Министерство образования и науки РФ

Федеральное государственное образовательное

Учреждение профессионального образования

«Казанский (приволжский) федеральный университет»

Институт вычислительной математики и информационных технологий

Кафедра системного анализа и информационных технологий

Направление: 02.03.02 - Фундаментальная информатика

и информационные технологии

Профиль: Системный анализ и информационные технологии

**КУРСОВАЯ РАБОТА**

Определение эмоций на лице человека

Студент 3 курса

Группа 09-408:

"\_\_\_"\_\_\_\_\_\_\_\_\_ 2017 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Рычков Ф.С.)

Научный руководитель

ассистент кафедры САиИТ

"\_\_\_"\_\_\_\_\_\_\_\_\_ 2017 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Нигматуллин Р.Р.)

**Казань – 2017**

[ВВЕДЕНИЕ 3](#_Toc493430566)

[1. ПОСТАНОВКА ЗАДАЧИ 4](#_Toc493430567)

[2. МЕТОДЫ НАХОЖДЕНИЯ ЛИЦА ПО ФОТОГРАФИИ 5](#_Toc493430568)

[2.1. Метод собственных лиц или метод главных компонент 5](#_Toc493430569)

[2.1.1. Описание алгоритма 5](#_Toc493430570)

[2.2. Линейный дискриминант Фишера 6](#_Toc493430571)

[2.2.1. Особенности метода 7](#_Toc493430572)

[3. РЕАЛИЗАЦИЯ 9](#_Toc493430573)

[4. ЭКСПЕРИМЕНТЫ 10](#_Toc493430574)

[4.1. Описание эксперимента 10](#_Toc493430575)

[4.1.1. Линейный дискриминант Фишера 10](#_Toc493430576)

[4.1.1.1. Примеры ошибок классификатора 10](#_Toc493430577)

[4.1.1.2. Качество распознавания изображений с шумом 12](#_Toc493430578)

[4.1.1.3. Качество распознавания изображений с уменьшением разрешения (без шума). 14](#_Toc493430579)

[4.1.2 Метод собственных лиц 14](#_Toc493430580)

[4.2.1.2. Качество распознавания изображений с шумом 15](#_Toc493430581)

[4.2.1.3. Качество распознавания изображений с уменьшением разрешения (без шума). 17](#_Toc493430582)

[5. ВЫВОД 18](#_Toc493430583)

[6. СПИСОК ЛИТЕРАТУРЫ 19](#_Toc493430584)

[7. ПРИЛОЖЕНИЕ 20](#_Toc493430585)

# ВВЕДЕНИЕ

На сегодняшний день мир заполонила идея создания искусственного интеллекта. Человечество добилось больших успехов в достижении данной цели. Общество старается создать большое количество новейшей техники, роботов, все больше похожих по своим характеристикам на людей. Доминирующей технологией по улучшению коммуникации между человеком и техникой в скором будущем, вероятнее всего, станет использование сканера по считыванию эмоций для наилучшего взаимодействия между машиной и человеком.

Для решения проблемы определения эмоции на лице человека существует множество подходов. Но эффективность каждого из них ограничена. В своей работе я бы хотел рассмотреть несколько из них.

# 1. ПОСТАНОВКА ЗАДАЧИ

Дано некоторое множество изображений, каждое из которых принадлежит определённой эмоции. Требуется построить классификатор, который по входному изображению выдавал бы номер эмоции. {0=neutral, 1=anger, 2=disgust, 3=fear, 4=happy, 5=sadness, 6=surprise}

Для того, чтобы научиться определять эмоции на лице человека необходимо выбрать метод, по которому мы будем распознавать лицо человека по фотографии.

Создание подобного метода весьма нетривиальная задача, над решением которой работает большое количество людей и разработано множество алгоритмов.

# 2. МЕТОДЫ НАХОЖДЕНИЯ ЛИЦА ПО ФОТОГРАФИИ

Описание математических моделей методов взяты из документации библиотеки OpenCV [2].

## 2.1. Метод собственных лиц или метод главных компонент

Проблема связанная с обработкой изображений – это их большая размерность. Двумерные серые изображения охватывают **-** мерное векторное пространство, поэтому изображения 100x100 пикселей уже находятся в пространстве размерности 10 000. Вопрос в том, являются ли все измерения одинаково полезными для нас? Мы можем только принять решение, если есть какая-либо разница в данных, поэтому мы ищем компоненты, на которые приходится большая часть информации.

Метод главных компонент [3] (МГК) был независимо предложен Карлом Пирсоном (1901) и Гарольдом Хотеллином (1933), чтобы превратить множество возможных коррелированных переменных в меньший набор некоррелированных переменных. Идея состоит в том, что высокоразмерный набор данных часто описывается коррелированными переменными, и поэтому для большинства информации приходится лишь несколько значимых измерений. Метод главных компонент находит направления с наибольшей дисперсией в данных, называемые главными компонентами.

## 2.1.1. Описание алгоритма

Пусть   случайный вектор, .

Вычисляем среднее

Вычисляем матрицу ковариации

Вычисляем собственные значения и собственные векторы из .

Очерёдность собственных векторов определяется по их значениям. компоненты собственных векторов сортируются по наибольшему значению.

компоненты представленные вектором находятся следующим образом:

Где .

Восстановление из МГК базиса производится следующим образом:

Где .

После, метод Eigenfaces выполняет распознавание лиц:

* Проецирование всех учебных образцов в подпространство МГК.
* Проецирование запрошенного изображения в подпространство МГК.
* Поиск ближайшего соседа между проецируемыми обучающими изображениями и проецируемым изображением в запросе.

## 2.2. Линейный дискриминант Фишера

Метод главных компонент (МГК), который является ядром метода собственных лиц (Eigenfaces), находит линейную комбинацию функций, которая максимизирует общую дисперсию данных. Хотя это явно мощный способ представления данных, он не рассматривает какие-либо классы, и поэтому при отбрасывании компонентов может быть потеряна большая дискриминационная информация. Представим ситуацию, когда дисперсия данных генерируется внешним источником, пусть это будет свет. Компоненты, идентифицированные с помощью МГК не обязательно содержат какую-либо дискриминационную информацию вообще. Поэтому проецируемые образцы размываются вместе, и классификация становится невозможной .

Линейный дискриминантный анализ [4] выполняет снижение размерности по классу и был изобретен Р. А. Фишером. Он успешно использовал его для классификации цветов в своей статье 1936 года. Чтобы найти комбинацию функций, которая лучше всего разделяет классы, линейный дискриминантный анализ максимизирует отношение между классами к разбросу внутри классов, а не максимизирует общий разброс. Идея проста: одни и те же классы должны быть тесно связаны друг с другом, а разные классы располагаются как можно дальше друг от друга.

## 2.2.1. Особенности метода

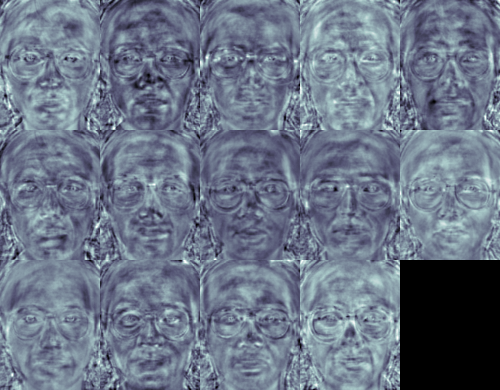


Рис 2.2.1. (Yale Facedatabase)

Метод Fisherfaces изучает матрицу преобразования класса, поэтому они не захватывают освещение, как метод Eigenfaces. Дискриминантный анализ вместо этого обнаруживает черты лица, чтобы различать людей. Важно отметить, что производительность Fisherfaces сильно зависит от входных данных. То есть: если вы изучаете Fisherfaces только для хорошо освещенных изображений, и вы пытаетесь распознать лица в сценах с плохой освещенностью, тогда метод, вероятно, найдет неправильные компоненты (просто потому, что эти функции могут не преобладать на плохо освещенных изображениях). Это вполне логично, так как метод не имел возможности изучить освещение.

## 2.2.2. Описание алгоритма

Пусть – случайный вектор признаков взятых из классов:

Матрица межклассовой дисперсии - и матрица внутреклассовой

дисперсии - вычисляются так:

Где – общее среднее:

А является средним класса :

Матрица W для проецирования пространства изображения на пространство

признаков выбирается из следующего условия:

Может существовать до  векторов составляющих базис пространства

признаков, где  – общее число классов. С помощью этих векторов

пространство изображений переводится в пространство

признаков. Поскольку работа непосредственно с матрицей

затруднительна из-за её размерности, использовано

предварительное уменьшение размерности с помощью метода главных

компонент, и затем вычисления производятся в пространстве меньшей

размерности:

Матрица преобразования W, которая проецирует образец в (c-1) -мерное пространство, определяется следующим образом:

# 3. РЕАЛИЗАЦИЯ

Все эти методы были реализованы на языке Python 3.5 с использованием

пакетов Numpy, OpenCV 3.1.0. Методы для работы с изображениями были

разделены на 3 файла: datasetCreated.py, EmotionClassifier.py, imgExtract.py.

В работе используется выборка изображений от Cohn-Kanade[1]. В файле readme авторы отмечают, что только подмножество (327 из 593) эмоциональных последовательностей действительно содержит архетипические эмоции. Каждая последовательность изображений состоит из формирования эмоционального выражения, начиная с нейтрального лица и заканчивая эмоциями.

В файле imgExtract.py с использованием папки изображений и папки с закодированными эмоциями были сформированы папки Neutral, Anger и т.д., где расположились соответствующие изображения эмоций. В файле datasetCreated.py с использованием четырёх каскадов Хаара, поставляемых вместе с библиотекой OpenCV, был написан скрипт для нахождения лица на фотографии. Скрипт вырезал найденные лица и заново сортировал изображения по эмоциям (пример Рис 3.1.).



Рис 3.1.

В файле EmotionClassifier.py проводится несколько экспериментов с

Eigenfaces и Fisherfaces, которые реализованы самой библиотекой OpenCV.

# 4. ЭКСПЕРИМЕНТЫ

Все вычисления проводились на ноутбуке с процессором Intel Core i3 6100U

CPU с максимальной частотой 2.3 ГГц, оперативной памятью объемом 11.9Гб

и типом DDR4. Для экспериментов использовались изображения с оттенками

серого, размером 350x350.

## 4.1. Описание эксперимента

## 4.1.1. Линейный дискриминант Фишера

Создаём набор для обучения и классификации, мы случайным образом проецируем и тренируем 80% данных и классифицируем оставшиеся 20% и повторяем процесс N раз. Такой подход дал ~ 69% точность. 69% это не очень хороший результат, однако если бы классификатор ничего не распознал точность была бы порядка 12.5% [ - т.к. 8 категорий].

Так как используемый набор данных слишком идеализирован (как на рис 4.1),все лица направлены в камеру и выражения лиц преувеличены. Было решено добавить более естественные изображения при помощи поиска Google. Это действие немного снизило качество распознавания до ~ 61%. Ухудшение качества распознавания, скорее всего, связано с изменением некоторых немаловажных признаков на изображении. Например: изменение ракурса съёмки, изменение освещения.

Было предположено, что такая малая точность распознавания так же связана с нехваткой примеров таких эмоций, как презрение (18 изображений), страх (25 изображений) и печаль (28 изображений). После того, как эти эмоции были вычеркнуты из списка классификатора, точность возросла до ~ 83%. Однако, учитывая, что это не является выходом из ситуации, было решено вернуть эти эмоции в классификатор.

## 4.1.1.1. Примеры ошибок классификатора

Ошибки, которые мог бы допустить и человек (Рис 4.1, 4.2, 4.3):



Рис 4.1. (Удивление классифицированное, как Радость).



Рис 4.2. (Удивление классифицированное, как Обычное).

****

Рис 4.3. (Грусть классифицированное, как Отвращение).

Однако большая часть ошибок представляет из себя проблемы распознавания классификатора (Рис 4.4):



Рис 4.4. (Агрессия классифицированная, как обычное выражение лица)

## 4.1.1.2. Качество распознавания изображений с шумом

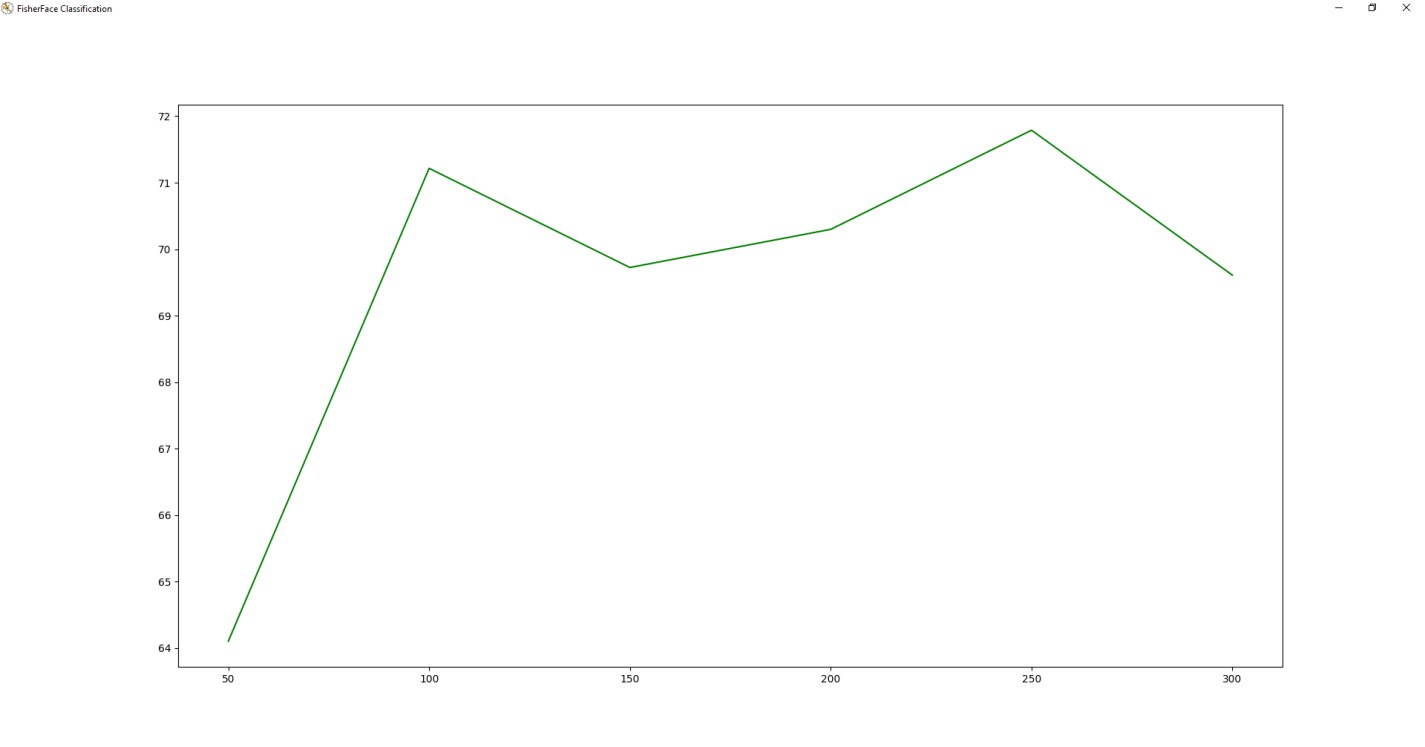


Рис 4.1.1.2.1. График качества распознавания классификатором изображений с аддитивным случайным шумом N(mean=0, sigma=50).

На рисунке 4.1.1.2.1. отображено качество распознавания изображений с аддитивным случайным шумом N(mean=0, sigma=50). Разрешение картинок 50x50, 100x100, 150x150,..,350x350. Вертикальная ось – точность от 64% – 72%. Классификатор обучался на картинках с шумом и пытался распознать эмоции на зашумлённых изображениях.

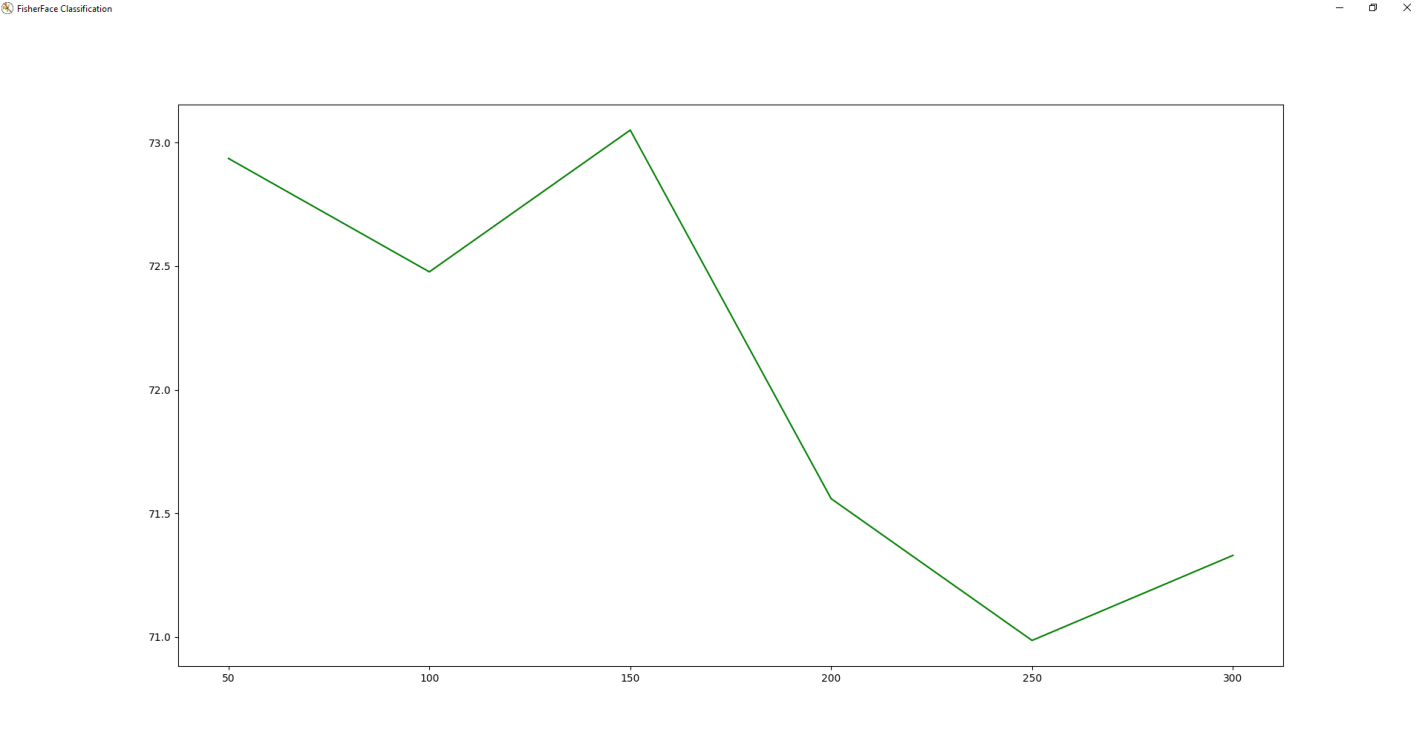


Рис 4.1.1.2.2. График качества распознавания классификатором изображений с аддитивным случайным шумом N(mean=0, sigma=50).

На рисунке 4.1.1.2.2. отображено качество распознавания изображений с аддитивным случайным шумом N(mean=0, sigma=50). Разрешение картинок 50x50, 100x100, 150x150,..,350x350. Вертикальная ось – точность от 71% - 73%. Классификатор обучался на картинках с шумом и пытался распознать эмоции на изображениях без помех.

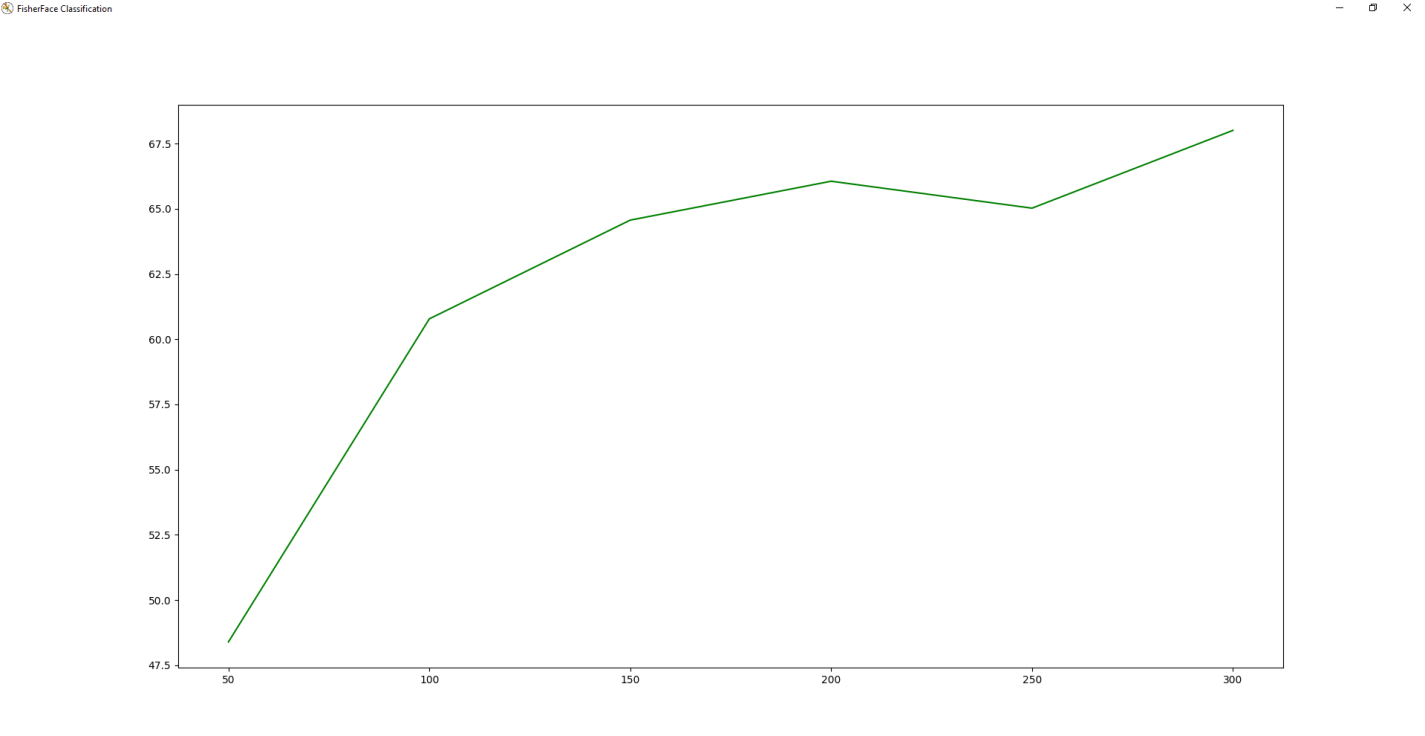


Рис 4.1.1.2.3 График качества распознавания классификатором изображений с аддитивным случайным шумом N(mean=0, sigma=50).

На рисунке 4.1.1.2.3 отображено качество распознавания изображений с аддитивным случайным шумом N(mean=0, sigma=50). Разрешение картинок 50x50, 100x100, 150x150,..,350x350. Вертикальная ось – точность от 47.5% - 67.5%. Классификатор обучался на картинках без шума и пытался распознать эмоции на зашумлённых изображениях.

## 4.1.1.3. Качество распознавания изображений с уменьшением разрешения (без шума).

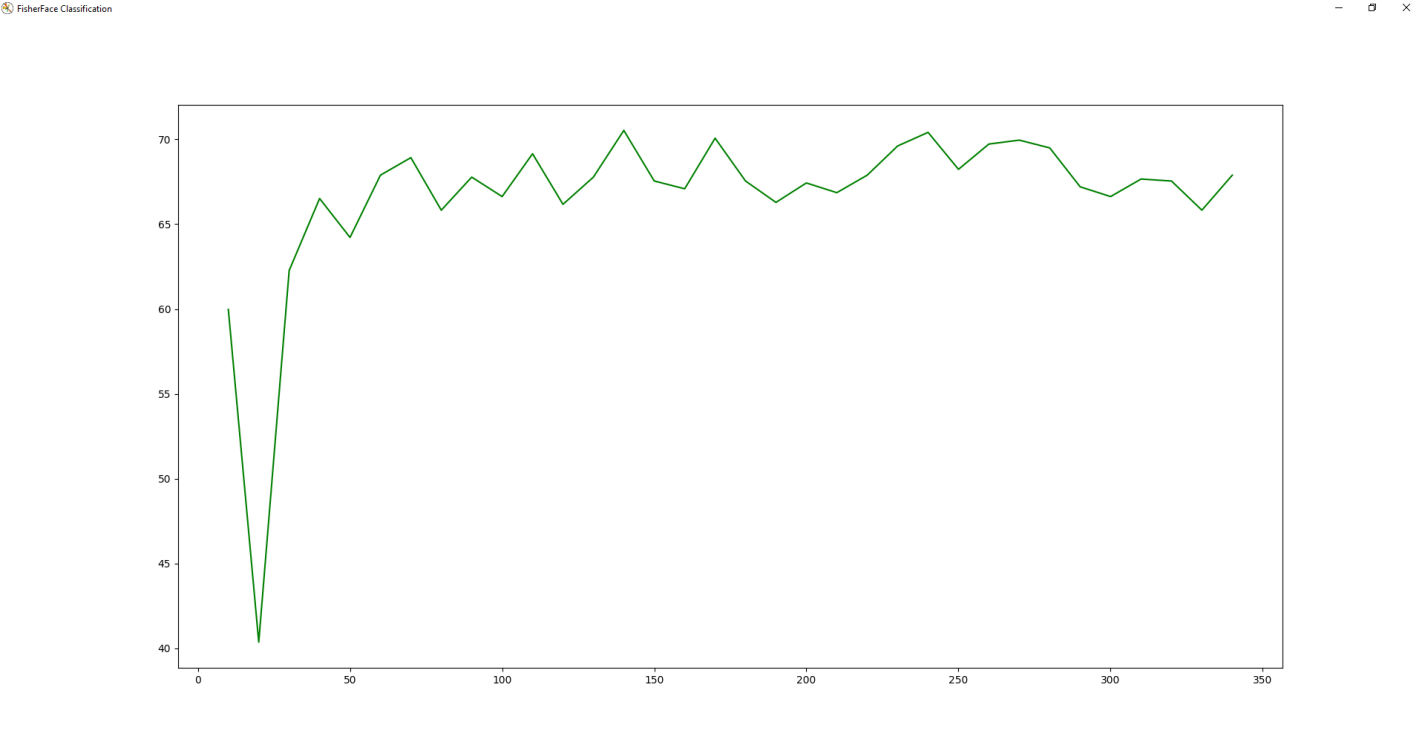


Рис 4.1.1.3 График качества распознавания классификатором на различных размерах изображений.

На рисунке 4.1.1.3 отображена зависимость распознавания эмоций на фотографии от размера изображения. По вертикальной оси представлены проценты (точность распознавания), по горизонтальной оси размер изображения. Эксперимент проходил так: изначально у нас было изображения 350x350 пикселей, каждую итерацию цикла размер уменьшался на 10 пикселей по каждой оси. Обучение и распознавание проводилось 8 раз, после бралось среднее значение точности и записывалось как итоговый результат на определённой размерности. На графике так же видно, что с уменьшением размера изображение, так же ухудшалось и качество распознавания. Однако после определённого размера классификатор вновь стал показывать «высокую» точность. Это связано с тем, что мы начали распознавать пятна, где то они были похожи, а где то нет. Эксперимент занял 128 минут (без распараллеливания).

## 4.1.2 Метод собственных лиц

В данном методе мы делаем аналогичные шаги, что делали в эксперименте с

дискриминантом Фишера. Создаём набор для обучения и классификации, мы

случайным образом проецируем и тренируем 80% данных и классифицируем

оставшиеся 20% и повторяем процесс N раз. Такой подход дал ~ **20%**

точности. Это плохой результат.

[FisherFace] при таких же условиях дал нам ~ **69%** точности.

## 4.2.1.2. Качество распознавания изображений с шумом

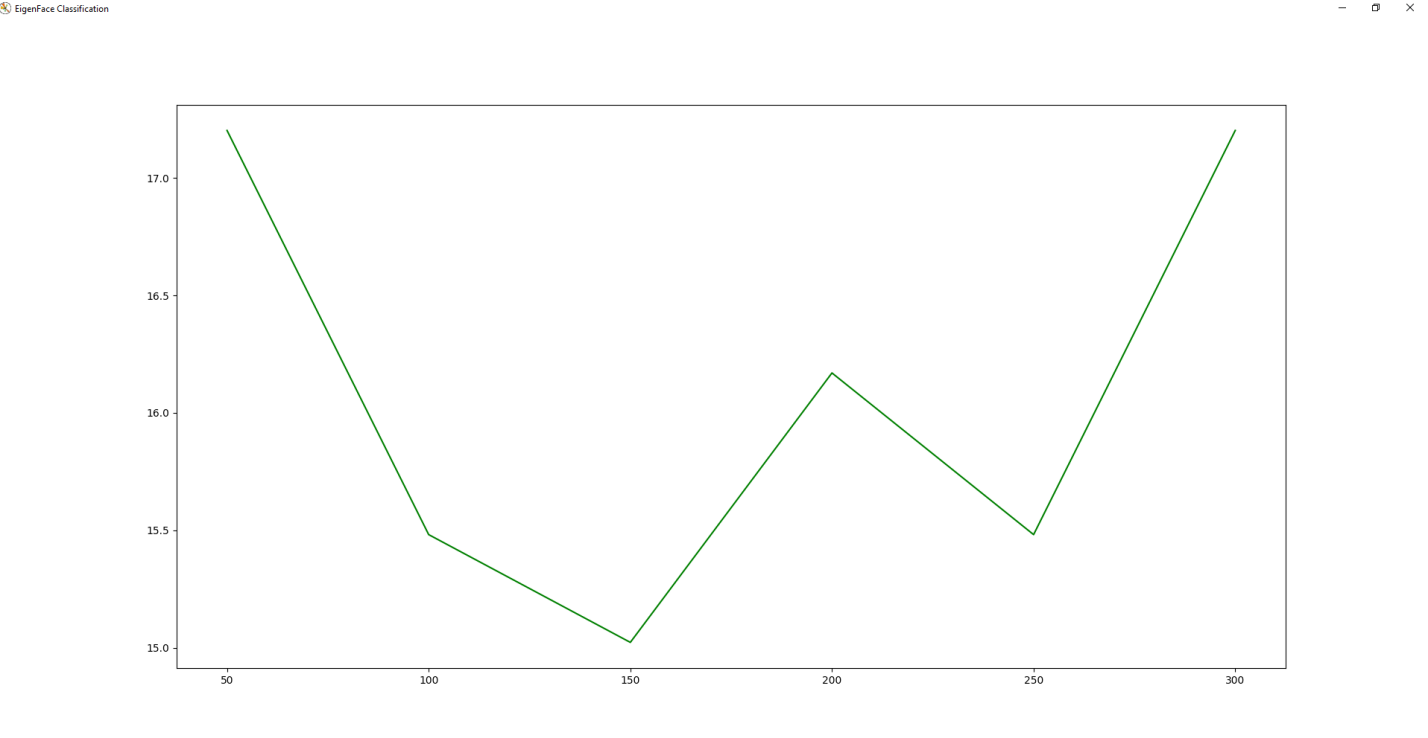


Рис 4.2.1.2.1 График качества распознавания классификатором изображений с аддитивным случайным шумом N(mean=0, sigma=50)

На рисунке 4.2.1.2.1 отображено качество распознавания изображений аддитивным случайным шумом N(mean=0, sigma=50). Разрешение картинок 50x50, 100x100, 150x150,..,350x350. Вертикальная ось – точность от 15% – 17%. Классификатор обучался на картинках с шумом и пытался распознать эмоции на зашумлённых изображениях.

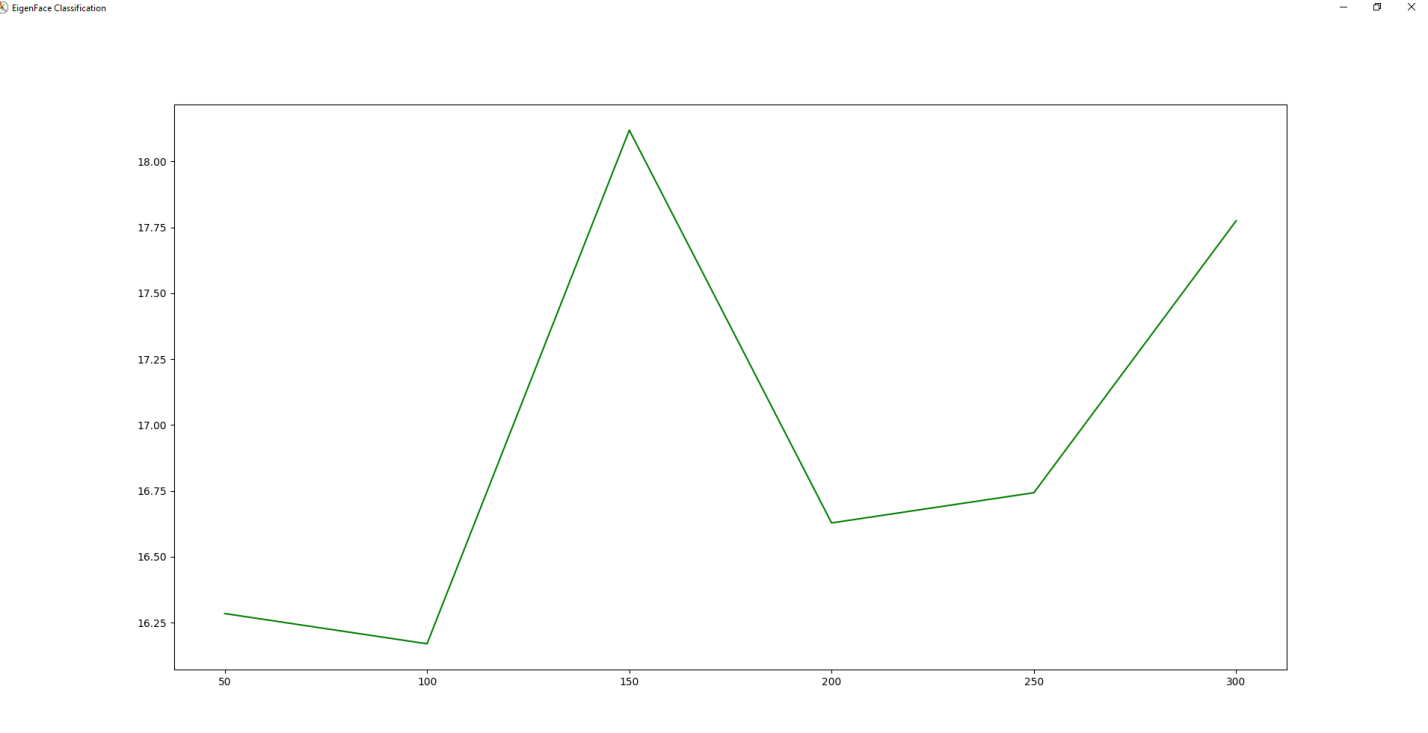


Рис 4.2.1.2.2 График качества распознавания классификатором изображений с аддитивным случайным шумом N(mean=0, sigma=50).

На рисунке 4.1.1.2.2 отображено качество распознавания изображений с аддитивным случайным шумом N(mean=0, sigma=50). Разрешение картинок 50x50, 100x100, 150x150,..,350x350. Вертикальная ось – точность от 16.25% - 18%. Классификатор обучался на картинках с шумом и пытался распознать эмоции на изображениях без помех.

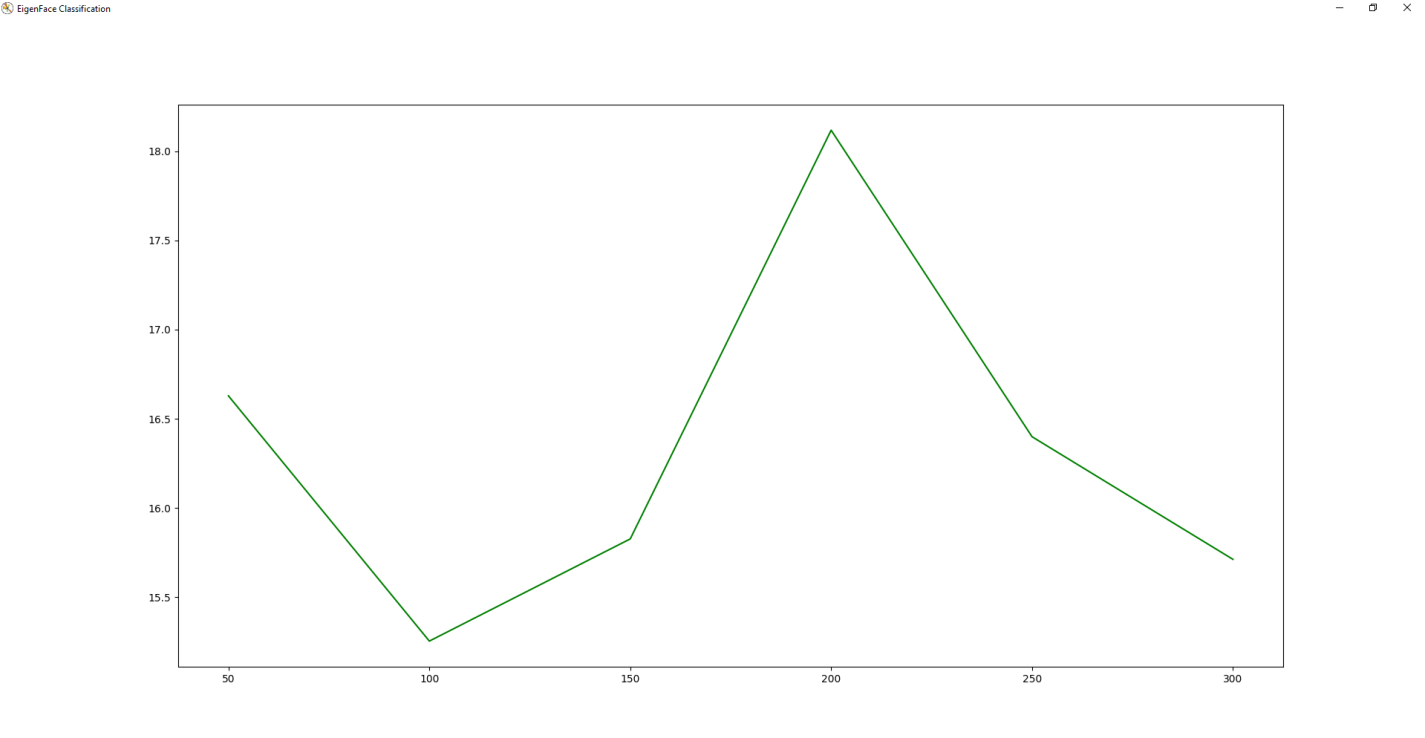


Рис 4.2.1.2.3 График качества распознавания классификатором изображений с аддитивным случайным шумом N(mean=0, sigma=50).

На рисунке 4.2.1.2.3 отображено качество распознавания изображений с аддитивным случайным шумом N(mean=0, sigma=50). Разрешение картинок 50x50, 100x100, 150x150,..,350x350. Вертикальная ось – точность от 14% - 19%. Классификатор обучался на картинках без шума и пытался распознать эмоции на зашумлённых изображениях.

## 4.2.1.3. Качество распознавания изображений с уменьшением разрешения (без шума).

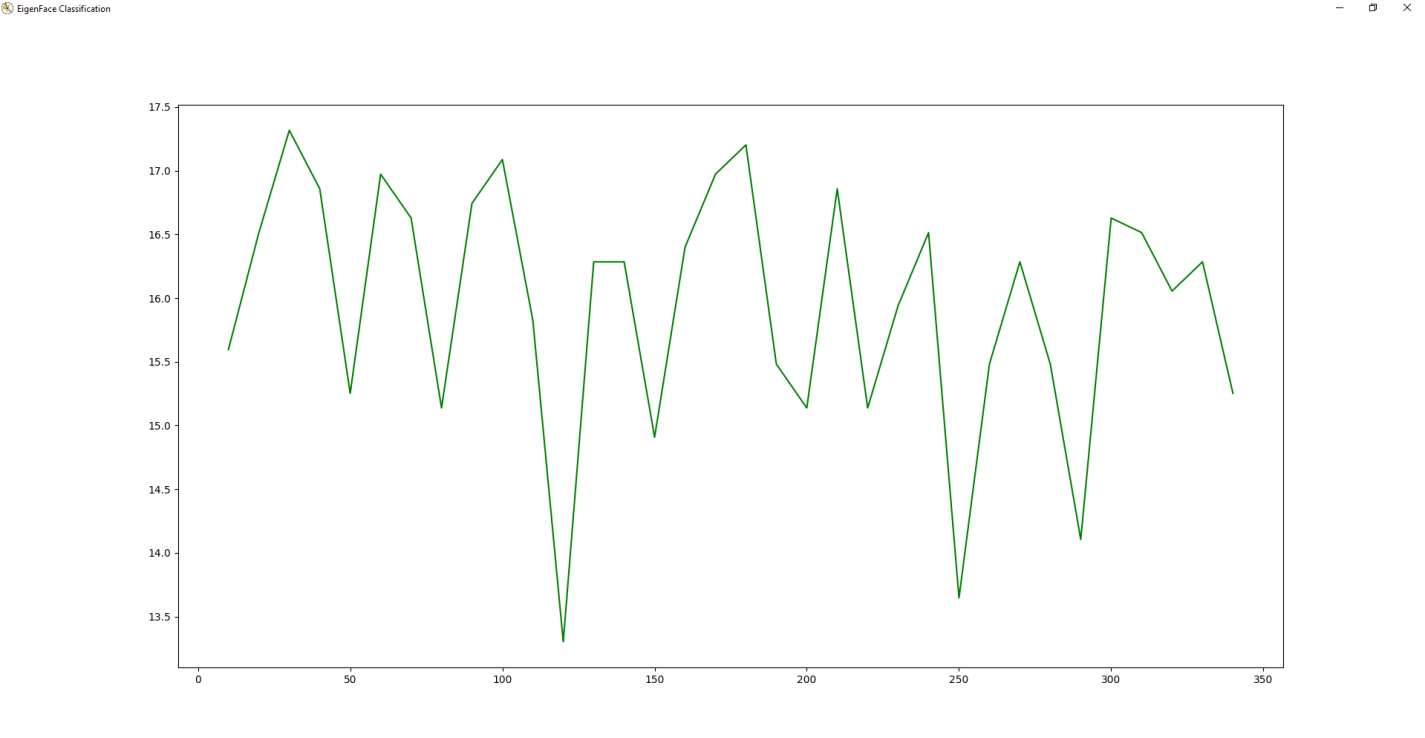


Рис 4.2.1.3 График качества распознавания классификатором на различных размерах изображений.

На рисунке 4.2.1.3 отображена зависимость распознавания эмоций на фотографии от размера изображения. По вертикальной оси представлены проценты (точность распознавания), по горизонтальной оси размер изображения. Эксперимент проходил так: изначально у нас было изображения 350x350 пикселей, каждую итерацию цикла размер уменьшался на 10 пикселей по каждой оси. Обучение и распознавание проводилось 8 раз, после бралось среднее значение точности и записывалось как итоговый результат на определённой размерности. Эксперимент занял 200 минут (без распараллеливания).

# 5. ВЫВОД

Из всех экспериментов видно, что линейный дискриминант Фишера или FisherFace, используемый в библиотеке OpenCV, показал большую эффективность во время классификации различных эмоций людей на фотографиях. Однако точность его работы хоть и не превышает порядка 72% это всё равно лучше, чем показал метод собственных лиц, который при таких же условиях отработал намного хуже. По скорости работы линейный дискриминант Фишера показывает себя лучше. На самом большом, по объёму вычислений, эксперименте, который был проведен в пунктах 4.1.1.3. и 4.2.1.3. линейный дискриминант Фишера завершил своё выполнение за 128 минут, тогда как метод собственных лиц за 200 минут.

Эта небольшая работа показывает, что даже такую нетипичную задачу, как классификация эмоции человека можно решить встроенными методами.

# 6. СПИСОК ЛИТЕРАТУРЫ

1 – Lucey, P., Cohn, J. F., Kanade, T., Saragih, J., Ambadar, Z., & Matthews, I. The Extended Cohn-Kanade Dataset (CK+) – 2010.

<http://www.consortium.ri.cmu.edu/ckagree/>

2 – Face Recognition with OpenCV (Open Source Computer Vision) – 2011 <http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html>

3 – Метод главных компонент <http://www.machinelearning.ru/wiki/index.php?title=метод_главных_компонент>

4 – Линейный дискриминант Фишера

<http://www.machinelearning.ru/wiki/index.php?title=Линейный_дискриминант_Фишера>

# 7. ПРИЛОЖЕНИЕ

imgExtract.py

**import** glob  
**from** shutil **import** copyfile  
emotions = [**"neutral"**, **"anger"**, **"contempt"**, **"disgust"**, **"fear"**, **"happy"**, **"sadness"**, **"surprise"**] *# Define emotion order*participants = glob.glob(**"source\_emotion\\\*"**) *# Returns a list of all folders with participant numbers***for** x **in** participants:  
 part = **"%s"** % x[-4:] *# store current participant number* **for** sessions **in** glob.glob(**"%s\\\*"** % x): *# Store list of sessions for current participant* **for** files **in** glob.glob(**"%s\\\*"** % sessions):  
 current\_session = files[20:-30]  
 file = open(files, **'r'**)  
  
 emotion = int(  
 float(file.readline())) *# emotions are encoded as a float, readline as float, then convert to integer.* sourcefile\_emotion = glob.glob(**"source\_images\\%s\\%s\\\*"** % (part, current\_session))[  
 -1] *# get path for last image in sequence, which contains the emotion* sourcefile\_neutral = glob.glob(**"source\_images\\%s\\%s\\\*"** % (part, current\_session))[  
 0] *# do same for neutral image* dest\_neut = **"sorted\_set\\neutral\\%s"** % sourcefile\_neutral[25:] *# Generate path to put neutral image* dest\_emot = **"sorted\_set\\%s\\%s"** % (  
 emotions[emotion], sourcefile\_emotion[25:]) *# Do same for emotion containing image* copyfile(sourcefile\_neutral, dest\_neut) *# Copy file* copyfile(sourcefile\_emotion, dest\_emot) *# Copy file*

datasetCreate.py

**import** glob  
  
**import** cv2  
  
faceDet = cv2.CascadeClassifier(**"HAAR/haarcascade\_frontalface\_default.xml"**)  
faceDet2 = cv2.CascadeClassifier(**"HAAR/haarcascade\_frontalface\_alt2.xml"**)  
faceDet3 = cv2.CascadeClassifier(**"HAAR/haarcascade\_frontalface\_alt.xml"**)  
faceDet4 = cv2.CascadeClassifier(**"HAAR/haarcascade\_frontalface\_alt\_tree.xml"**)  
  
emotions = [**"neutral"**, **"anger"**, **"contempt"**, **"disgust"**, **"fear"**, **"happy"**, **"sadness"**, **"surprise"**] *# Define emotions***def** detect\_faces(emotion):  
 files = glob.glob(**"sorted\_set\\%s\\\*"** % emotion) *# Get list of all images with emotion* filenumber = 0  
 **for** f **in** files:  
 frame = cv2.imread(f) *# Open image* gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) *# Convert image to grayscale  
  
 # Detect face using 4 different classifiers* face = faceDet.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5),  
 flags=cv2.CASCADE\_SCALE\_IMAGE)  
 face2 = faceDet2.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5),  
 flags=cv2.CASCADE\_SCALE\_IMAGE)  
 face3 = faceDet3.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5),  
 flags=cv2.CASCADE\_SCALE\_IMAGE)  
 face4 = faceDet4.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5),  
 flags=cv2.CASCADE\_SCALE\_IMAGE)  
 *# Go over detected faces, stop at first detected face, return empty if no face.* **if** len(face) == 1:  
 facefeatures = face  
 **elif** len(face2) == 1:  
 facefeatures == face2  
 **elif** len(face3) == 1:  
 facefeatures = face3  
 **elif** len(face4) == 1:  
 facefeatures = face4  
 **else**:  
 facefeatures = **""** *# Cut and save face* **for** (x, y, w, h) **in** facefeatures: *# get coordinates and size of rectangle containing face* print(**"face found in file: %s"** % f)  
 gray = gray[y:y + h, x:x + w] *# Cut the frame to size* **try**:  
 out = cv2.resize(gray, (350, 350)) *# Resize face so all images have same size* cv2.imwrite(**"dataset\\%s\\%s.jpg"** % (emotion, filenumber), out) *# Write image* **except**:  
 **pass** *# If error, pass file* filenumber += 1 *# Increment image number***for** emotion **in** emotions:  
 detect\_faces(emotion) *# Call functiona*

EmotionCLassifier.py

**import** glob  
**import** random  
  
**import** cv2  
**import** numpy **as** np  
  
emotions = [**"neutral"**, **"anger"**, **"disgust"**, **"fear"**, **"happy"**, **"surprise"**] *# Emotion list*fishface = cv2.face.EigenFaceRecognizer\_create() *# Initialize fisher face classifier  
#EigenFaceRecognizer\_create 22% correct  
#FisherFaceRecognizer\_create 66% correct*data = {}  
**def** get\_files(emotion): *# Define function to get file list, randomly shuffle it and split 80/20* files = glob.glob(**"dataset\\%s\\\*"** % emotion)  
 random.shuffle(files)  
 training = files[:int(len(files) \* 0.8)] *# get first 80% of file list* prediction = files[-int(len(files) \* 0.2):] *# get last 20% of file list* **return** training, prediction  
**def** make\_sets():  
 training\_data = []  
 training\_labels = []  
 prediction\_data = []  
 prediction\_labels = []  
 **for** emotion **in** emotions:  
 training, prediction = get\_files(emotion)  
 *# Append data to training and prediction list, and generate labels 0-7* **for** item **in** training:  
 image = cv2.imread(item) *# open image* gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) *# convert to grayscale* training\_data.append(gray) *# append image array to training data list* training\_labels.append(emotions.index(emotion))  
 **for** item **in** prediction: *# repeat above process for prediction set* image = cv2.imread(item)  
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 prediction\_data.append(gray)  
 prediction\_labels.append(emotions.index(emotion))  
  
 **return** training\_data, training\_labels, prediction\_data, prediction\_labels  
**def** run\_recognizer():  
 training\_data, training\_labels, prediction\_data, prediction\_labels = make\_sets()  
 print(**"training eigen face classifier"**)  
 print(**"size of training set is:"**, len(training\_labels), **"images"**)  
 fishface.train(training\_data, np.asarray(training\_labels))  
  
 print(**"predicting classification set"**)  
 cnt = 0  
 correct = 0  
 incorrect = 0  
 **for** image **in** prediction\_data:  
 pred, conf = fishface.predict(image)  
 **if** pred == prediction\_labels[cnt]:  
 correct += 1  
 cnt += 1  
 **else**:  
 cv2.imwrite(**"difficult\\%s\_%s\_%s.jpg"** % (emotions[prediction\_labels[cnt]], emotions[pred], cnt),  
 image) *# <-- this one is new* incorrect += 1  
 cnt += 1  
 **return** ((100 \* correct) / (correct + incorrect))  
e1 = cv2.getTickCount()  
*# Now run it*metascore = []  
**for** i **in** range(0, 1):  
 correct = run\_recognizer()  
 print(**"got"**, correct, **"percent correct!"**)  
 metascore.append(correct)  
print(**"\n\nend score:"**, np.mean(metascore), **"percent correct!"**)  
  
e2 = cv2.getTickCount()  
t = (e2 - e1) / cv2.getTickFrequency()  
print(**'work time='**, t)

noiseimage.py

**import** glob  
**import** random  
**import** cv2  
**import** numpy **as** np  
**from** matplotlib **import** pyplot **as** plt  
emotions = [**"neutral"**, **"anger"**, **"disgust"**, **"fear"**, **"happy"**, **"surprise"**] *# Emotion list*fishface = cv2.face.EigenFaceRecognizer\_create() *# Initialize fisher face classifier  
#EigenFaceRecognizer\_create 22% correct  
#FisherFaceRecognizer\_create 66% correct*imSize = 350  
data = {}  
mean = 0  
sigma = 50  
**def** get\_files(emotion): *# Define function to get file list, randomly shuffle it and split 80/20* files = glob.glob(**"dataset\\%s\\\*"** % emotion)  
 *#out = cv2.resize((30, 30))  
 #add\_normal\_noise(files, 0, 1)* random.shuffle(files)  
 training = files[:int(len(files) \* 0.8)] *# get first 80% of file list* prediction = files[-int(len(files) \* 0.2):] *# get last 20% of file list  
 #training = cv2.resize(files[:int(len(files) \* 0.8)], (30, 30))* **return** training, prediction  
**def** make\_sets():  
 training\_data = []  
 training\_labels = []  
 prediction\_data = []  
 prediction\_labels = []  
 **for** emotion **in** emotions:  
 training, prediction = get\_files(emotion)  
 *# Append data to training and prediction list, and generate labels 0-7* **for** item **in** training:  
 nores\_image = cv2.imread(item) *# open image* image = cv2.resize(nores\_image, (imSize, imSize))  
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) *# convert to grayscale* noise = np.random.normal(mean, sigma, gray.shape)  
 img = gray.astype(noise.dtype)  
 img += noise  
 training\_data.append(gray) *# append image array to training data list* training\_labels.append(emotions.index(emotion))  
 **for** item **in** prediction: *# repeat above process for prediction set  
 #image = cv2.imread(item)* nores\_image = cv2.imread(item) *# open image* noise = np.random.normal(mean, sigma, image.shape)  
 image = cv2.resize(nores\_image, (imSize, imSize))  
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) *# convert to grayscale* noise = np.random.normal(mean, sigma, gray.shape)  
 img = gray.astype(noise.dtype)  
 img += noise  
 *#gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)* prediction\_data.append(img)  
 prediction\_labels.append(emotions.index(emotion))  
 **return** training\_data, training\_labels, prediction\_data, prediction\_labels  
  
  
  
**def** run\_recognizer():  
 training\_data, training\_labels, prediction\_data, prediction\_labels = make\_sets()  
  
  
 print(**"size of training set is:"**, len(training\_labels), **"images"**)  
 print(**"size of prediction set is:"**, len(prediction\_labels), **"images"**)  
 fishface.train(training\_data, np.asarray(training\_labels))  
  
 print(**"predicting classification set"**)  
 cnt = 0  
 correct = 0  
 incorrect = 0  
 **for** image **in** prediction\_data:  
 pred, conf = fishface.predict(image)  
 **if** pred == prediction\_labels[cnt]:  
 correct += 1  
 cnt += 1  
 **else**:  
 cv2.imwrite(**"difficult\\%s\_%s\_%s.jpg"** % (emotions[prediction\_labels[cnt]], emotions[pred], cnt),  
 image) *# <-- this one is new* incorrect += 1  
 cnt += 1  
  
 **return** ((100 \* correct) / (correct + incorrect))  
  
  
e1 = cv2.getTickCount()  
  
*# Now run it*metascore = []  
imSizeArr = []  
endscore = []  
*# for i in range(0, 4):  
# imSize -= 50  
# correct = run\_recognizer()  
# print("got", correct, "percent correct!")  
# metascore.append(correct)  
# imSizeArr.append(imSize)***for** i **in** range(0, 6):  
 imSize -= 50  
 **for** j **in** range(0, 8):  
 correct = run\_recognizer()  
 print(**"got"**, correct, **"percent correct!"**)  
 metascore.append(correct)  
 print(**"\nend score:"**, np.mean(metascore), **"percent correct!\n\n"**)  
  
 endscore.append(np.mean(metascore))  
 imSizeArr.append(imSize)  
  
 metascore = []  
  
*#print("\n\nend score:", np.mean(metascore), "percent correct!")*print(**'\n\nImage Size: '**, imSizeArr)  
print(**'\n\nCorrect: '**, endscore)  
e2 = cv2.getTickCount()  
t = (e2 - e1) / cv2.getTickFrequency()  
print(**'\n\nwork time='**, t)  
plt.figure(**'EigenFace Classification'**)  
plt.plot(imSizeArr, endscore, **'g'**)  
plt.show()