, путём объединения их в комитет, предложенный Йоавом Фройндом и Робертом Шапире. Этот алгоритм может использоваться в сочетании с несколькими алгоритмами классификации для улучшения их эффективности. AdaBoost является адаптивным в том смысле, что каждый следующий комитет классификаторов строится по объектам, неверно классифицированным предыдущими комитетами. AdaBoost чувствителен к шуму в данных и выбросам. Однако он менее подвержен переобучению по сравнению с другими алгоритмами машинного обучения. Работает Adaboost для данной задачи следующим образом. Каждый признак применяется к каждому изображению. Отбираются признаки с наименьшим количеством ошибок.

Вначале тестовым изображениям присваивается одинаковый вес, после каждой неправильной классификации вес изображения возрастает. Этот процесс происходит до тех пор, пока не достигнута требуемая точность, либо достигнут определенный коэффициент ошибок, либо отобрано определенное количество признаков.

После данной операции количество признаков значительно сократилось (с 16000 до 6000), но в фотографиях, в большинстве случаях, лицо занимает малую область, и чтобы не прогонять по всем признакам ненужные области, ввели понятие каскад классификаторов. Теперь признаки группируются по стадиям. Первые стадии содержат малое количество признаков, в каждой последующей стадии их количество будет увеличиваться. Область, которая прошла все стадии, является лицом.

3.2 Отслеживание лиц

3.2.1 Оптический поток

Оптический поток — это изображение видимого движения объектов, поверхностей или краев сцены, получаемое в результате перемещения наблюдателя (глаз или камеры) относительно сцены. Алгоритмы, основанные на оптическом потоке, — такие как регистрация движения, сегментация объектов, кодирование движений и подсчет диспаритета в стерео, — используют это движение объектов, поверхностей и краев.

Оптический поток работает на следующих предположениях:

- Интенсивности пикселей объекта не изменяются между последовательными кадрами;
- Соседние пиксели имеют аналогичное движение.

Рассмотрим пиксель *I(x, y, t)* в первом кадре, например, в следующем кадре он сдвинулся на расстояние *(dx, dy)*, и прошло время *dt*. Так как эти пиксели одинаковы и их интенсивность не поменялась, можно сказать:

$$I(x, y, t) \approx I(x + \delta x, y + \delta y, t + \delta t) \tag{3.3}$$

Считая, что перемещение мало, и используя ряд Тейлора, получаем:

$$I(x + \delta x, y + \delta y, t + \delta t) \approx I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t \tag{3.4}$$

Из этих равенств следует:

$$\begin{gathered} \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0 \\ \frac{\partial I}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial I}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial I}{\partial t}\frac{\delta t}{\delta t} = 0 \end{gathered} \tag{3.5}$$

Отсюда получается, что:

$$\frac{\partial \mathrm{I}}{\partial \mathrm{x}} \mathrm{V_x} + \frac{\partial \mathrm{I}}{\partial \mathrm{y}} \mathrm{V_y} + \frac{\partial \mathrm{I}}{\partial \mathrm{t}} = 0\,, \tag{3.6}$$

где V_x , V_y – компоненты скорости оптического потока в $I(x,y,t)$.

В уравнении (3.6), V_x , V_y являются неизвестными, то есть у нас получается одно уравнение с двумя неизвестными, и его нельзя однозначно решить. Эта проблема разрешается с помощью алгоритмов определения оптического потока, в данной работе используется алгоритм Лукаса-Канаде.

3.2.2 Алгоритм Лукаса-Канаде

Используя предположение, что соседние пиксели будут иметь аналогичное движение, берется область 3x3 вокруг точки, таким образом мы считаем, что точки в этой области имеют такое же движение, как у центровой. Таким образом, получается 9 уравнений с 2 неизвестными:

$$\begin{cases} I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1) \\ \dots \\ I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n) \end{cases}, \tag{3.7}$$

где $q_1, \dots, q_n$– пиксели внутри области, $I_x(q_i), I_y(q_i), I_t(q_i) -$ частные производные изображения I по координатам x, y и времени t, вычисленные в точке q_i.

Эта система уравнений может быть записана в матричном виде:

$$Av = b, \tag{3.8}$$

где

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad b = \begin{bmatrix} -I_t(q_1) \\ \vdots \\ -I_t(q_n) \end{bmatrix}.$$

Полученную систему решаем с помощью метода наименьших квадратов. Таким образом, получается система уравнений 2x2:

$$\begin{cases} A^T A v = A^T b \\ v = (A^T A)^{-1} A^T b \end{cases} \tag{3.9}$$

И в итоге получаем:

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum I_x(q_i)^2 & \sum I_x(q_i)\, I_y(q_i) \\ \sum I_x(q_i)\, I_y(q_i) & \sum I_x(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum I_x(q_i)\, I_t(q_i) \\ -\sum I_y(q_i)\, I_t(q_i) \end{bmatrix}. \tag{3.10}$$

3.3 Распознавание лиц

3.3.1 Метод EigenFaces

Основным подходом этого метода является сжатие информации исходного изображения без существенных потерь информативности с помощью метода главных компонент (PCA — Principle Component Analysis) [8].

Шаги алгоритма PCA в Eigenfaces:

1. Дана выборка: $X = \{x_1, x_2, \dots, x_n\}$, где x_i – матрицы лиц, преобразованные в вектора, которые есть столбцы общей матрицы X (рисунок 3.2).

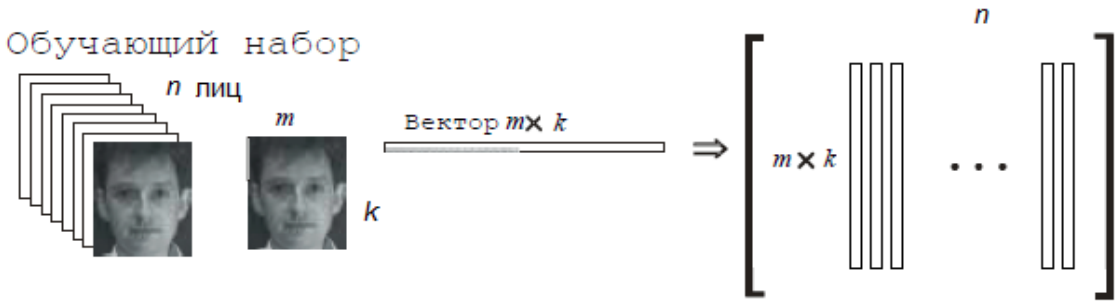

Рисунок 3.2 – Выборка лиц

2. Вычисляем математическое ожидание:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.11}$$

Если вывести математическое ожидание в формате изображения, то мы получим примерно такую картину (рисунок 3.3):

Рисунок 3.3 – Пример среднего лица

3. Вычисляем матрицу ковариации:

$$S = \frac{1}{n}\sum_{i=1}^{n} (x_i - m_x)\,(x_i - m_x)^T. \tag{3.12}$$

На этом шаге из каждого изображения вычитается среднее, это делается для того, чтобы оставить только уникальную информацию лица, если вывести каждый столбец в виде изображения, то получится примерно такая картина (рисунок 3.4):

Рисунок 3.4 – Эталонные изображения и их отклонение от среднего

4. Найдем собственные значения λ и собственные вектора v:

$$Sv_i = \lambda_i v_i, i = 1 \ldots n. \quad (3.13)$$

5. Отбираем k собственных векторов с наибольшим собственным значением λ.
6. Перестраиваем базис, тем самым получаем новое пространство:

$$\begin{cases} y = W^T(x - m_x) \\ x = Wy + m_x \end{cases} \quad (3.14)$$

где $W = \{v_1, \ldots, v_n\}$, x – новое сокращенное пространство.

Таким образом, данный метод выполняет распознавание лиц с помощью:

- Проектирования всех тренировочных образцов внутри подпространства PCA.
- Проектирования входного изображения внутри пространства PCA.
- Поиска ближайшего соседа между тренировочными образцами и входным изображением.

Есть еще одна проблема. Например, если дано 400 изображений размером 100 x 100. С помощью PCA мы получим матрицу ковариации $S = XX^T$, где размер матрицы X равен 400 x 10000, а размер матрицы S – 10000 x 10000, это приблизительно 0.8 GB. Данная проблема решается с помощью линейной алгебры, в которой есть правило: матрица M x N, при $M > N$, может иметь только $N - 1$ ненулевых собственных значений. Так что можно взять разложение собственных значений $S = X^TX$ с размером N x N вместо:

$$X^TXv_i = \lambda_i vi. \quad (3.15)$$

И получить исходные собственные вектора:

$$XX^T(Xv_i) = \lambda_i(Xv_i). \quad (3.16)$$

В итоге после процедуры распознавания, мы получаем расстояние и индекс объекта из базы, к которому входное изображение расположено ближе всего.

Недостатки:

- Чувствительность к смене ракурса и освещения.
- Обучение происходит единожды, при добавлении новых лиц в базу, придется строить пространство из собственных векторов заново.

Достоинства:

- Достаточно быстрое распознавание.
- Упрощенная база лиц занимает мало места.

3.3.2 Метод FisherFaces

Метод главных компонент (PCA), который является основой метода Eigenfaces, находит линейную комбинацию признаков, которая максимизирует общую дисперсию данных. Это довольно мощный способ представления данных, но он не учитывает классы объектов, и может быть потеряно много отличительной информации из-за отбрасывания компонент, которые не вошли в базис. Например, если в базе есть фотографии с различным освещением, и поскольку известно, что компоненты, определенные с помощью PCA, не всегда содержат в себе всю отличительную информацию, поэтому образцы разных классов смешиваются друг с другом, и дальнейшая классификация становится невозможной. Эту проблему решает алгоритм LDA (Линейный дискриминантный анализ), который является основой метода FisherFaces [8].

LDA – метод статистики и машинного обучения, применяемый для нахождения линейных комбинаций признаков, наилучшим образом разделяющих два или более класса объектов или событий (т.е. объекты одного и того же класса должны находиться, как можно ближе друг к другу в пространстве, и при этом происходит максимизация расстояния между классами). Полученная комбинация может быть использована в качестве линейного классификатора или для сокращения размерности пространства признаков перед последующей классификацией.

Описание алгоритма:

1. Дана выборка: $X = \{x_1, x_2, \dots, x_n\}$, где x_i – матрицы лиц, преобразованные в вектора, которые есть столбцы общей матрицы X.
2. Вычисляем общее математическое ожидание формуле:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \quad . \tag{3.17}$$

И математическое ожидание для каждого класса:

$$\mu_i = \frac{1}{|X_i|} \sum_{x_j \in X_i} x_j \tag{3.18}$$

3. Вычисляем матрицу межклассового разброса:

$$S_B = \sum_{i=i}^{c} N_i \, (\mu_i - \mu)(\mu_i - \mu)^T \tag{3.19}$$

И матрицу внутриклассового разброса:

$$S_w = \sum_{i=i}^{c} \sum_{x_j \in X_i} (x_j - \mu_i)(x_j - \mu_j)^T \quad (3.20)$$

где c – количество классов в обучаемой базе.

4. Получаем итоговый базис:

$$W_{opt} = argmax_W \frac{|W^T S_B W|}{|W^T S_W W|}. \quad (3.21)$$

Данный алгоритм лучше распознает при разных освещениях, чем EigenFaces, только обучение должно проходить также при разных освещениях.

На рисунке 3.5 изображены результаты работы с базой лиц AT&T.

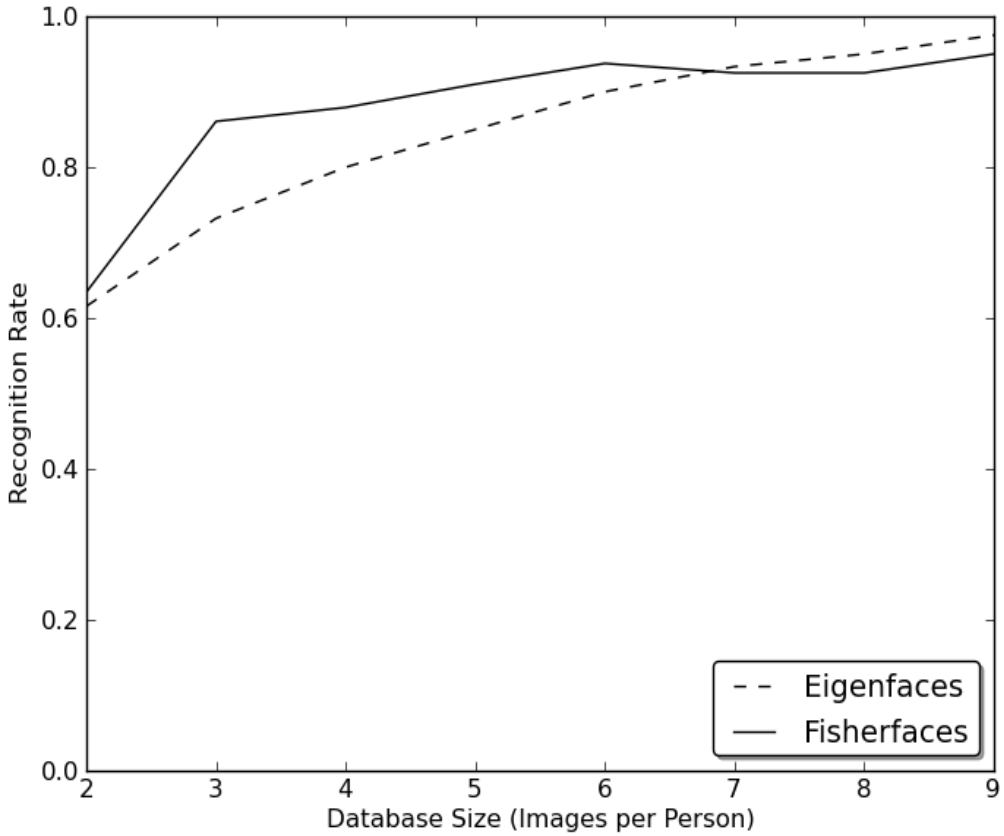

Рисунок 3.5 – График зависимости точности распознавания от количества фотографий для каждого человека в базе

База AT&T является одной из самых простых, в ней соблюдены идеальные условия: хорошее освещение, достаточное количество фотографий для каждой личности и т.д.

3.3.3 Метод LBPH

Ученые решили попробовать извлекать признаки из окрестностей изображения, этот набор признаков имеет маломерную структуру, что является положительной чертой данного метода. Также, поскольку признаки извлекаются из

окрестностей, это позволяет быть алгоритму устойчивым к масштабу, поворотам и т.д.

Описание работы метода LBPH: сначала изображение делится на одинаковые блоки, которые образуют сетку (рисунок 3.7). Далее для каждого блока строится гистограмма кодов, которые вычисляются таким образом: берется пиксель, который сравнивается с соседями, если интенсивность центрального пикселя больше или равна интенсивности соседа, то он обозначается 1, иначе — 0. В итоге каждому пикселю будет соответствовать двоичное число, состоящее из результатов сравнений. Эти преобразования наглядно показаны на рисунке 3.6. Полученные гистограммы объединяются в одну общую, которая является итоговым дескриптором, который используется для классификации лица [8, 7].

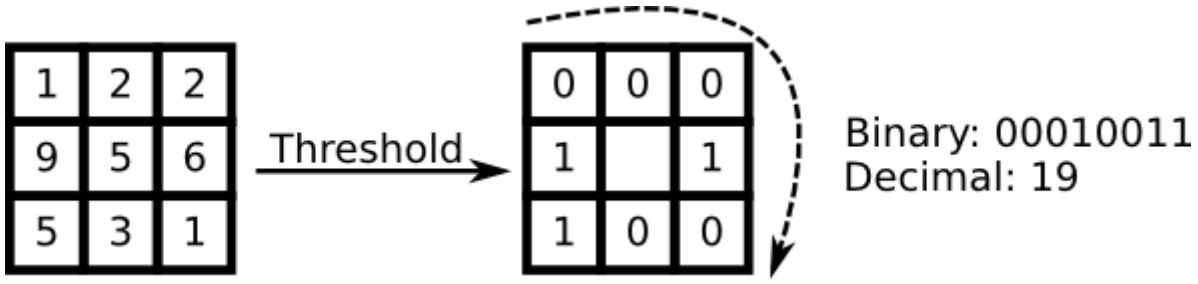

Рисунок 3.6 – Преобразование области в двоичный код

Полная схема работы изображена на рисунке 3.7:

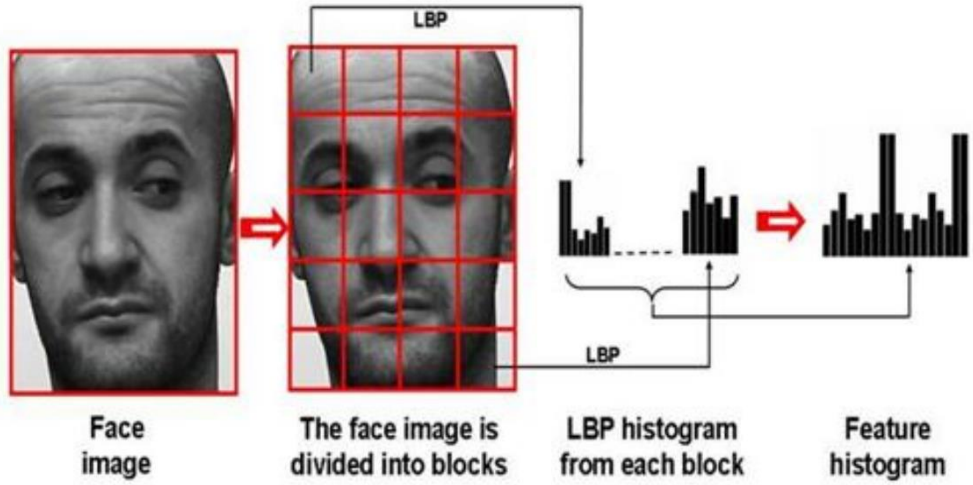

Рисунок 3.7 – Схема работы алгоритма LBPH

Описанный подход позволяет захватывать очень мелкозернистые детали, позже оказалось, что LBPH не мог кодировать детали разного масштаба, из-за этого он был расширен, и теперь число соседей может варьироваться. Идея расширения заключалась в том, чтобы расположить соседей по кругу определенного радиуса, и, таким образом, выделялись следующие признаки окрестностей:

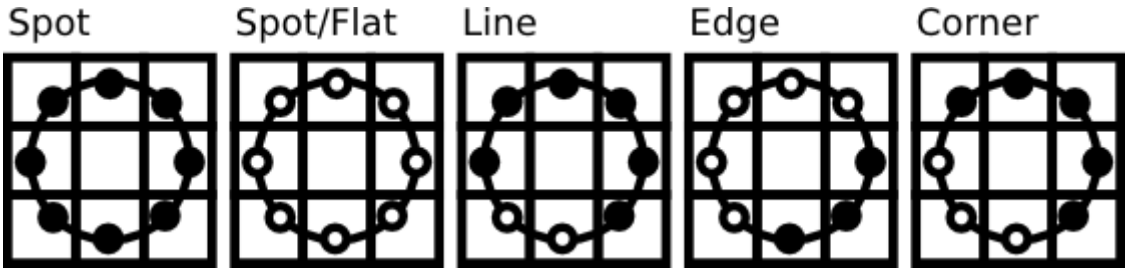

Рисунок 3.8 – Признаки круговой окрестности

Формулы для расчета координат соседей:

$$x_p = x_c + R\cos\left(\frac{2\pi p}{P}\right)$$
$$y_p = y_c + R\sin\left(\frac{2\pi p}{P}\right), \quad (3.22)$$

где x_p , y_p — координаты соседа; x_c , y_c — координаты центра; R — радиус окружности; p — номер соседа; P — количество соседей.

Если координаты точек не соответствуют координатам изображения, к ним будет применен метод интерполяции.

3.3.4 FaceNet

Технология Google, опубликованная в 2015 году, разработанная Флорианом Шрофом, Дмитрием Калиниченко и Джеймсом Филибином [6].

Для распознавания используется обученная глубокая сверточная нейронная сеть, которая возвращает 128-размерный вектор признаков, отлично классифицирующийся. Евклидово расстояние используется для метрики похожести лиц.

На рисунке 3.9 цифрами обозначается евклидово расстояние между векторами признаков лиц на изображениях.

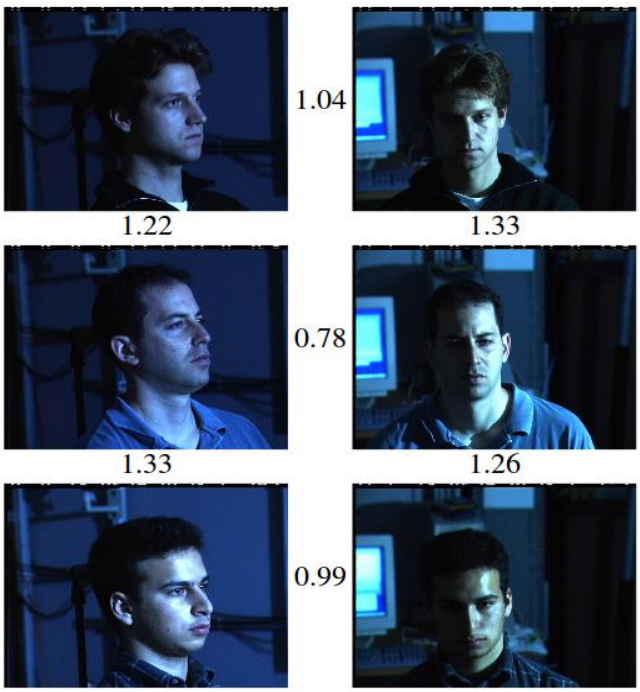

Рисунок 3.9 – Примеры лиц в разном положении и освещении.

На базе Labeled Faces in the Wild (LFW) метод показал точность 99,63%, а на базе YouTube Faces DB – 95,12%.

3.3.4.1 Triplet Loss

Основная задача данного метода – уместить лица в пространстве размерности R^d, в котором будет минимальное расстояние между лицами одной личности и максимальное – между лицами из разных классов (рисунок 3.10).

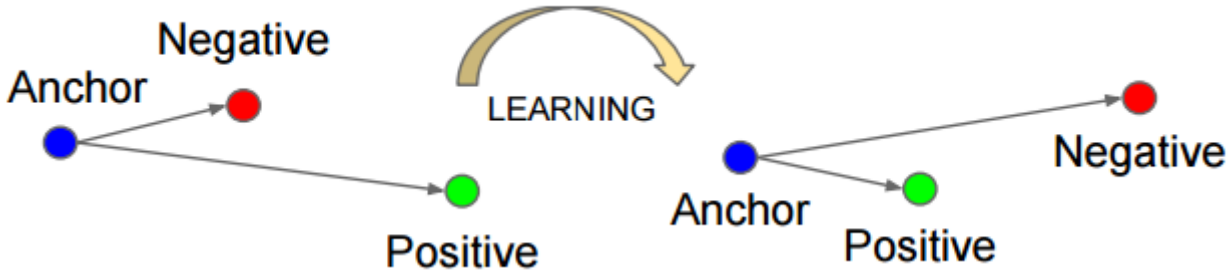

Рисунок 3.10 – Работа Triplet Loss

Вычисляется это с помощью формулы:

$$\left\|f(x_i^a) - f(x_i^p)\right\|_2^2 + \propto < \|f(x_i^a) - f(x_i^n)\|_2^2, \forall (f(x_i^a), f(x_i^p), f(x_i^n)) \in T, \quad (3.23)$$

где x_i^a– лицо определенной персоны i; x_i^p – лица той же личности (положительный исход); x_i^n – лица любой другой личности (негативный исход); $\propto$ – грань между положительными и негативными парами; T – множество всех возможных триплетов в обучающем множестве.

Вложения представлены в виде $f(x) \in R^d$. Изображение x встраивается внутрь d-мерного Евклидова пространства. Кроме того, это вложение непосредственно ограничивается d-мерной гиперсферой, т.е. $\|f(x)\|_2 = 1$. Сами ошибки минимизируются с помощью:

$$L = \sum_i^N \left[\left\|f(x_i^a) - f(x_i^p)\right\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \propto \right]. \quad (3.24)$$

Эти триплеты не должны способствовать обучению, что может привести к медленной сходимости результатов, так как они должны проходить через сеть. Крайне важно выбрать триплеты, которые могут способствовать улучшению модели. В следующем разделе будут рассмотрены различные подходы выбора триплетов.

3.3.4.2 Выбор триплетов

Для того чтобы обеспечить быструю сходимость, нужно выбрать такие триплеты, которые нарушают ограничения в формуле (3.23). Это означает, что при x_i^a должны быть выбраны x_i^p такие, что $argmax_{x_i^p} \left\|f(x_i^a) - f(x_i^p)\right\|_2^2$, и x_i^n – такие, что $argmin_{x_i^n} \|f(x_i^a) - f(x_i^n)\|_2^2$.

Невозможно вычислить *argmin* и *argmax* для всей обучающей выборки. Также это может привести к неправильной подготовке, так как лица могут быть неправильно размечены и в плохом качестве, из-за этого будут доминировать жесткие позитивные и негативные исходы. Есть два способа, чтобы избежать эту проблему:

- Генерировать триплеты каждые n шагов, используя недавнюю контрольную точку сети и вычисляя *argmin* и *argmax* на подмножестве данных;
- Генерировать триплеты в режиме реального времени. Это может быть сделано путем выбора жестких позитивных и негативных экземпляров в пределах мини-выборки.

В данном методе, ориентир идет на генерацию в режиме реального времени и используются большие мини-выборки порядка несколько тысяч экземпляров, и вычисляются только *argmin* и *argmax* в мини-выборках.

Для того чтобы иметь хорошие положительные дистанции, в мини-выборке должно быть минимальное количество экземпляров какой-либо личности. В одной мини-выборке около 40 лиц на каждую личность. Также в каждую мини-выборку добавлены лица, представляющие негативный исход.

3.3.4.3 Глубокая сверточная сеть

CNN обучается с помощью SGD (Stochastic Gradient Descent) и AdaGrad. Обучение начинается с коэффициента 0.05, который будет постепенно уменьшаться для завершения модели. Модели моделируются случайным образом. Обучение на кластере составляет от 1000 до 2000 часов. Увеличение точности замедляется после 500 часов обучения, но дополнительное обучение все еще может повысить эффективность.

3.4 Метод опорных векторов

Метод опорных векторов (англ. SVM, support vector machine) — набор схожих алгоритмов обучения с учителем, использующихся для задач классификации и регрессионного анализа. Принадлежит к семейству линейных классификаторов, может также рассматриваться как специальный случай регуляризации по Тихонову. Особым свойством метода опорных векторов является непрерывное уменьшение эмпирической ошибки классификации и увеличение зазора, поэтому метод также известен как метод классификатора с максимальным зазором [12].

Основная идея метода — перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей наши классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

3.4.1 Формальное описание задачи

Мы полагаем, что точки имеют вид:

$$\{(x_1, c_1), \dots, (x_n, c_n), \tag{3.25}$$

где c_i – принимает значение 1 или -1, в зависимости от того, какому классу принадлежит точка x_i; x_i – это p-мерный вещественный вектор, обычно нормализованный значениями [0,1] или [-1,1].

Если точки не будут нормализованы, то точка с большими отклонениями от средних значений координат точек слишком сильно повлияет на классификатор. Мы можем рассматривать это как учебную коллекцию, в которой для каждого элемента уже задан класс, к которому он принадлежит. Мы хотим, чтобы алгоритм метода опорных векторов классифицировал их таким же образом.

Для этого мы строим разделяющую гиперплоскость, которая имеет вид:

$$w * x - b = 0, \tag{3.26}$$

где w – перпендикуляр к разделяющей гиперплоскости; b – скалярный порог.

Параметр $\frac{b}{\|w\|}$ равен по модулю расстоянию от гиперплоскости до начала координат. Если параметр b равен нулю, гиперплоскость проходит через начало координат, что ограничивает решение.

Так как нам необходимо оптимальное разделение, нас интересуют опорные вектора и гиперплоскости, параллельные оптимальной и ближайшие к опорным векторам двух классов. Можно показать, что эти параллельные гиперплоскости могут быть описаны следующими уравнениями (с точностью до нормировки).

$$\begin{aligned} w * x - b &= 1 \\ w * x - b &= -1. \end{aligned} \tag{3.27}$$

Если обучающая выборка линейно разделима, то мы можем выбрать гиперплоскости таким образом, чтобы между ними не лежала ни одна точка обучающей выборки и затем максимизировать расстояние между гиперплоскостями. Ширину полосы между ними легко найти из соображений геометрии, она равна $\frac{2}{\|w\|}$, таким образом, наша задача минимизировать $\|w\|$. Чтобы исключить все точки из полосы, мы должны убедиться для всех i, что:

$$\begin{bmatrix} w * x_i - b \geq 1, & c_i = 1 \\ w * x_i - b \geq -1, & c_i = -1 \end{bmatrix}. \tag{3.28}$$

3.4.2 Линейно разделимая выборка

Проблема построения оптимальной разделяющей гиперплоскости сводится к минимизации $\|w\|$, при условии в формуле (3.27). Это задача квадратичной оптимизации, которая имеет вид:

$$\begin{cases} \|w\|^2 \to min \\ c_i(w * x_i - b) \geq 1, \quad 1 \leq i \leq n \end{cases} \tag{3.29}$$

По теореме Куна-Таккера эта задача эквивалентна двойственной задаче поиска седловой точки функции Лагранжа.

$$\begin{cases} L(w, b; \lambda) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \lambda_i \big(c_i\big((w * x_i) - b\big) - 1\big) \to \min_{w,b}\max_{\lambda} \\ \lambda_i \geq 0, \;\; 1 \leq i \leq n \end{cases}, \tag{3.30}$$

где $\lambda = (\lambda_1, \dots, \lambda_n)$ – вектор двойственных переменных.

Сведем эту задачу к эквивалентной задаче квадратичного программирования, содержащей только двойственные переменные:

$$\begin{cases} -L(\lambda) = -\sum_{i=1}^{n} \lambda_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j c_i c_j \, (x_i * x_j) \to \min_{\lambda} \\ \lambda_i \geq 0, \;\; 1 \leq i \leq n \\ \sum_{i=1}^{n} \lambda_i c_i = 0 \end{cases}. \tag{3.31}$$

Допустим мы решили данную задачу, тогда w и b можно найти по формулам:

$$\begin{aligned} w &= \sum_{i=1}^{n} \lambda_i c_i \, x_i \\ b &= w * x_i - c_i, \;\; \lambda_i > 0. \end{aligned} \tag{3.32}$$

В итоге алгоритм классификации может быть записан в виде:

$$a(x) = sign\left(\sum_{i=1}^{n} \lambda_i c_i \, x_i * x - b\right). \tag{3.33}$$

При этом суммирование идет не по всей выборке, а только по опорным векторам, для которых $\lambda_i \neq 0$.

3.4.3 Случай линейной неразделимости

Рассмотрим теперь общий случай, когда выборку невозможно идеально разделить гиперплоскостью. Это означает, что какие бы w и b мы не взяли, хотя бы одно из ограничений будет нарушено:

$$\exists x_i \in X^l : c_i\big((w_i * x_i) + b\big) < 1. \tag{3.34}$$

Сделаем эти ограничения «мягкими», введя штраф $\xi_i \geq 0$ за их нарушение:

$$c_i\big((w_i * x_i) + b\big) \geq 1 - \xi_i, \;\; i = 1, \dots, l. \tag{3.35}$$

Отметим, что если отступ объекта лежит между нулем и единицей ($0 \leq c_i\big((w_i * x_i) + b\big) < 1$), то объект верно классифицируется, но имеет ненулевой штраф $\xi > 0$. Таким образом, мы штрафуем объекты за попадание внутрь разделяющей полосы.

Величина $\frac{1}{\|w\|}$ в данном случае называется мягким отступом (soft margin). С одной стороны, мы хотим максимизировать отступ, с другой — минимизировать штраф за неидеальное разделение выборки $\sum_{i=1}^{l} \xi_i$. Эти две задачи противоречат друг другу: как правило, излишняя подгонка под выборку приводит к маленькому отступу, и наоборот — максимизация отступа приводит к большой ошибке на обучении. В качестве компромисса будем минимизировать взвешенную сумму двух указанных величин.

Приходим к оптимизационной задаче:

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \xi_i \to \min\limits_{w,b,\xi} \\ c_i\big((w_i * x_i) + b\big) \geq 1 - \xi_i, \quad i = 1, \dots, l \cdot \\ \xi_i \geq 0, \quad i = 1, \dots, l \end{cases} \tag{3.36}$$

Чем больше здесь параметр *C*, тем сильнее мы будем настраиваться на обучающую выборку. Данная задача также является выпуклой и имеет единственное решение.

3.4.4 Многоклассовый SVM

Классификаторы строятся одновременно, в рамках одной оптимизационной задачи. Для простоты будем считать, что в выборке имеется константный признак, и не будет явно указываться сдвиг *b*. Будем настраивать *K* наборов параметров $w_1, \dots, w_k$, и итоговый алгоритм определим как:

$$a(x) = \underset{k \in \{1 \dots K\}}{\operatorname{argmax}} \langle w_k, x \rangle. \tag{3.37}$$

Рассмотрим следующую функцию потерь:

$$\max_k \{\langle w_k, x \rangle + 1 - [k = y(x)]\} - \langle w_{y(x)}, x \rangle. \tag{3.38}$$

Разберемся сначала с выражением, по которому берется максимум. Если $k = y(x)$, то оно равно $\langle w_k, x \rangle$; в противном же случае оно равно $\langle w_k, x \rangle + 1$. Если оценка за верный класс больше оценок за остальные классы хотя бы на единицу, то максимум будет достигаться на $k = y(x)$; в этом случае потеря будет равна нулю. Иначе же потеря будет больше нуля. Здесь можно увидеть некоторую аналогию с бинарным SVM: мы штрафуем не только за неверный ответ на объекте, но и за неуверенную классификацию (за попадание объекта в разделяющую полосу).

Рассмотрим сначала линейно разделимую выборку — т.е. такую, что существуют веса $w_1, \dots, w_k$, при которых потеря (3.38) равна нулю. В бинарном SVM мы строили классификатор с максимальным отступом. Известно, что аналогом отступа для многоклассового случая является норма Фробениуса матрицы *W*, *k*-я строка которой совпадает с w_k:

$$p = \frac{1}{\|W\|^2} = \frac{1}{\sum_{k=1}^{K}\sum_{j=1}^{d} w_{kj}^2}. \quad (3.39)$$

Получаем следующую задачу:

$$\begin{cases} \dfrac{1}{\|W\|^2} \to \min\limits_{w} \\ \langle w_{y_i}, x\rangle + [k = y_i] - \langle w_k, x_i\rangle \geq 1, \quad i = 1 \dots, l; k = 1 \dots, K \end{cases}. \quad (3.40)$$

Перейдем теперь к общему случаю. Как и в бинарном методе опорных векторов, перейдем к мягкой функции потерь, введя штрафы за неверную или неуверенную классификацию.

Получим задачу:

$$\begin{cases} \dfrac{1}{2}\|W\|^2 + C\sum\limits_{i=1}^{l}\xi_i \to \min\limits_{w,\xi} \\ \langle w_{y_i}, x\rangle + [k = y_i] - \langle w_k, x_i\rangle \geq 1 - \xi_i, \quad i = 1 \dots, l; k = 1 \dots, K \\ \xi_i \geq 0, \quad i = 1, \dots, l \end{cases}. \quad (3.41)$$

Отметим, что такой подход решает проблему с несоизмеримостью величин, выдаваемых отдельными классификаторами: классификаторы настраиваются одновременно, и выдаваемые ими оценки должны правильно соотноситься друг с другом, чтобы удовлетворять ограничениям.

4 РАСПОЗНАВАНИЕ ЛИЦ В ВИДЕОПОТОКЕ

Была написана программа, которая распознает лица в видеопотоке с использованием средств, которые перечислены и описаны в разделе 2.3. Также программа может создавать частично индексированную базу лиц из видеопотока.

4.1 Опции программы

- -v – путь к видеофайлу, на котором будет происходить процесс распознавания, если опция не задана, в качестве источника будет использоваться веб-камера;
- -p – путь, по которому будут сохраняться распознанные лица, в каждой папке будут лежать вырезанные лица определенного человека;
- -a – название алгоритма для распознавания, можно использовать только методы, которые описаны в разделе 3.3;
- -t – эта опция используется, если есть уже натренированная модель для распознавания (указывается путь к этой модели). Если опция не задана, то программа будет динамически распознавать и обучаться, в таком режиме можно использовать только методы LBPH и FaceNet;
- --notracking – если задана эта опция, программа будет распознавать лица с определенным шагом. Если не задана, то программа не будет делать снимки лиц для распознавания, пока лица отслеживаются.

4.2 Этапы распознавания

4.2.1 Обнаружение лица

Для обнаружения лица используется следующая функция (листинг 4.1).

Листинг 4.1 – Функция обнаружения лица

```
def detect(img, cascade):
    rects = cascade.detectMultiScale(img,scaleFactor=1.3,
          minNeighbors=4, minSize=(60, 60),
               flags=cv2.CASCADE_SCALE_IMAGE)
    if len(rects) == 0:
        return []
    rects[:,2:] += rects[:,:2]
    return rects
```

На вход функции принимаются параметры:

- img – входное изображение;
- cascade – могут быть использованы разные каскады: LBP каскад, каскад Хаара (используется в данной программе).

Функция detectMultiScale(), из библиотеки OpenCV, обнаруживает лица на изображении и имеет следующие входные параметры:

- img – изображение, в котором будет совершаться поиск лица;
- scaleFactor – шаг, который обозначает, во сколько раз область искомого лица будет увеличиваться. Чем больше шаг, тем быстрее будет совершаться поиск, но также увеличивается риск пропустить область с лицом;

- minNeighbors – этот параметр влияет на качество обнаруженного лица, чем выше – тем лучше качество. Для хорошего детектирования лица можно варьировать этот показатель примерно в пределах 3 – 6;
- minSize – минимальная искомая область. Области меньшим размером будут игнорироваться. С увеличением параметра увеличивается скорость работы метода, соответственно с уменьшением – наоборот.

Эта функция возвращает массив векторов, вектор имеет формат (x, y, w, h), где:

- x, y – координаты начальной точки прямоугольника (в нем находится обнаруженное лицо);
- w – ширина прямоугольника;
- h – высота прямоугольника.

4.2.2 Отслеживание лица

Поскольку функция detectMultiScale() возвращает прямоугольник, в котором помимо лица есть ненужные точки, а нам для отслеживания нужны именно точки лица, то для решения этой проблемы используется метод AAM (Активные модели внешнего вида – Active Appearance Models). Уже обученная AAM накладывает маску на область с лицом и подгоняет её под лицо. Данные действия выполняет функция predictor() из листинга 4.2. Обученная модель AAM и функция predictor() взяты из библиотеки dlib.

Листинг 4.2 – Обнаружение лицевых точек

```
shape = self.predictor(frame, nrect)
for point in shape.parts():
     cv2.circle(self.vis, (point.x, point.y),
                    2, (0, 255, 0), -1)
     self.tracks[-1].append((point.x,
                                point.y))
```

Функция predictor() (из библиотеки dlib) возвращает массив с координатами лицевых точек. Далее в цикле эти точки отрисовываются и добавляются в контейнер, где хранятся все лицевые точки отслеживаемых лиц.

В листинге 4.3 происходит вычисление местоположения отслеживаемых точек.

Листинг 4.3 – Вычисление координат отслеживаемых точек в текущем кадре

```
p0 = np.float32(self.tracks[i]).reshape(-1, 2)
p1, st, err = cv2.calcOpticalFlowPyrLK(img0,
                                img1, p0, None,
                                **lk_params)
p0r, st, err = cv2.calcOpticalFlowPyrLK(img1,
                                img0, p1, None,
                                **lk_params)
d = abs(p0r - p0).reshape(-1, 2).max(-1)
good = d < 1
```

Используя координаты этих точек в предыдущем кадре видео (также используется сам кадр). Далее, если разница между точкой в предыдущем и текущем кадре превышает определенную границу, то эта точка считается потерянной и далее не отслеживается.

В листинге 4.4 сначала происходит отсеивание лиц, у которых количество точек отслеживания меньше определенной границы. В копии кадра, который предназначен детектору лиц, вокруг отслеживаемых точек рисуется белый прямоугольник, для того чтобы детектор рекурсивно не захватил уже отслеживаемые точки (рисунок 3.9).

Листинг 4.4 – Отрисовка белого прямоугольника вокруг отслеживаемых точек.

```
if len(self.tracks[i]) > self.min_track_len:
     points = np.reshape(self.tracks[i], (-1, 2))
     x, y, w, h = cv2.boundingRect(points)
     cv2.rectangle(self.vis, (x, y), (x + w, y + h),
                   color=(0, 0, 0))
     cv2.rectangle(frame, (x, y), (x + w, y + h),
                   color=255, thickness=-1)
else:
     list_pop.append(i)
```

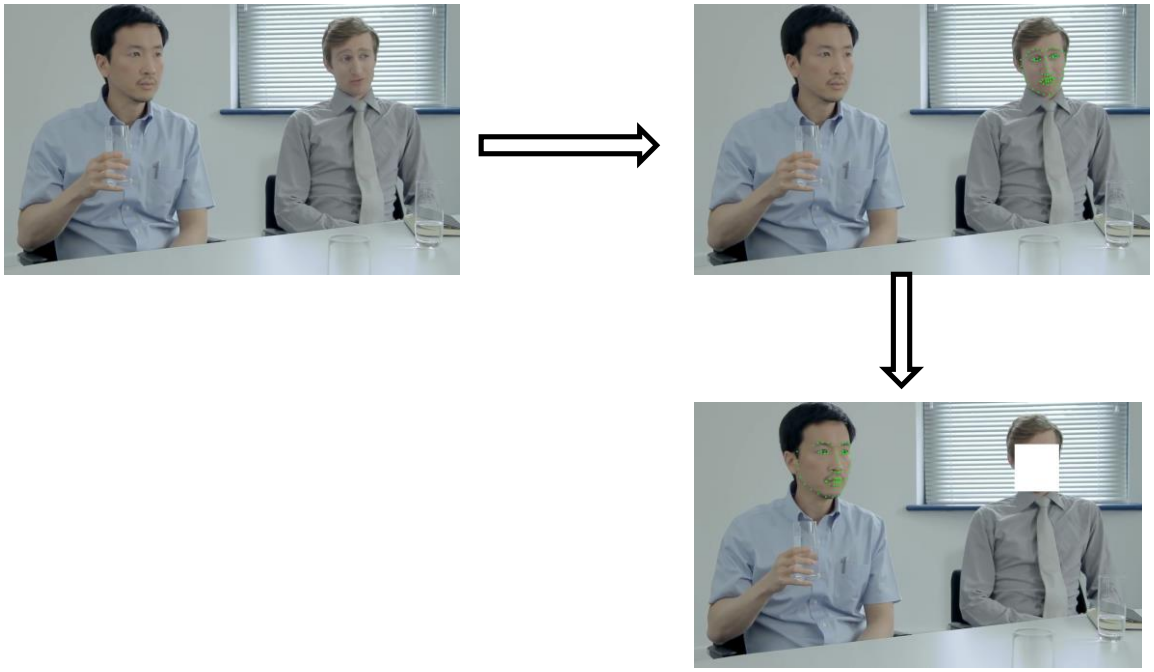

Рисунок 4.1 – Пример устранения рекурсивного захвата лицевых точек

4.2.3 Стандартизация изображений.

Сначала нужно найти центры глаз для выравнивания изображения (листинг 4.5). Поскольку AAM всегда возвращает точки со статической нумерацией, и нам известны номера точек глаз, то их центры высчитываются таким образом:

Листинг 4.5 – Вычисление центров глаз

```
def get_center_eyes(points):
    r = reduce(lambda x, y: (x[0] + y[0], x[1] + y[1]),
               points[36:42])
    l = reduce(lambda x, y: (x[0] + y[0], x[1] + y[1]),
               points[42:48])
    l = (int(l[0] / 6), int(l[1] / 6))
    r = (int(r[0] / 6), int(r[1] / 6))
    return l, r
```

Просто находим среднее из координат точек глаза, это и есть центры глаз. Далее выравнивается и обрезается лицо (листинг 4.6)

Листинг 4.6 – Выравнивание и обрезка лица

```
def CropFace(image, eye_left=(0,0), eye_right=(0,0),
               offset_pct=(0.2,0.2), dest_sz = (70,70)):
    # вычисление смещений в изображении
    offset_h=math.floor(float(offset_pct[0])*dest_sz[0])
    offset_v=math.floor(float(offset_pct[1])*dest_sz[1])
    eye_direction = (eye_right[0] - eye_left[0],
                     eye_right[1] - eye_left[1])
    # вычисляем угол поворота в радианах
    rotation = -math.atan2(float(eye_direction[1]),
                           float(eye_direction[0]))
    # дистанция между глазами
    dist = Distance(eye_left, eye_right)
    reference = dest_sz[0] - 2.0*offset_h
    # масштаб
    scale = float(dist)/float(reference)
    # поворот изображения вокруг левого глаза
    image = ScaleRotateTranslate(image, center=eye_left,
                                 angle=rotation)
    # обрезка повернутого изображения
    crop_xy = (eye_left[0] - scale*offset_h, eye_left[1] -
               scale*offset_v)
    crop_size = (dest_sz[0]*scale, dest_sz[1]*scale)
    image = image.crop((int(crop_xy[0]), int(crop_xy[1]),
                        int(crop_xy[0]+crop_size[0]),
                        int(crop_xy[1]+crop_size[1])))
    image = image.resize(dest_sz, Image.ANTIALIAS)
    return image
```

Для того чтобы привести лица к общему стандарту, используется функция CropFace(), взятая из библиотеки OpenCV. Она выравнивает и вырезает лицо по следующим входным параметрам:

- image – изображение;
- eye_left, eye_right – координаты центров левого и правого глаза;
- offset_pct – процент области изображения от глаз, по которому будет обрезаться лицо (в вертикальном и горизонтальном направлении);

- dest_sz – размер выходного изображения.

4.2.4 Распознавание лиц

Поскольку распознавание с помощью технологии FaceNet показало наилучшие результаты (тестирование алгоритмов рассматривается в главе 5), но имеет большое время работы, которое непригодно для распознавания лиц в режиме реального времени, то было принято решение, вынести процесс распознавания в отдельный поток, т.е. основной поток просто добавляет в очередь обнаруженные лица, а уже второй поток забирает эти лица из очереди и выполняет процесс распознавания.

По умолчанию для классификации вектор-признаков используется алгоритм SVM (Метод опорных векторов), который не может обучаться динамически. Были разработаны следующие методы динамической классификации:

- Метод среднего расстояния: высчитывается среднее расстояние от входного вектора признаков (признаки распознаваемого лица) до объектов каждого класса. В итоге входное лицо принадлежит классу с минимальным средним расстоянием;
- Метод одной ошибки: если входное лицо имеет расстояние до объекта класса, превышающее определенный порог, то считается, что оно не принадлежит этому классу;
- Метод частоты подобных: подсчитывается частота вхождения расстояния от входного до объектов класса в определенный порог. Входной объект принадлежит классу, который имеет наибольшую частоту.

Последний алгоритм работает лучше, чем остальные, поэтому используется по умолчанию для динамического обучения и распознавания.

5 ТЕСТИРОВАНИЕ АЛГОРИТМОВ РАСПОЗНАВАНИЯ ЛИЦ

Для тестирования алгоритмов распознавания были написаны две программы. Одна – для обучения моделей, другая – для распознавания.

5.1 Программа для обучения моделей распознавания

Перед началом обучения нужно собрать лица и их индексы (листинг 5.1).

Листинг 5.1 – Функция, которая проходит по папкам базы для обучения и собирает проиндексированные лица людей.

```
def travel_dirs(path, cascade, predictor, labels, net, outpath,
                min_count, faces=None, faces_nn=None):
    counter = 0
    max_dir = 20
    csv_pred  =  open("{}/pred_{}.csv".format(outpath,  min_count),
                                'w')
    for id, (dir, subdirs, files) in enumerate(os.walk(path)):
        if id == 0 or len(files) < min_count + 1:
            continue
        print id, counter, dir
        flg = False
        for file in files[:min_count]:
            if file.isspace(): continue
            path = os.path.join(dir, file)
            img = cv2.imread(path)
            img = detect_and_cropface(img, cascade, predictor)
            if faces_nn != None:
                    img_nn = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                    img_nn = net.forward(img_nn)
                    faces_nn.append(img_nn)
               else:
                    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                    faces.append(img)
            labels.append(id)
        for file in files[min_count:min_count + 3]:
            if file.isspace(): continue
            line = "{};{}\n".format(os.path.join(dir, file), id)
            csv_pred.write(line)
        counter += 1
        if counter == max_dir: break
        csv_pred.close()
```

Входные аргументы функции:

- path – путь к базе фотографий людей;
- cascade – объект каскада, который обнаруживает лица на фотографии;
- predictor – объект AAM, который нужен для определения лицевых точек;
- outpath – путь для вывода;
- min_count – минимальное количество лиц на каждую персону в базе;

Выходные аргументы функции:

- labels – последовательность индексов собранных лиц;
- faces – сами собранные лица.

Функция travel_dirs() проходит по папкам обучаемой базы, если в папке количество фотографий не достигает mincount + 1 (‘+1’, потому что должна быть хотя бы одна фотография для тестов распознавания), то она игнорируется. Далее на фотографиях детектируются лица и приводятся к определенному стандарту. Те лица, которые не вошли в обучаемую выборку, будут использоваться для тестирования распознавания, данные об этих лицах записаны в файле в определенном формате ('путь фотографии лица'; 'индекс персоны').

После этого происходит само обучение. Функция training() обучает модели по выборке, которую собрала функция travel_dirs() (листинг 5.2).

Листинг 5.2 – Функция обучения

```
def training(alg_name, faces, labels, min_count, net):
    if alg_name == 'eigen':
        face_recognizer = cv2.face.createEigenFaceRecognizer()
    elif alg_name == 'fischer':
        face_recognizer = cv2.face.createFisherFaceRecognizer()
    elif alg_name == 'lbph':
        face_recognizer = cv2.face.createLBPHFaceRecognizer()
    elif alg_name == 'nn':
        clf = SVC(C=1, kernel='linear', probability=True)
        face_recognizer = [net, clf]
    if alg_name == 'nn':
        net, clf = face_recognizer
    if alg_name != 'nn':
        face_recognizer.train(np.asarray(faces),np.asarray(labels))
    face_recognizer.save("{}_{}.xml".format(os.path.join(outpath,
                         alg_name), min_count))
    else:
        clf.fit(np.asarray(faces), np.asarray(labels))
        with open("{}_{}.xml".format(os.path.join(outpath,
                                  alg_name), min_count), 'w') as f:
            pickle.dump(clf, f)
```

5.2 Программа для тестирования распознавания

Входные аргументы программы:

- -f – форматированный файл для тестирования (список изображения, которые будут распознваться);
- -a – имя тестируемого алгоритма;
- -m – обученная модель;
- -t – флаг тестирования алгоритмов по времени;
- -n – максимальное количество объектов для тестирования;
- --nocrop – флаг, обозначающий, что изображения для тестов уже нормализованы.

Основную работу выполняет функция распознавания (листинг 5.3).

Листинг 5.3 Функция распознавания

```
def recognition(predict_csv, face_recognizer, alg_name, max_count,
                    flg_nocrop):
    align=openface.AlignDlib("../data/cascades/shape_predictor_68_
                         face_landmarks.dat")
    fd = open(predict_csv, 'r')
    if alg_name == 'nn':
        net, clf = face_recognizer
    count = 0
    hit = 0
    total = 0
    for line in fd:
        if line.isspace(): continue
        path, id = line[:-1].split(';')
        id = int(id)
        img = cv2.imread(path)
        if flg_nocrop == True:
            rgbImg = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            bb = align.getLargestFaceBoundingBox(rgbImg)
            img = align.align(96, rgbImg, bb,landmarkIndices
                         =openface.AlignDlib.OUTER_EYES_AND_NOSE)
        if alg_name == 'nn':
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            t = time()
            img = net.forward(img).reshape(1, -1)
            total += time() - t
            pr = clf.predict(img)[0]
        else:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            pr = face_recognizer.predict(img)
        if id == pr:
            hit += 1
        count += 1
        if count > max_count: break
              return float(hit) / count, count
```

5.3 Тестирование на базе Georgia Tech Face Database

5.3.1 Особенности базы

- 50 человек;
- 15 фотографий на человека;
- Разные положения лиц;
- Разное освещение;
- Разные выражения лиц;
- Разрешение 640x480.

На рисунке 5.1 приведены фотографии при разном освещении и положении головы.

Рисунок 5.1 – Пример фотографий одного человека из базы Georgia Tech Face Database

5.3.2 Результаты тестирования

Для тестирования были использованы фотографии 20 случайных человек из базы. Данный тест измеряет точность распознавания от количества фотографий на человека. Результаты тестирования представлены в виде графика на рисунке 5.2.

База является простой для распознавания, поскольку фотографии в довольно хорошем качестве. Поэтому все алгоритмы, кроме EigenFaces, имеют довольно хорошие результаты, при количестве фотографий на каждого человека больше семи. Метод EigenFaces отработал плохо, потому что лица в базе были сфотографированы в разном положении и освещении, что является слабой стороной этого метода.

Метод FisherFaces, при количестве фотографий больше пяти в обучаемой базе, работает лучше LBPH, поскольку, если посмотреть на выравненные лица, то видно, что в базе присутствуют лица под наклоном, но небольшим.

Абсолютным лидером в этом тесте оказался метод FaceNet, его точность составляет почти 100% для любого количества фотографий на человека.

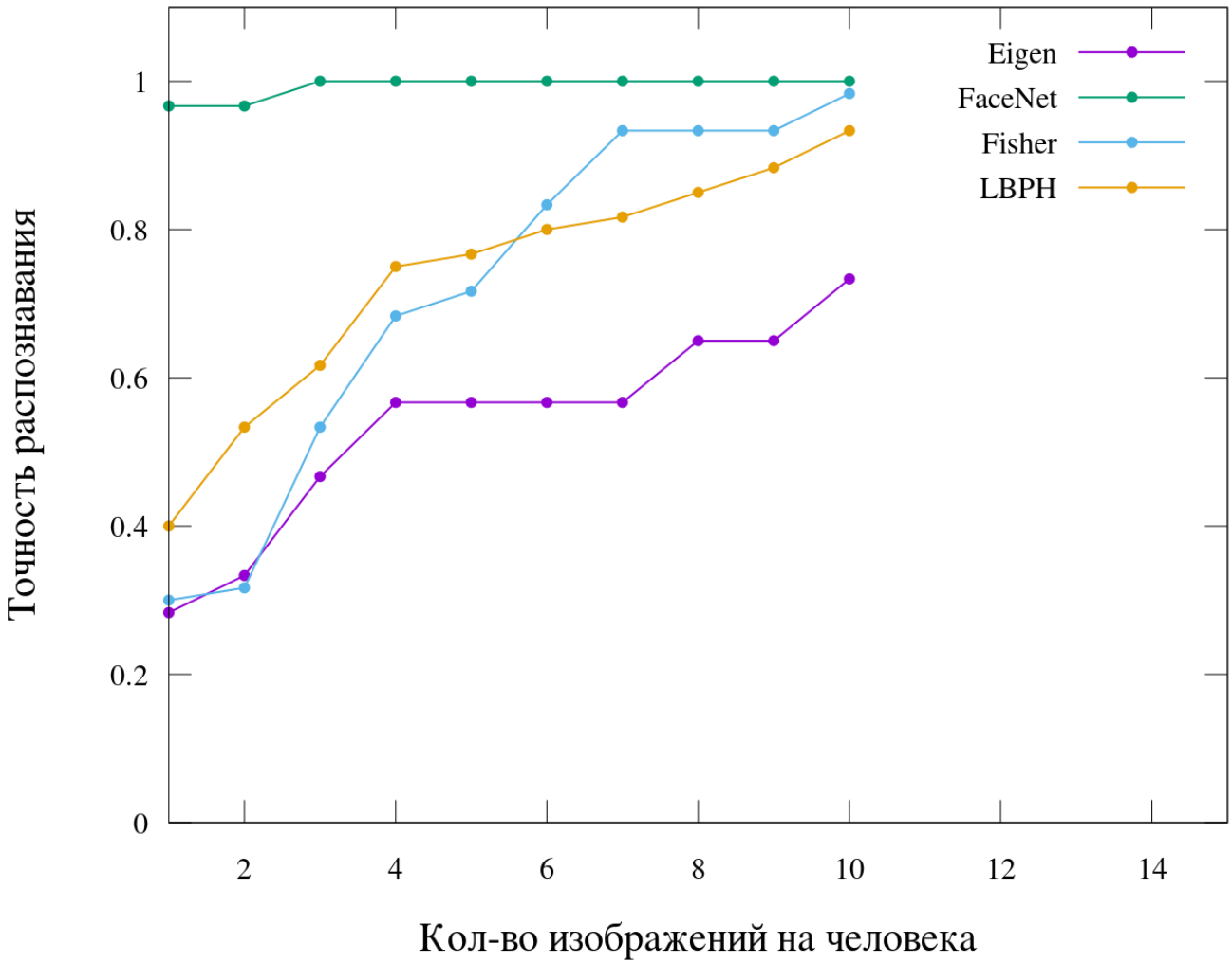

Рисунок 5.2 – График сравнения точности алгоритмов распознавания на базе Georgia Tech Face Database

5.4 Тестирование на базе LFW

5.4.1 Особенности базы

- 5749 человек, 12000 изображений, 1680 человек имеют по 2 и более фотографии, остальные по одной;
- Фотографии имеют разрешение 250x250;
- Разное время съёмки и качество фотографий;
- Имеются всяческие преграждения;
- Является одной из самых сложных баз для распознавания лиц.

На рисунке 5.3 приведены примеры фотографий из базы.

5.4.2 Результаты тестирования

Для тестирования были использованы фотографии 20 случайных человек из базы. Данный тест измеряет точность распознавания от количества фотографий на человека. Результаты тестирования представлены в виде графика на рисунке 5.4. Поскольку база LFW тяжелее, чем GTFD, соответственно результаты тестирования оказались похуже. Но метод FaceNet имеет почти 100% точность во всех случаях.

Рисунок 5.3 – Фотографии из базы LFW

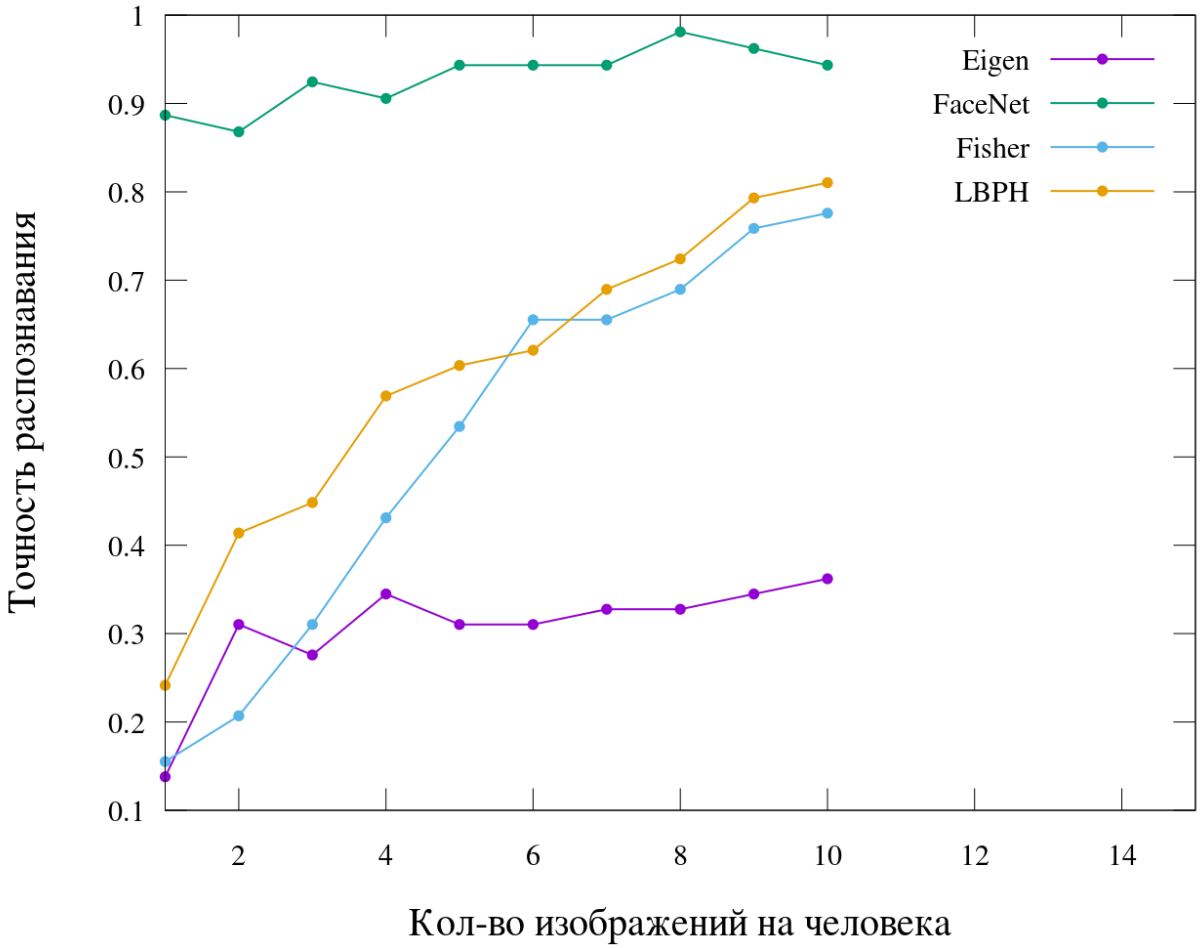

Рисунок 5.4 – График сравнения точности алгоритмов распознавания на базе LFW (20 персон в выборке)

Поскольку нет ничего идеального, особенно в задаче распознавания лиц, и нужно было найти еще какую-нибудь зависимость результатов, кроме качества фотографий в базе. Решено было попробовать увеличить количество классов в выборке. Количество классов было увеличено с 20 до 40. Результаты эксперимента показаны на рисунке 5.5.

Как и ожидалось, точность у всех алгоритмов упала. То есть можно сделать вывод, что точность распознавания зависит от количества персон в выборке. И метод FaceNet не является идеальным при большом количестве разных личностей в выборке.

Таким образом выделяются следующие факторы от которых зависит точность распознавания:

- Качество фотографий;
- Количество персон в выборке;
- Количество фотографий на человека (фотографии должны быть не однотипные).

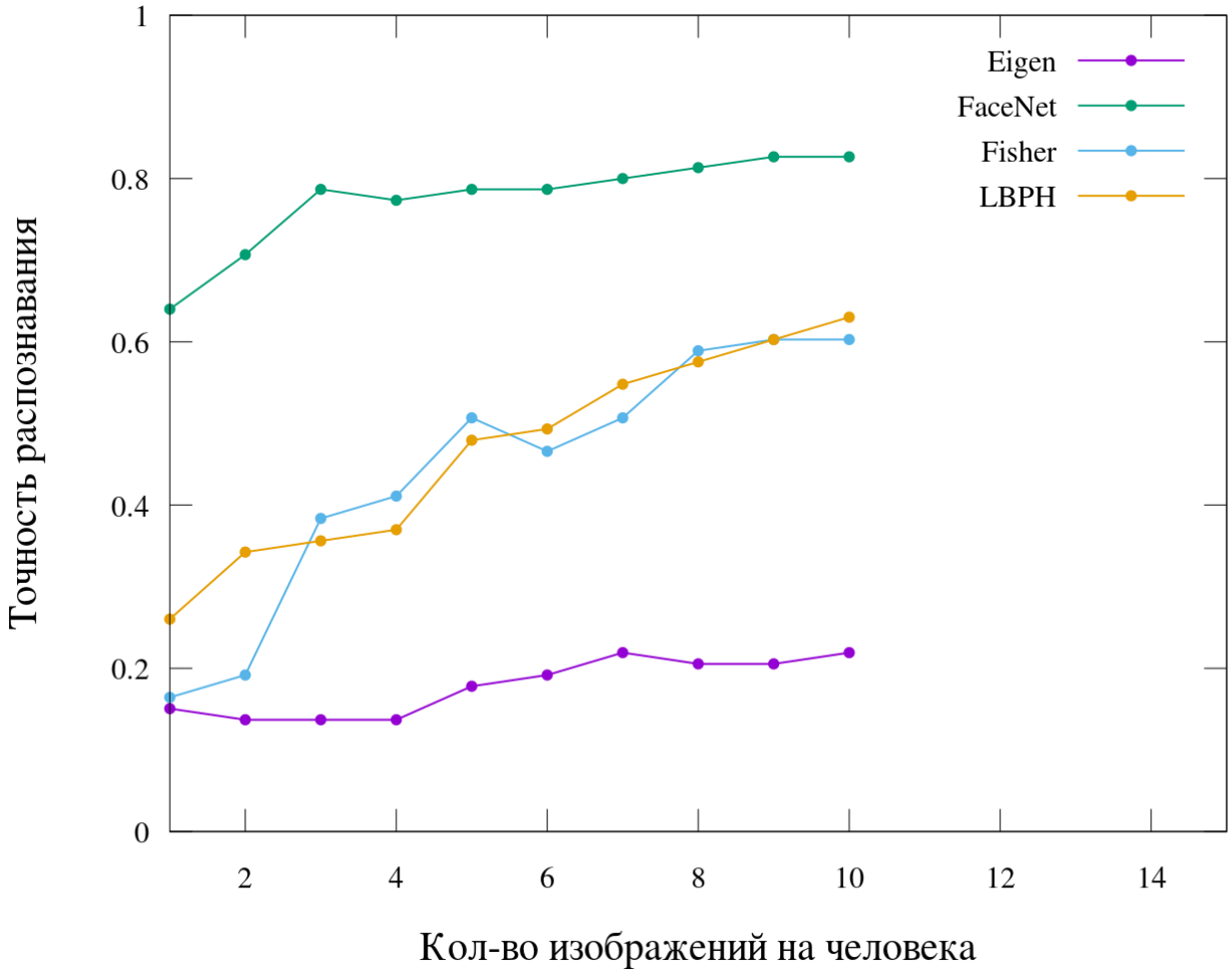

Рисунок 5.5 – График сравнения точности алгоритмов распознавания на базе LFW (40 персон в выборке)

5.5 Сравнение времени выполнения алгоритмов

Были выполнены замеры среднего времени выполнения распознавания всех рассматриваемых алгоритмов в данной работе. Результаты замеров представлены в виде гистограммы на рисунке 5.6.

Все алгоритмы, кроме FaceNet, показали довольно хорошее время, которое подходит для работы в режиме реального времени. Алгоритм FaceNet оказался примерно в 50 раз медленнее (99% времени выполнения занимает выделение признаков, остальное приходится на классификацию), чем LBPH, у которого среднее время работы самое большое среди остальных методов.

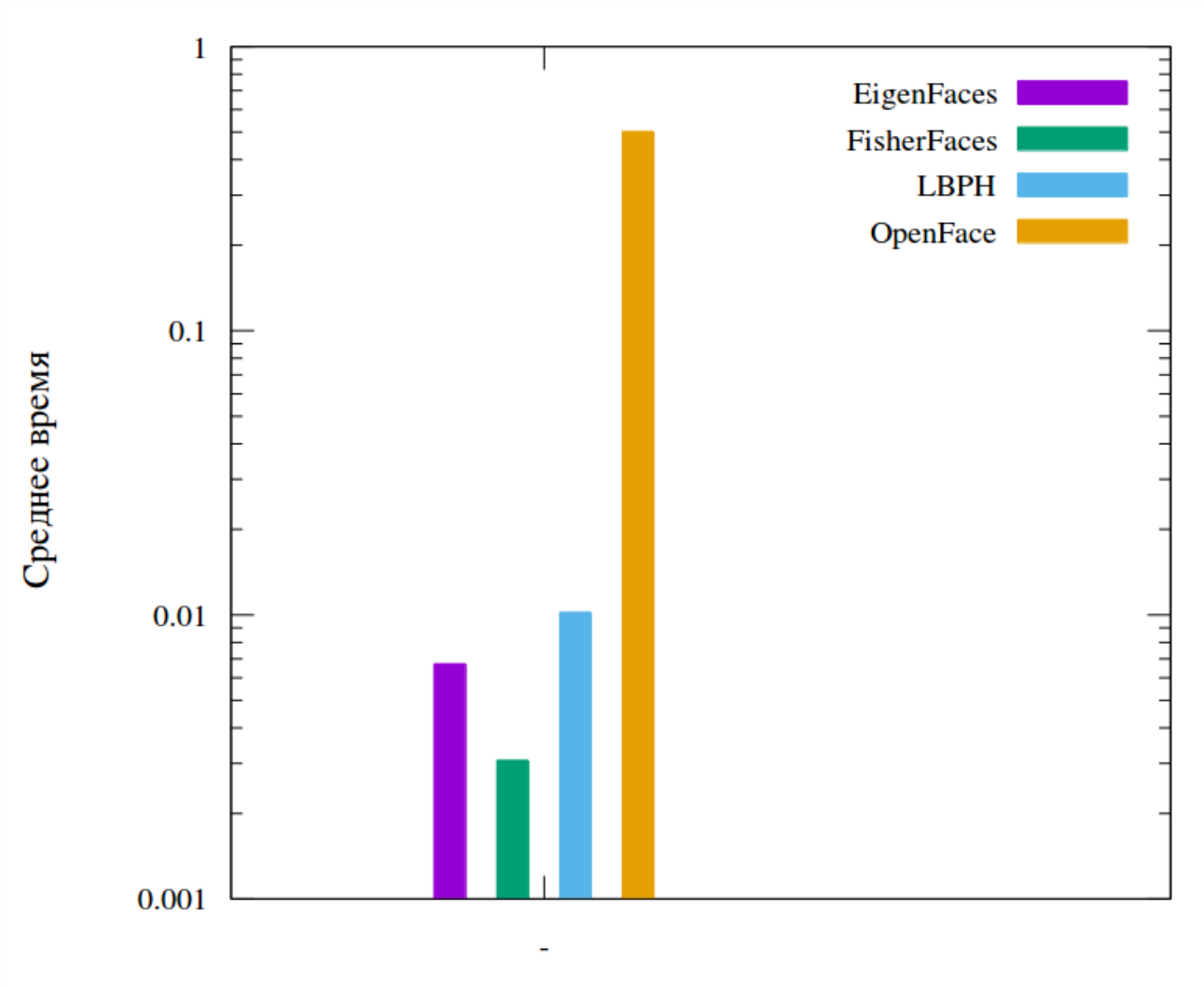

Рисунок 5.6 – Среднее время работы алгоритмов

6 ЗАКЛЮЧЕНИЕ

В рамках бакалаврской работы были рассмотрены:

- этапы подготовки лиц к распознаванию;
- основные методы распознавания лиц.

Все рассмотренные методы были протестированы с разными параметрами и условиями. По результатам экспериментов был выбран самый точный метод распознавания из рассмотренных (FaceNet), который используется в целевой программе, распознающей лица из видеопотока.

ПРИЛОЖЕНИЕ А

(справочное)
Библиография

1 Разработка мультимедийных приложений с использованием библиотек OpenCV и IPP [Электронный ресурс]/ А.В. Бовырин [и др.].— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 515 с.— Режим доступа: http://www.iprbookshop.ru/39564.— ЭБС «IPRbooks», по паролю

2 Шапиро Л. Компьютерное зрение / М.: БИНОМ. Лаборатория знаний, 2013.— 752 с.

3 Georgia Tech Face Database. URL: http://www.anefian.com/research/face_reco.htm (Дата обращения 15.03.2016)

4 Kanade, T. Picture processing system by computer complex and recognition of human faces. PhD thesis, Kyoto University, 1973.

5 Turk, M., and Pentland, A. Eigenfaces for recognition. Journal of Cognitive Neuroscience 3, 1991. 71–86 с.

6 FaceNet: A Unified Embedding for Face Recognition and Clustering. URL: https://arxiv.org/pdf/1503.03832v3.pdf (Дата обращения 15.03.2016)

7 Face Recognition with Local Binary Patterns. URL: http://uran.donetsk.ua/~masters/2011/frt/dyrul/library/article8.pdf (Дата обращения 15.03.2016)

8 Face Recognition with OpenCV URL: http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html (Дата обращения 15.03.2016)

9 Dlib. URL: http://dlib.net/

10 Labeled Faces in the Wild. URL: http://vis-www.cs.umass.edu/lfw/ (Дата обращения 15.03.2016)

11 Метод Виолы-Джонса (Viola-Jones) как основа для распознавания лиц. URL: https://habrahabr.ru/post/133826/ (Дата обращения 15.03.2016)

12 Метод опорных векторов URL: https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BE%D0%BF%D0%BE%D1%80%D0%BD%D1%8B%D1%85_%D0%B2%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BE%D0%B2

ПРИЛОЖЕНИЕ Б

(рекомендуемое)
Наиболее употребляемые текстовые сокращения

SVM – support vector machine
AAM – active appearance model
LFW – Labeled Faces in the Wild
CNN – convolutional neural network
GTFD – Georgia Tech Face Database