Implementation and Testing of Local Binary Pattern Histogram as a Method of Face Recognition

by

Rafi Boghosians

Master's Degree, State Engineering University of Armenia, 2011

A thesis submitted in partial satisfaction of

the requirements for the degree of

Master of Science

in

Computer & Information Science

in the

COLLEGE OF SCIENCE AND ENGINEERING

of the

AMERICAN UNIVERSITY OF ARMENIA

Supervisor: Suren Khachatryan

Signature: ___________________ Date:_____________

Committee Member: ____________________

Signature: ___________________ Date:_____________

Committee Member: ____________________

Signature: ___________________ Date:_____________

Committee Member: ____________________

Signature: ___________________ Date:_____________

Implementation and Testing of Local Binary Pattern Histogram as a Method of Face Recognition

Contents

Abstract

Implementation and Testing of Local Binary Pattern Histogram as a Method of Face Recognition

Author: Rafi Boghosians

The popularity of the cameras in smart electronic devices led the industries to use them more efficiently. Facial recognition research is one of the favorite topics among practitioners and researchers and is a key to the future of smart technologies. This study is an attempt to indicate the effectiveness of existing facial recognition algorithms using OpenCV library and C# programming language. This thesis aims to investigate several facial recognition algorithms and make comparisons in respect of their accuracy. We will use Viola-Jones Face Detection algorithm for detecting the face, and the Eigenfaces, Fisherfaces, Local Binary Pattern Histogram algorithms for recognizing the face.

The thesis covers the complete process of face recognition, including face detection, preprocessing of images, the comparison of the algorithms mentioned above and the real-time application of Local binary pattern histogram.

We will discuss the concept of each algorithm, and comparative analysis will reveal the most accurate one.

The development of the test cases will indicate that among the compared facial recognition algorithms the Local Binary Pattern Algorithm has the highest accuracy to identify faces.

Keywords:

Local Binary Pattern Histogram, Eigenface, Fisherface, Face Detection, Facial Recognition, EmguCV, Computer Vision, Viola Jones, OpenCV

Abbreviations:

LBPH - Local Binary Patterns Histograms

PCA - Principal Components Analysis

LDA - Linear Discriminant Analysis

OpenCV – Open Source Computer Vision Library

[illegible]

1. Introduction

Nowadays cameras are widely spread in our society. We can see them in streets, in restaurants, in supermarkets and other public places. These cameras are more often used for security purposes. Cameras are everywhere. The number of devices per person is increasing, and they usually possess a camera. For example, all smartphones and tablets have one. People use cameras for recording themselves, capturing wonderful moments and taking pictures of beautiful sights. However, it is possible to use them for many other goals.

Through face detection and recognition, cameras have got commercial and security applications in identity validation and recognition. An example of a software that uses facial recognition for security purposes is MasterCard Identity Check Mobile app [1]. This application verifies online payments through facial recognition. Businesses can make online payments without cards or passwords. App users can verify their payments by a picture of their faces. In addition to verifying a payment, facial biometrics can also be integrated with physical objects. In future, consumers may be able to get into their houses, cars and other secure locations by merely looking at them. Jaguar is already working on its face and gait recognition system called walking gait ID [2] – a potential parallel to facial recognition technology.

We can see applications of facial recognition systems in the healthcare industry as well. For instance, AiCure [3], an AI company uses computer vision and facial recognition technology to improve medication adherence practices. The company's software is delivered through an app, which reportedly performs three main functions, it identifies the patient, the prescribed medicine and can visually confirm if the patient has ingested the medicine.

In the marketing sector, application of facial recognition technology can support advertisers to get closer to their target markets and improve customer loyalty. For example, the app Facedeals [4] integrates facial recognition with customers' Facebook profiles aiming to inform customers about special offers from businesses they often visit. Specifically, cameras at the business entrance would recognize customers as they enter. At the same time, the customer would receive a notification of a special offer based on his/her Facebook "Like" history.

Currently one of the biggest companies that use computer vision is Google. Google has the most applicable and most comprehensive image search on the web. The search engine of Google can show images related to a search even if they have not been tagged. This is because the engine can identify what is shown in the image. Another example is the facial recognition performed by

Facebook which is called when a user uploads a new photo. Facebook detects faces in the picture and recognizes the persons. Then, it suggests the user tag him/her.

All the examples mentioned above use computer vision, but also, they use machine learning algorithms. These algorithms are constructed so that a machine can learn from and make predictions on data provided. The larger the provided data is, the better the results are. Google and Facebook provide the most precise information due to the amount of data. They collect a vast amount of training data which increases the efficiency of machine learning.

So, the companies possess an enormous amount of data and construct robust algorithms to perform the facial recognition accurately. As we can see, some of these algorithms are public. For instance, Google made public their method of face recognition and clustering called FaceNet [5]. Later, Facebook did the same with their facial recognition system DeepFace [6]. These algorithms use different approaches and provide a certain level of accuracy. So, what are the distinctive characteristics of these approaches? Is it possible to recognize a face accurately through these algorithms? Can we measure and compare the accuracy and find out whether the results are reliable?

2. Goals and Objectives

This thesis aims to study the different existing facial recognition algorithms. The evaluation is limited to those algorithms that are mostly used by the commercial sector and are open source. Every algorithm will be trained with the identical data set. Then, we will use a webcam to test those algorithms. Comparative analysis of the test results will indicate the most accurate face recognition algorithm. Subsequently, the software implemented by C# Programming language using .Net Windows Form Application and EmguCV library will demonstrate the reliability of the chosen algorithm.

For face detection, we will use Viola-Jones Object Detection technique, which is based on the idea that rather than identifying whether the image contains a face, we can more quickly determine whether the picture does not contain a face because eliminations can be done quickly, although recognition of faces will require more time.

For face recognition, we will examine Fisherfaces, Eigenfaces and Local Binary Patterns Histograms algorithms.

We will describe the steps of the recognition process briefly to provide a sufficient understanding of the importance of each step. The lightning and other conditions affect the quality of the image and can play a significant role in the test results. Therefore, we will provide information about those conditions as well.

The most complicated step is teaching the machine to recognize faces. We should note that the training data should contain as many pictures as possible taken from different angles. Using the provided data and machine learning algorithms, the computer will learn how to differentiate faces. These machine learning algorithms use different approaches, such as statistical approach or search for patterns. The algorithms will be compared, and their strengths and weaknesses will be discussed.

Finally, we will demonstrate a desktop application, which will show the reliability and effectiveness of the chosen Face Detection and Recognition method.

3. Application Specifications

For developing the application, we use EmguCV [7] library which is cross-platform .Net wrapper to the standard OpenCV image processing library. It allows the OpenCV functions to be called from .NET compatible languages such as C#, VB, IronPython, etc. The wrapper can be compiled by Visual Studio, Xamarin Studio, and Unity; it can run on Windows, Linux, Mac OS X, iOS, Android and Windows Phone.

The application is written in C# language on the .NET framework 4. It is based on Windows Form application which is based on MVVM design pattern.

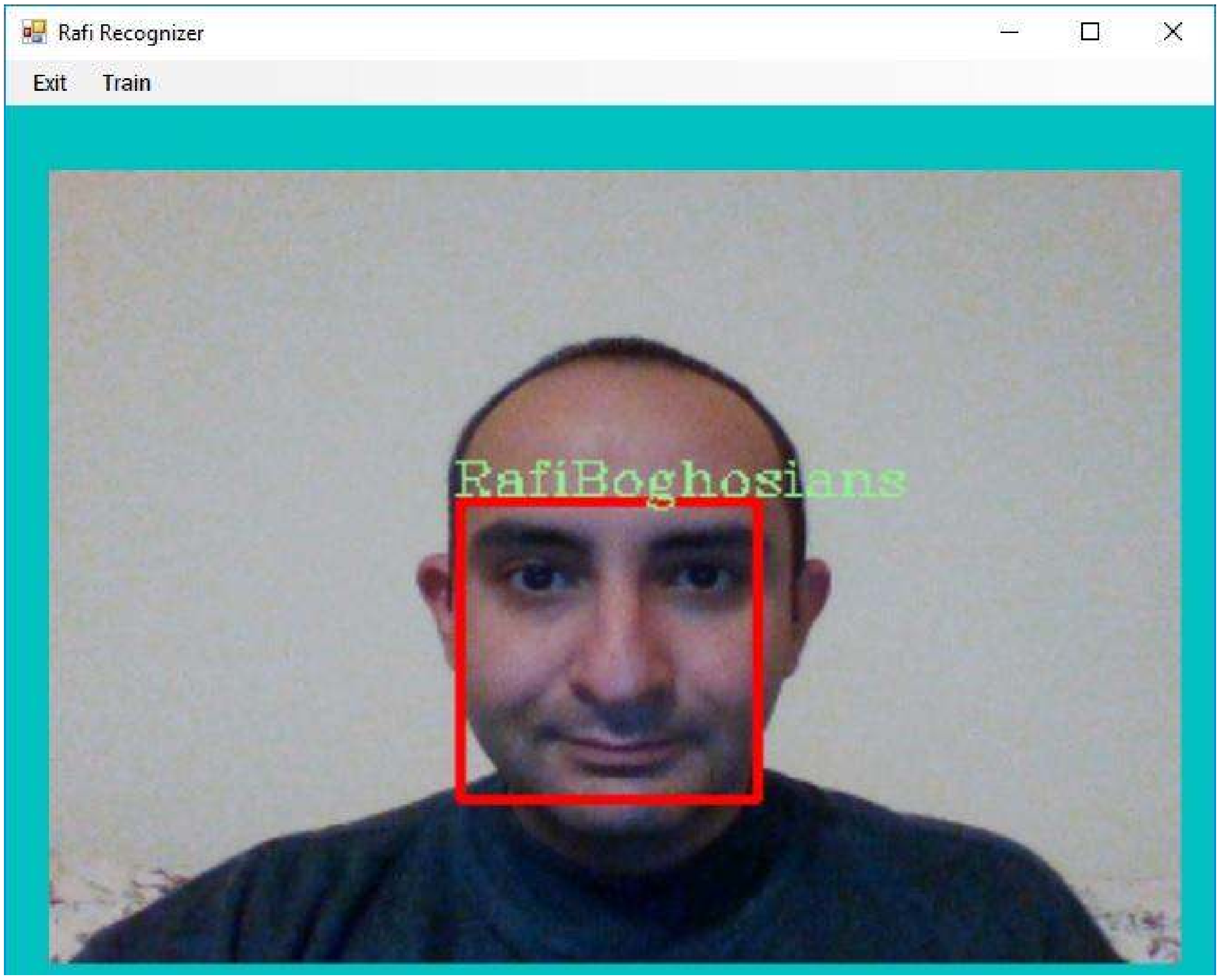
The whole application consists of two forms (GUI) and two C# classes.

- *Main Form.cs*
  - Is responsible for displaying the result of face recognition.
- *Traning Form.cs*.
  - Is responsible for training the data set.
- *Program.cs*
  - The main entry point for the application.
- *Classifier_Train.cs*
  - Saving and Loading datasets (Just for more readability of the code).

3.1. Distribution of classes

Main Form.cs

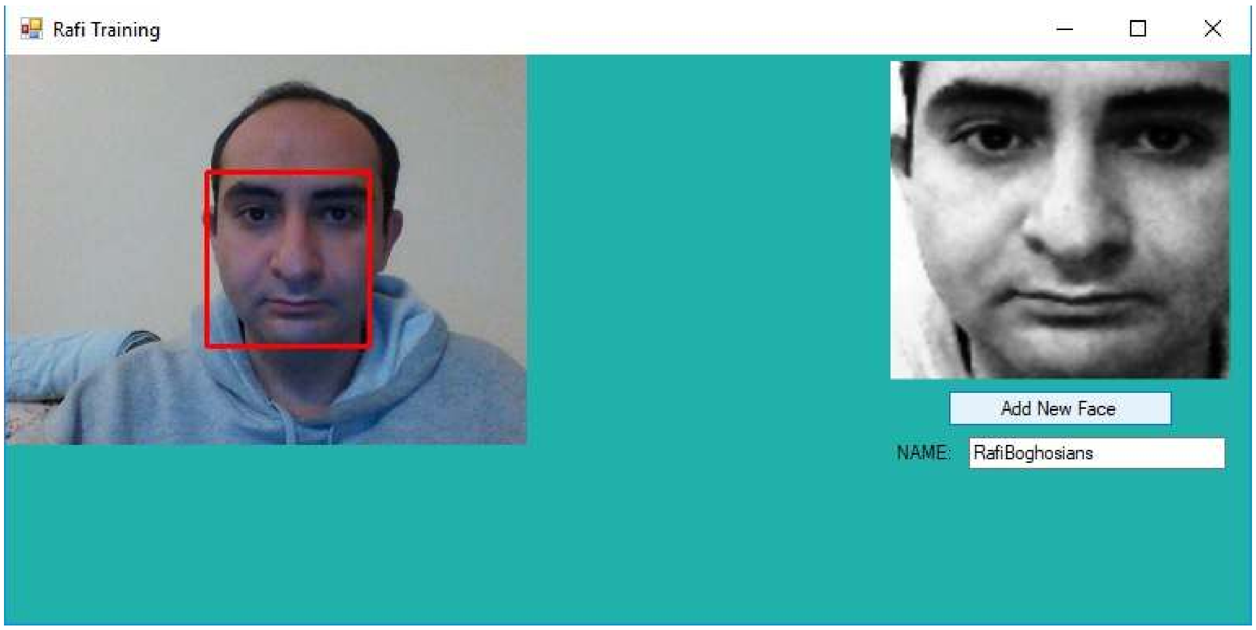
- Declaration of the variables related to the displaying the result of recognition.
- Loading the training data.
- Initializing the *FrameGrabber* event.
  1. Get the current frame from the device which captures the picture
  2. Convert it to Grayscale
  3. Detect the face using Haar-cascade EmguCV classifier
  4. Focusing on the face and painting a Red color rectangle around the face
  5. Loading Recognition of the face using EmguCV face recognizer
  6. Display the label for each detected and recognized face



Traning Form.cs

- Declaration of the variables related to training data set.
- Initializing the *FrameGrabber* event.
    1. Get the current frame from the device which captures the picture
    2. Convert it to Grayscale
    3. Detect the face using Haar-cascade classifier of EmguCV
    4. Focusing on the face and painting a Red color rectangle around the face
- Saving the new face.
    1. Each image is saved by generating random number so, we will be able to train our data set with the same person name but different pictures

2. The default location for training data is within the TrainedFaces. The folder also includes a single XML file that includes tags for the person name. The XML file has the following structure:



XML file example

```
<FACE>
   <NAME>Rafi</NAME>
   <FILE>face_Rafi_909797833.jpg</FILE>
</FACE>
<FACE>
   <NAME>Rafi</NAME>
   <FILE>face_Rafi_1560838153.jpg</FILE>
</FACE>
<FACE>
   <NAME>Rafi</NAME>
   <FILE>face_Rafi_64394826.jpg</FILE>
</FACE>
<FACE>
```

```
    <NAME>Rafi</NAME>
    <FILE>face_Rafi_715435146.jpg</FILE>
</FACE>
```

The Gray Scale image examples

face_RafiBoghosians_ 1638985040 | face_RafiBoghosians_ 1786709253 | face_RafiBoghosians_ 2133106717 | face_RafiBoghosians_ 566139861 | face_RafiBoghosians_ 765022211 | face_RafiBoghosians_ 507226928

3.2. The logical flow of the application

The new FaceRecognizer is a global constructor that allows Eigen, Fisher, and LBPH classifiers to be used together. The class combines common method calls between the classifiers.

FaceRecognizer recognizer = new LBPHFaceRecognizer(radius, neighbors, grid_x, grid_y, threshold);

1. Firstly, we need to create an variable of a Reference type `FaceRecognizer recognizer`. `FaceRecognizer` is a global constructor that allows Eigen, Fisher, and LBPH classifiers to be used together. The class combines common method calls between the classifiers.

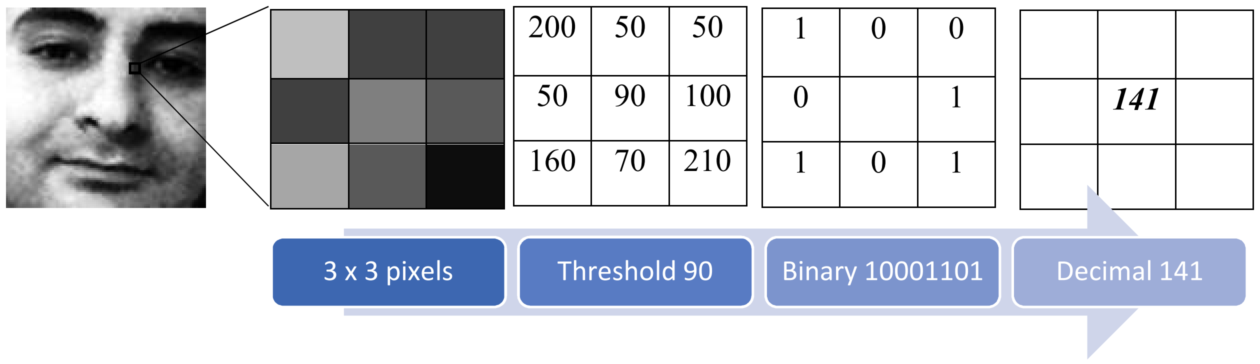
```
using Emgu.Util;

namespace Emgu.CV
{
    public abstract class FaceRecognizer : UnmanagedObject
    {
        protected FaceRecognizer();

        public void Load(string fileName);
        public PredictionResult Predict(IImage image);
        public void Save(string fileName);
        public void Train(IImage[] images, int[] labels);
        protected override void DisposeObject();

        public struct PredictionResult
        {
            public int Label;
            public double Distance;
        }
    }
}
```

2. Secondly, we should create an object of type `new LBPHFaceRecognizer (1, 8, 8, 8, 100).` If we do not set any parameters, the program will use its default settings. Here is a description of the five parameters.

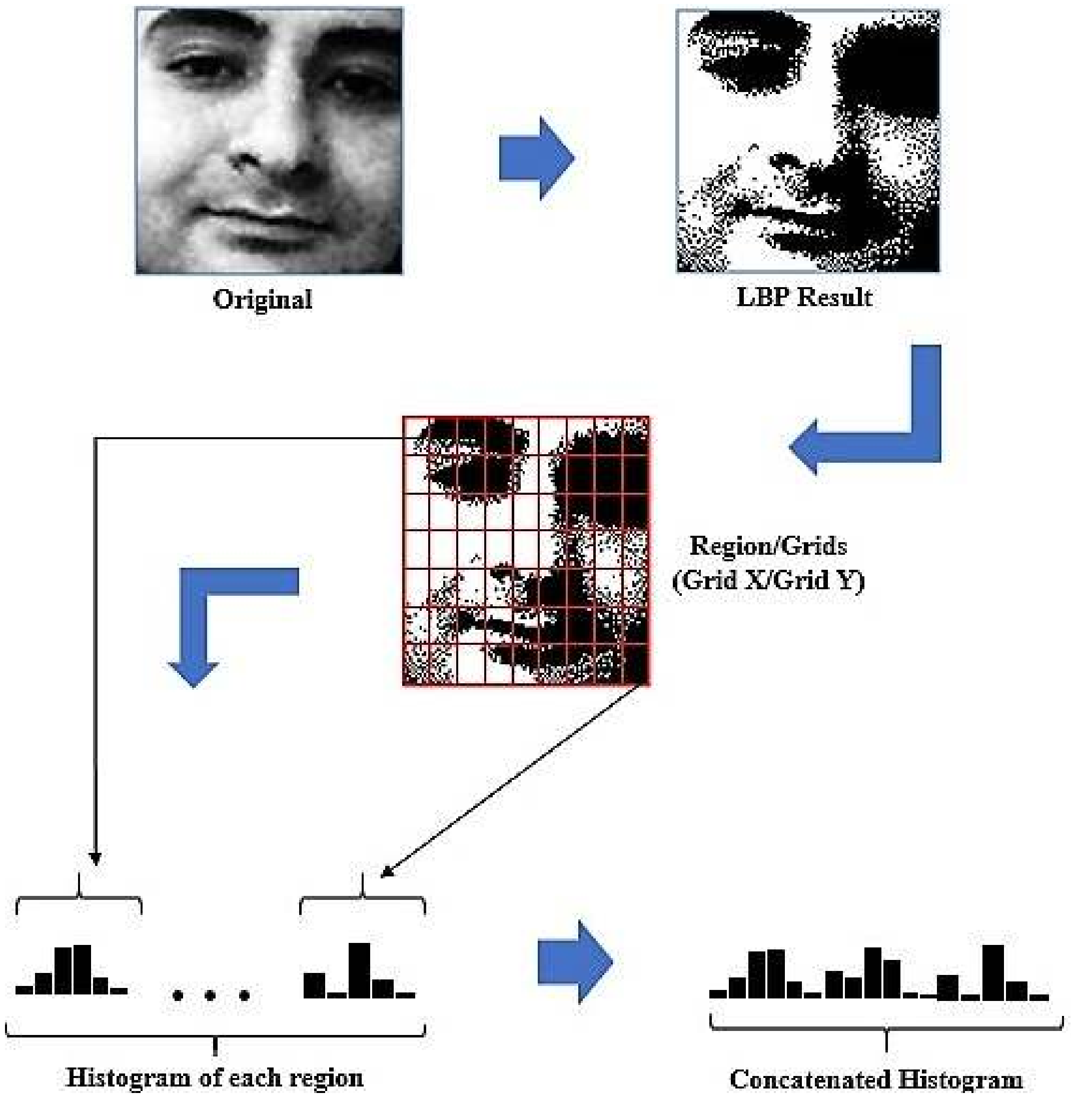
- Radius – The radius used for building the Circular Local Binary Pattern. The default value is 1.
- Neighbors – The amount of sample points to construct a Circular Local Binary Pattern. The value suggested by OpenCV Documentations is '8' sample points. Acknowledged: the more sample points are included, the higher the computational cost is.
- GridX– The number of cells in the horizontal direction, 8 is a standard value used in EmguCV documentations. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector.
- GridY – The number of cells in the vertical direction, 8 is a standard value used in EmguCV documentations. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector.
- Threshold – The threshold applied in the prediction. If the distance to the nearest neighbor is larger than the threshold, this method returns *-1*.

If the Distance calculated is above this value, the *Predict()* method will return a *-1* value indicating an unknown.

3. Third, we should train the algorithm. To do that we need to call the `Train(IImage[] images, int[] labels);` function passing a slice of images and a slice of labels by the parameters. All images must have the same size. The tags (labels) are used as IDs for the images, so if you have more than one model of the same face, the labels should be the same.
4. The `Train()` function first checks if all images of the face have the same size. If somehow there is an image which does not have the same size, the method will return an error, and thus, the algorithm will not be trained.
5. The next step is applying the basic LBP operation by changing each pixel based on its neighbors using a default radius of one with eight neighbors. The example of LBP process is described below:



6. After processing the LBP operation, we extract the histograms of each image based on the number of grids (GridX and GridY) which are passed by parameter. After obtaining the histogram of each region, we collect all the individual histograms and create a new one that will represent the whole image.



7. In this stage, we have all the necessary stored data of faces, labels, and histograms.
8. We can say that the algorithm is already trained, and we can Predict a new face.
9. To predict a new image, we need to call the `PredictionResult Predict(IImage image);` function and pass the image as parameter. The `Predict()` function will extract the histogram from the new model and will compare it to the histograms that are already stored and will return the label and distance corresponding to the closest histogram. Note: It uses the Euclidean distance metric as the default metric to compare the histograms. The closer to zero is the distance; the higher is the confidence.

The `Predict()` function returns 2 values:

- `int Label`: The label corresponding to the predicted image.

- `double Distance`: The distance between the histograms from the input test image and the matched image (from the training set).

We use the `Label` to check if the algorithm has correctly predicted the image. But in a real-world application, it is not feasible to manually verify all images, so we can use the `Distance` to figure out if the algorithm has predicted the image correctly.

4. Methodology and Theoretical Background

OpenCV

OpenCV (Open Source Computer Vision) is free for use, cross-platform programming library mainly aimed at real-time computer vision [8]. OpenCV, initially developed by Intel, was later supported by Willow Garage which is robotics research lab and technology incubator and is now maintained by Itseez.

OpenCV is written in C++, and its interface is in C++. There are also bindings in Java, Python, and MATLAB. However, the API for these interfaces can be found in other languages such as C#, Perl, Haskell, and Ruby [9].

OpenCV's application areas include:

Egomotion estimation: Egomotion is defined as the 3D motion of a camera within an environment [10]. An example of egomotion estimation can be estimating a car's moving position relative to street signs or lines on the road being observed from the vehicle itself.

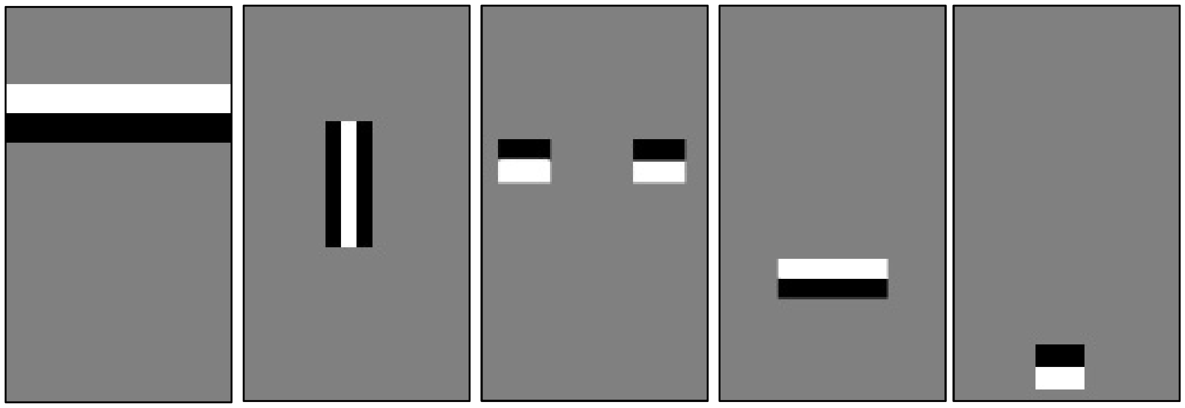
Facial recognition system: A facial recognition system originates from a purpose-built union of high-end hardware components and efficient software to automatically distinguish or verify an individual from a digital image, as needed in many security and inspection installations. The process of identification is done by comparing the facial features extracted from an image with those features previously stored in a facial database.

Gesture recognition: Gesture recognition aims to interpret human gestures using mathematical algorithms. Gestures can originate from body motion but mostly originate from the hand or face. Currently, the focus of this field includes hand gesture recognition and emotion recognition [11].

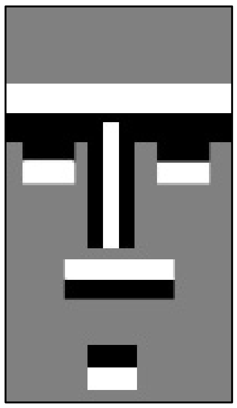
4.1. Face Detection

As can be supposed, detecting a face is more straightforward than recognizing a face of a specific human. To be able to find out that a particular image contains a face (or several), we need to be able to describe the overall structure of a face. Fortunately, human faces do not significantly vary from each other; we all have eyes, nose, mouth, and chin. So, all of these compose the common structure of a face.

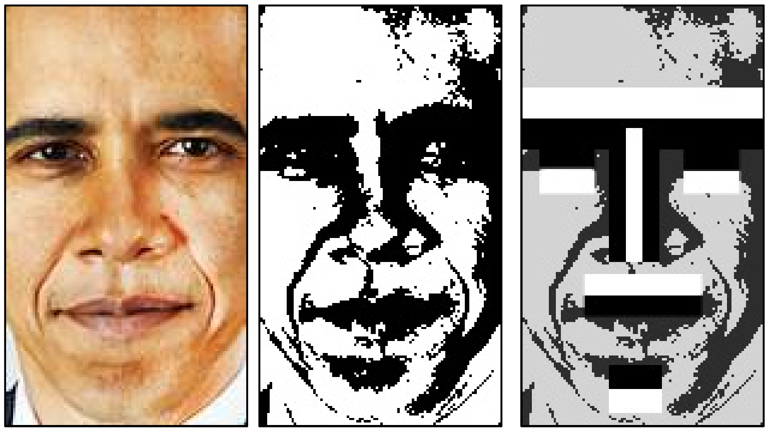
Consider the following five figures:



Each of these elements represents a general feature of a human face. So, combining all these, we will receive something that resembles a face.



We can conclude whether the image contains a face or not. Note that this does not have to be a precise match. We need to know whether, roughly, each of these elements corresponds to some part of the image. The technique is called Template Matching.



This method is designed in such a way that it first checks the rough features and only if these features match, it will continue to the next iteration. In each iteration, it can quickly ignore areas of the picture which do not match a face and keep checking areas which it is not sure about. In other words, rather than finding whether the image contains a face, we can more quickly determine if the picture does not contain a face, since eliminations can be done quickly, while approval of faces will require more time. This process is called *cascading*. The method depicted above is an over-simplified description of the Viola-Jones method (also known as Haar cascades [12]).

The Viola-Jones object detection framework [13] is the first object detection framework to provide competitive object detection rates in real-time which is published in the paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. The primary motivation of the problem was face detection, though the algorithm can be trained to detect a variety of object classes, such as cars, people, furniture, etc. The approach is based on Machine Learning where a cascade function is trained from lots of positive and negative images. Then, it is used to detect objects in other models.

The primary goal was the detection of faces in an image. A human can do this quickly, but a computer needs precise instructions and computation. The Viola-Jones algorithm requires the full view frontal of faces. Hence, to be detected, the entire front side of the face must point towards the camera and should not be inclined to either side. Although it seems, these constraints could somehow reduce the algorithm's utility, since the detection step is usually followed by a recognition step, in practice these limits on pose are quite acceptable.

4.2. Face Recognition

Face recognition is an easy task for people. Experiments [14] show that even three-day-old infants can distinguish familiar faces. Can a machine perform this? How does our brain analyze and encode an image? David Hubel’s and Torsten Wiesel’s brain research shows that our brain has nerve cells which are responding to specific local features of a scene, such as angles, edges, lines or movements. Still, we don't see the world in scattered pieces, which means, our brain somehow combines the various sources of information into useful patterns. Automatic face recognition is performed by extracting meaningful features from an image, putting them into a useful representation and performing some classifications on them.

Automated face recognition systems have developed over time. They have used different approaches to recognize faces, and these approaches have evolved to reach a higher level of accuracy. One of the facial identification methods uses geometric features of a face. The first [15] automated facial recognition systems used facial marks (position, the shape of eyes, ears, nose, etc.) to build a feature vector (distance, the angle between the marks). The system then calculated the Euclidean distance between feature vectors of an image and identified the face. The drawback of this method was that despite the use of robust algorithms it was still very complicated to precisely register the facial marks. Further experiments have carried out using geometric face recognition methods. Some of those experiments [16] used large datasets and a 22-dimensional feature vector. However, it became obvious, that geometrical features alone may not carry sufficient information for face recognition.

Recently various methods appeared for local feature extraction. To avoid the high-dimensionality of the input data, only local regions of face images are described. Consequently, the obtained features are strong against partial obstruction, lighting, and small sized sample. Local feature extraction uses the following algorithms: Gabor Wavelets [17], Discrete Cosine Transform [18], and Local Binary Patterns [19]. Researchers still do not answer the question what the best way to preserve spatial information when applying a local feature extraction is?

The difference between face recognition and detection is that in detection we need to determine if there is some face in the picture, but in recognition, we need to find out whose face it is.

OpenCV provides three methods of face recognition:

- Eigenfaces
- Fisherfaces

- Local Binary Patterns Histograms (LBPH).

All three algorithms do the recognition by comparing the face with some training set of already recognized faces. In the training set, we provide the algorithm of faces and tell it to who they belong. Then, the algorithm uses the training set to recognize unknown faces. Each of the methods mentioned above uses the training set a little differently. The first two algorithms (Fisherfaces and Eigenfaces) find a mathematical description of the most dominant elements of the training set as a whole, while LBPH analyzes each face in the training set separately and independently.

4.2.1. Eigenfaces

Eigenfaces is a method of performing facial recognition based on a statistical approach. Sirovich and Kirby [20] first developed the concept of utilizing principal components to express human faces and then used by Turk and Pentland [21] for face detection and recognition. This method aims to extract the principal components which mostly affect the variation of the images. This is a holistic (as opposed to a parts-based or feature-based) approach: the method for predicting a face is based on the entire training set. There is no specific analysis of images from two different classes. A class represents a person. Pre-processed pictures with grayscale are needed to train the machine learning algorithm. Each pixel of an image represents one dimension, it means a 100 x 100 pixels image is defined into 100 x 100= 1000 dimensions. Eigenfaces uses Principal Component Analysis (PCA) to reduce the number of dimensions while maintaining the most significant information. The training part of Eigenfaces is to calculate the eigenvectors and the related eigenvalues of the covariance matrix of the training set.
The motivation of Eigenfaces is as follow:

- Takeout the relevant facial information, which may or may not be directly related to the human intuition of facial features such as the eyes, nose, and lips. One way to do so is to capture the statistical variation between face images.
- Represent face images effectively. As means to reduce the space and computation complexity, each face image can be represented by a small number of parameters.

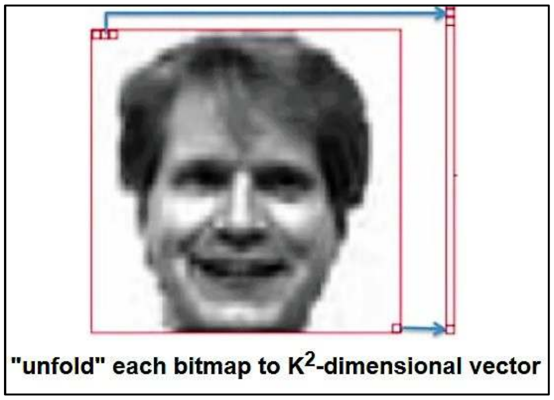
Here is step by step description of how eigenfaces work.
Eigenface do PCA on bitmap images of human faces, so we have a dataset.

input: dataset of N face images

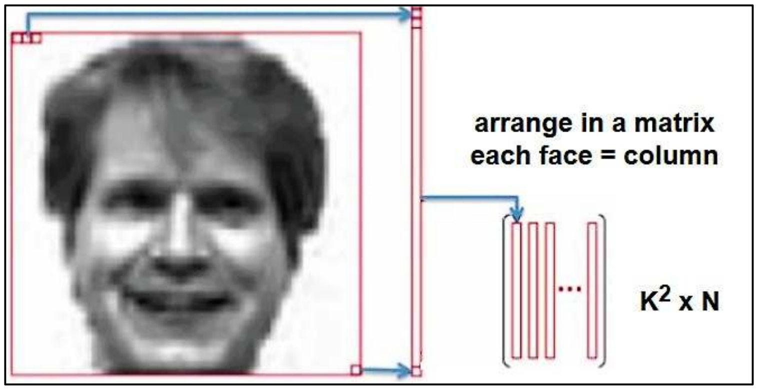
Every bitmap face here is a data instance. Here we can see an example of data instance.

face: k x k bitmap of pixels

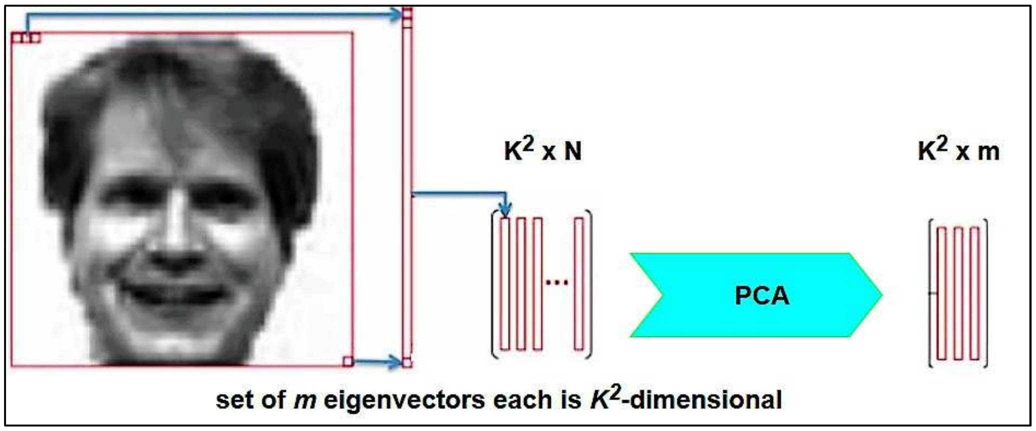
We assume that it is a k x k grayscale image. So, each pixel here is just a number between 0 and 255 which represents the level of grayness. Now we can take the amount of each pixel to form a single vector out of each instance. Since vectors or some attributes must represent all of our instances, we can unfold the bitmap into one large vector. So, we are going to take the first row of pixels, and that will become the primary *K* attributes in our vector, the second row becomes the second K, and the last row becomes the last K. Thus, we will end up with a vector that has K squared dimensions.



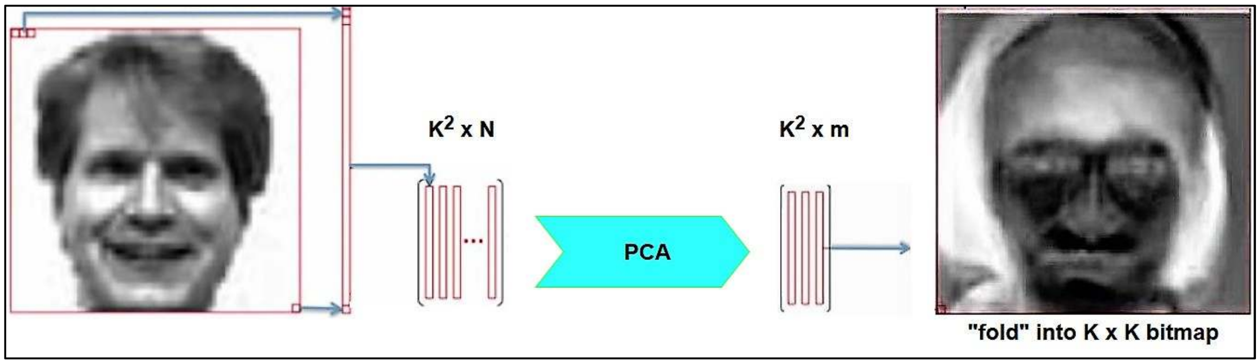
The above is an example of one instance. After doing this operation for every instance of our dataset, we will end up with a matrix which has K^2 rows. So, these are our new attributes of pixels, and we have N columns that represent the individual instances of our whole dataset.



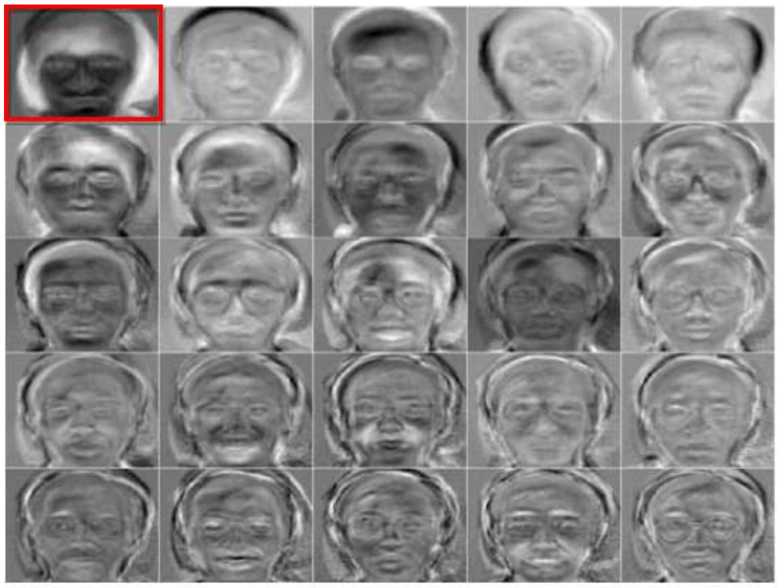
In this stage, we do PCA on our data. As a result, we will have specific number m of these new dimensions of eigenvectors with the highest eigenvalues.



It is interesting that we are going to have *m* columns which are our new dimensions, but each column is K^2 dimensional. So, each column substantially corresponds to the original positions of pixels in a bitmap. Since the attributes here are just pixels and we have K^2 of them. After doing PCA we still have K^2, each one of the eigenvectors has K^2 dimensions which are as many as pixels. As attributes should be numbers, we can take and plot them and see what they look like. We choose one of the eigenvectors, it has K^2 numbers and folds it back into a bitmap. So, we take the first K attributes make it into a row of pixels, make it to the second row of pixels and all the way down to the end. And as a result, we will end up with something like a ghostly face.



Now, what is our new image? It indeed does not look like an average face of any kind, and it should not be an average face since in the first step of PCA we subtract the mean, so we have subtracted the average face. Thus, this bitmap is the first principal component which is showing the most prominent deviation from the mean in this dataset. The mean is prototypical average looking face, and this representation of the face is the dimension along which people seem to vary the most from the mean. In the image below we can see eigenvectors of the whole dataset. The feature vectors can later be used for classification. The number of Eigenfaces to use for constructing the feature vectors is a free parameter of the technique.


eigenvectors of dataset

Eigenface is a practical method for face recognition. Thanks to the simplicity of its algorithm, we could implement the Eigenface recognition system easily. Likewise, it is efficient for processing in both time and space. PCA reduces the dimension size of an image considerably in a short period. Eigenface has high accuracy near 90% for frontal faces, though, there has a high correlation between the training data and the input data. The accuracy of Eigenface depends on many factors. Since it takes the pixel value as a comparison for the projection, the precision would decrease with changing light intensity. Apart from, orientation and scale of an image will affect the accuracy significantly. Preprocessing of the image is necessary to achieve a satisfactory result. The Eigenface method was an essential step towards appearance-based recognition in computer vision. However, the technique is sensitive to variations in lighting, scale, pose, facial expression, and occlusion. To work effectively, the face must be presented in frontal view, at the appropriate scale, in similar lighting conditions, in a defined (typically neutral) expression, and unconcluded. The other disadvantage of this approach is that finding the eigenvectors and eigenvalues are time-consuming.

4.2.2. Fisherfaces

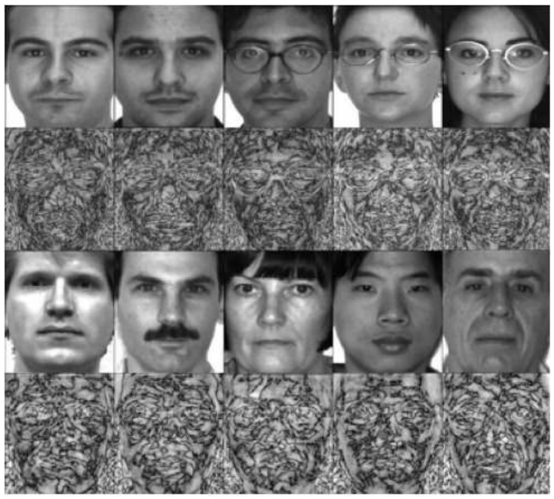
One way to illustrate the input data is by determining a subspace which represents most of the data variance as we already discussed in eigenface approach. This process can be done by using

Principal Components Analysis (PCA). When applied to face images, PCA produces a set of eigenfaces. These eigenfaces are the eigenvectors associated to the most significant eigenvalues of the covariance matrix of the training data. PCA approach is indeed a powerful way to represent the data. It guarantees the data variance is preserved while excluding unnecessary existing correlations between the original features (dimensions) in the sample vectors.

Fisherfaces [22] is similar to Eigenfaces in that it performs a linear mapping from the high dimensional image space to a lower dimensional subspace. The major difference is that Fisherfaces uses linear discriminant analysis (LDA) instead of PCA. LDA is like PCA, but it focuses on maximizing the separatability among known categories.

The idea here is that LDA keeps information that differentiates classes and rejects data that accounts for variations within single classes. Similar to PCA, the approach produces a set of models called Fisherfaces. Feature vectors composed of the contributions of each Fisherface to the reconstruction of the image.

The intuition here is that instead of looking at linear combinations of pixels that explain the variance in the data which is called Eigenfaces, the Fisherfaces looks for linear combinations of pixels that define the variance between people. That means if something is useful for describing the difference between various instances of the same person, it won't count, wheres if something is helpful to discriminate between different people - it will. It's quite intuitive. Think of an example where we try to distinguish between different people based on weight and height, and we have multiple measurements of the same people every couple of days. "eigenfaces" might consider weight to be a useful feature because it discriminates between repeated measurements. Fisherfaces would look on to the height signal because it's robustly helpful to distinguish between people. Here is the example of a sample of input data and corresponding output Fisher face images.



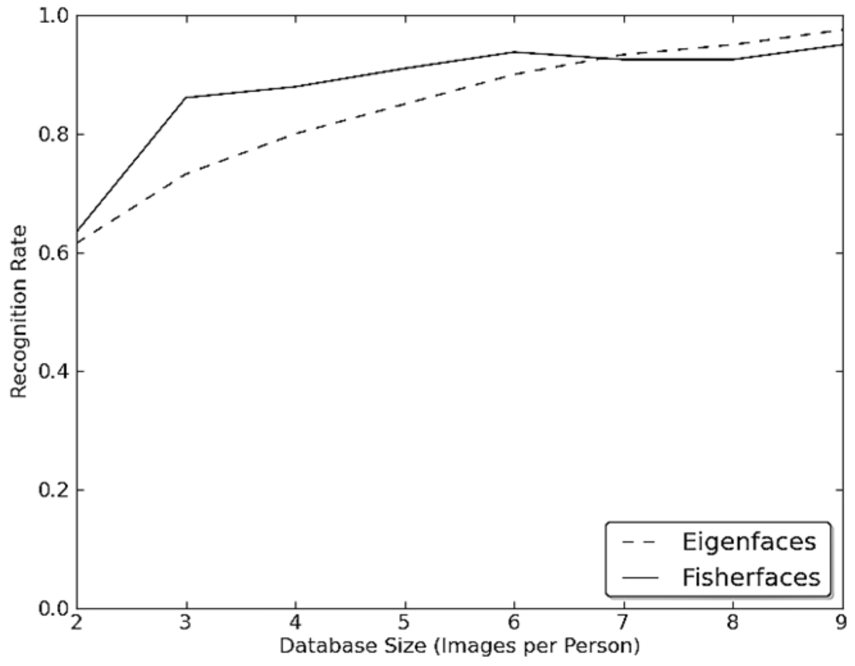
sample of input data and corresponding output Fisherface images

Fisherface could classify the training set to deal with various people and various facial expression. We could have higher accuracy in facial expression than Eigenfaces method. Moreover, Fisherface removes the first three principal components which are responsible for light intensity changes: it is more steady to light intensity. Fisherface is more complicated than Eigenface in finding the projection of face space. Calculation of ratio of between-class scatters to within-class scatter needs lots of processing time. Finally, due to the demand for better classification, the dimension of projection in face space is not as compact as Eigenface. As a result, it consumes larger storage of the face and more processing time in recognition.

4.2.3. Local Binary Pattern Histogram

Eigenfaces and Fisherfaces take a kind of holistic approach to face recognition. We handle our data as a vector somewhere in a high-dimensional image space. We know that high-dimensionality has problems, so a lower-dimensional subspace is identified, where probably, useful information is saved. The Eigenfaces approach maximizes the total scatter, which can lead to problems if an external source creates the variance because components with a maximum variance over all classes are not necessarily useful for classification. So, to preserve some discriminative information we applied a Linear Discriminant Analysis and optimized as described in the Fisherfaces method. The Fisherfaces method worked relatively better at least for some scenarios.

The conditions in our real life is not perfect. We cannot guarantee perfect light settings in our images or lots of different photos of a person. So, what can we do if there is only one image for each person? Our covariance estimates for the subspace may be wrong, so will the result of recognition also be false? The experiment [23] shows that the Eigenfaces method had a 96% recognition rate on the AT&T Facedatabase [24] in which for every person there were 10 different images from different angles. But the question is how many photos do we need to get such effective estimates? Below we can see recognition rates of the Eigenfaces and Fisherfaces methods on the AT&T Facedatabase.

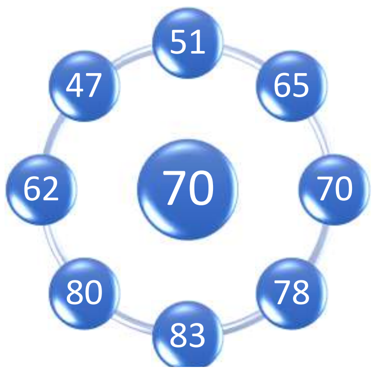


Therefore, to have high recognition rates, we need a minimum of 9 images of each person, and the Fisherfaces method cannot be helpful in this situation.

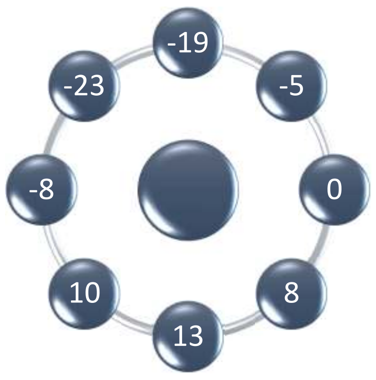
Local Binary Pattern [25] is a simple meanwhile so efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and observes the result as a binary number. Due to its discriminative power and computational simplicity, LBP texture operator has become a favorite approach in various applications. It can be seen as a combining approach to the traditionally divergent statistical models of texture analysis. Perhaps the essential feature of the Local Binary Pattern operator in real-world applications is its robustness to monotonic gray-scale changes caused, for example, by illumination variations. Another valuable feature is its computational simplicity, which makes it applicable to analyze images in challenging real-time conditions.

The primary idea for developing the LBP operator was that two-dimensional surface textures could be described by two complementary measures: local spatial patterns and grayscale contrast. The initial LBP operator (Ojala et al. 1996) creates labels for the pixels of the image by thresholding the 3×3 neighborhood of every pixel with the central value and considering the result as a binary number. The histogram of these $2^8 = 256$ various labels can then be used for describing a texture. This operator used jointly with a simple local contrast measure delivered excellent performance in unsupervised texture partition [26]. After this, many related approaches have been evolved for color and texture partitioning.

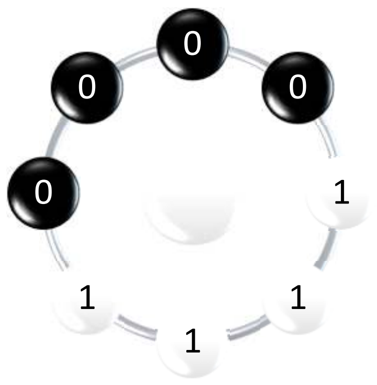
For the face recognition, the algorithm also requires grayscale images for processing the training set. On the contrary to the previous two algorithms, this one is not a holistic approach and treats each picture of data set as a unit. As we already discussed, LBPH aims to work by blocks of 3 by 3 pixels. The neighbor's value will be subtracted from the central value, and as a result, each neighbor who has a negative value will be set the 0 otherwise 1 (Figure below). When all the comparisons have been completed, each result will be multiplied by a weight. Each pixel weights the power of two from 2^0 to 2^7. Each pixel in the center of a *3 x 3* square has eight neighbors. These eight pixels represent one byte which explains the reason for using these weights. The weights are affected in a circular order. It doesn't matter which weight is affected to which pixel, however, the weight of a pixel does not change. For example, if the pixel top left weights 128, it will keep this weight for all the comparisons in the picture. Then, the sum of the weights is calculated and becomes the value of the pixel in the middle of the square. The figure below shows the results of the comparisons and the weight which is related to each pixel.



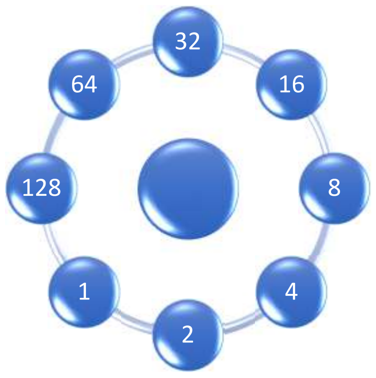
1.Sample



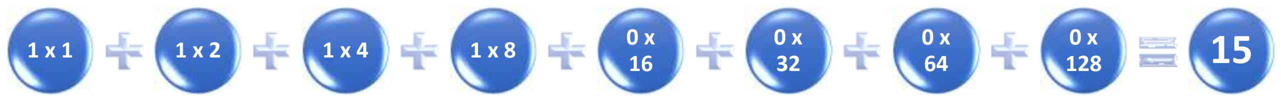
2. Difference



3.1 Threshold



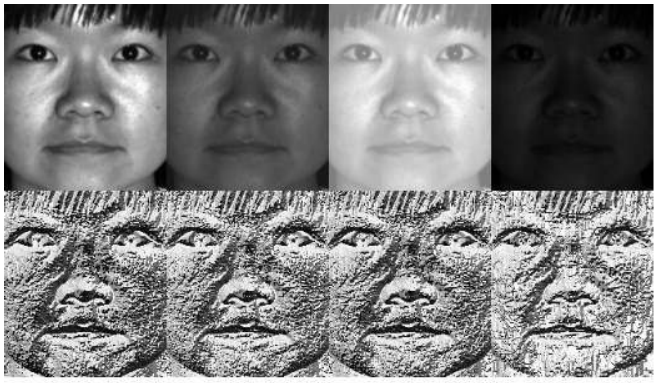
3.2. Weights



5. Multiply by power of two and sum

When this process is completed for each part of the image, the image is divided into a certain number of regions. Then, a histogram is pulled out from each area, and all the histograms are concatenated. For recognizing a face accurately, the same process is performed, and the final histogram is compared to each final histogram in the training data. The label related to the closest histogram is the prediction of the algorithm. As for the Histogram of Oriented Gradients detector, this algorithm is not sensitive to a variation of luminosity.

By definition, the LBP operator is robust against monotonic gray scale transformations. We can merely show this by looking at the LBP image of an artificially modified image.


what an LBP image looks like

5. Tests

We should consider different factors when comparing the algorithms. Several tests will be done with various modifications in the training set such as the number of persons, the number of pictures per person and a change in the background lighting. The training set consists of faces with various emotional gestures. The algorithms which will be used for the test are Eigenfaces, Fisherfaces, and local binary patterns histograms.

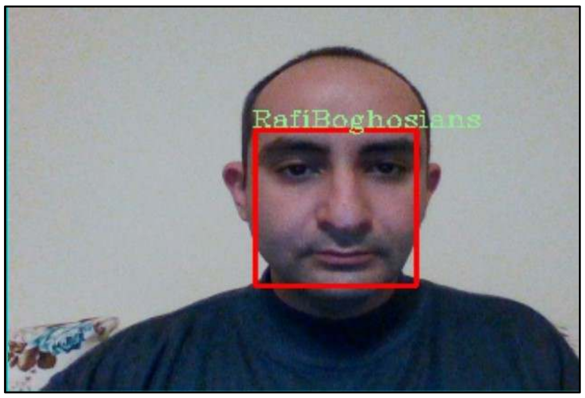
To test how accurate each algorithm works in the same environment with the same lighting conditions and how each algorithm reacts to the pictures from another place we will split the test into two stages.

1. In the first phase, the training data of a person will be in the same environment as the test data. In other words, the pictures are taken in the same room and with the same lighting conditions.
2. In the second stage, the images from the training data and the test data will not be from the same environment. The photos can be taken in another place and where the luminosity is different.

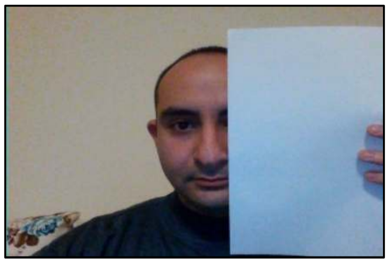
We will label all the picture. So, we will be able to compute the percentage of accuracy or inaccuracy of each algorithm. If the prediction result is equivalent to the label of the training set, the algorithm predicted the person accurately. All the photos in training set are captured with a Dell Inspiron 15 3542 laptop webcam. OpenCV haar-cascade does the face detection part.

Although it is not within the scope of this paper to test face detection part which is done by the haar-cascade approach, however, we do some tests.

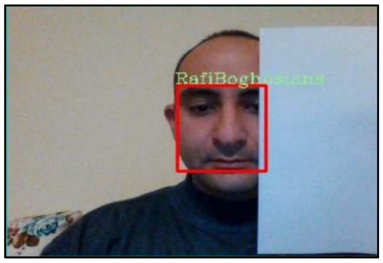
Here is the picture which the program can adequately detect and track the image.



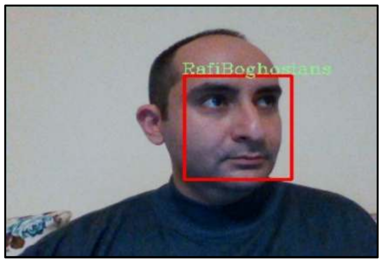
If we try to cover the part of the face like eyes, mouth, the program will not be able to detect the face. Below, we can see the negative side of this approach.



But it can successfully work when we cover the half of the one eye.

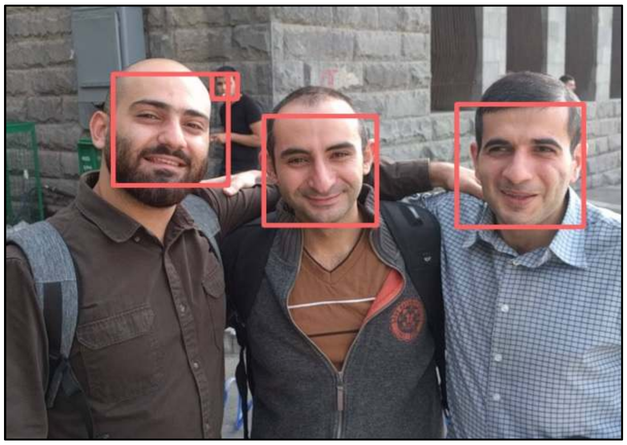


In the dynamic environment, the algorithm will not work when the face deviates is less than 45 degrees.

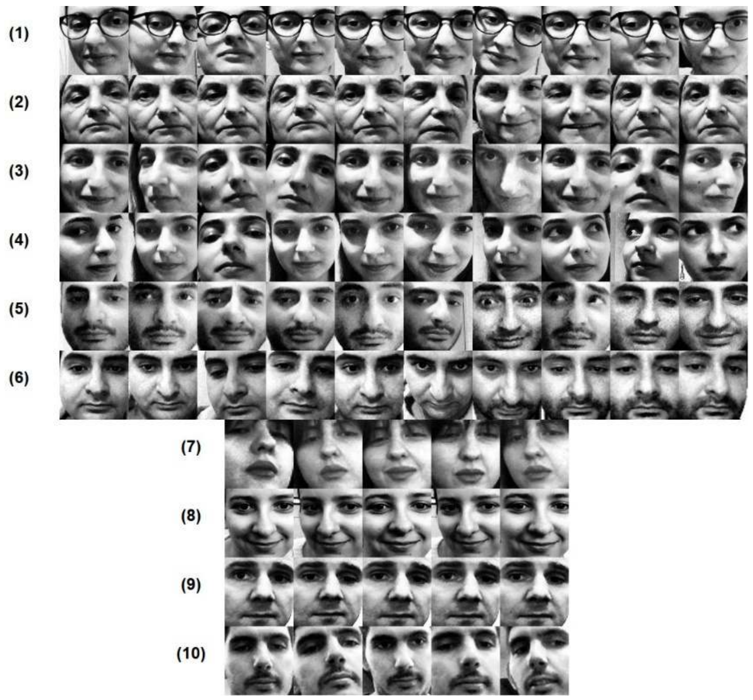


The algorithm does not work when there is strong lightning source behind the face.

However, the Viola-Jones algorithms have high performance when we test it in a static environment. The left picture was just captured in Spring of 2016 in front of AUA Paramaz Avedisian Building and on the right side we had Solvay Conference in 1927. Both images tested with Object Detection using Haar feature-based cascade classifiers of OpenCV library.

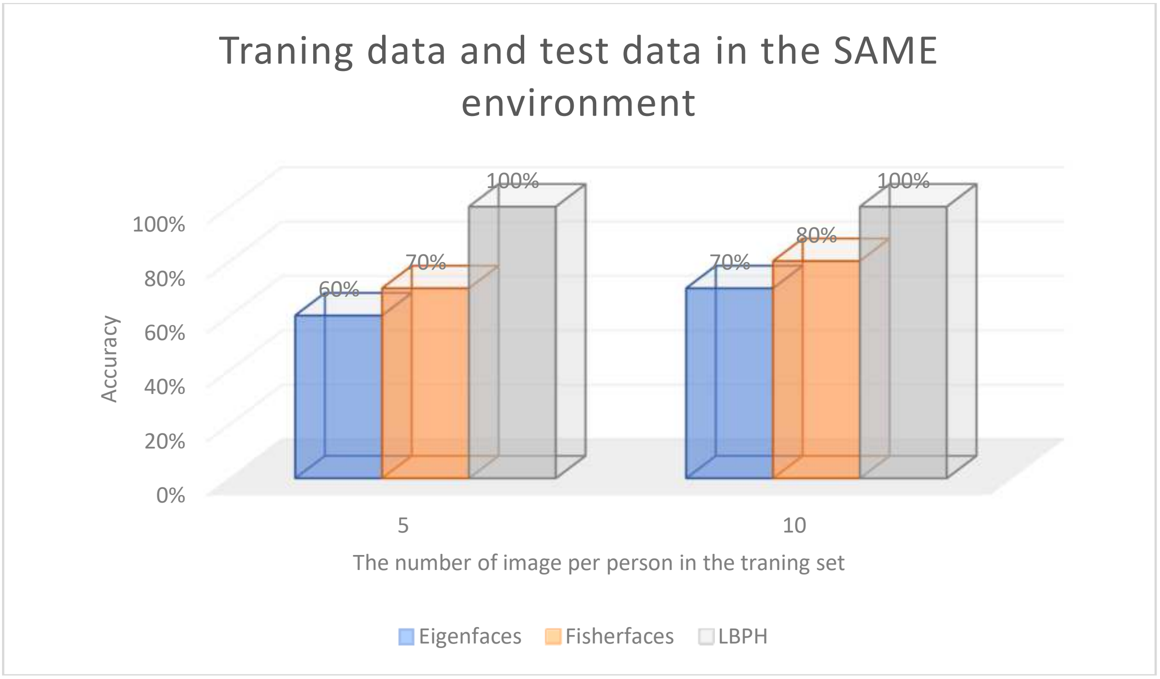


When we talk about dynamic face recognition, everything becomes more complicated. In the static environment, we can have more precise results, and we can make a better conclusion based on our findings. But in dynamic recognition, a little bit of changing in the environment, facial feature or distance from the camera may bring false results. Below, we represent the part of the training set which we use for our experiments.

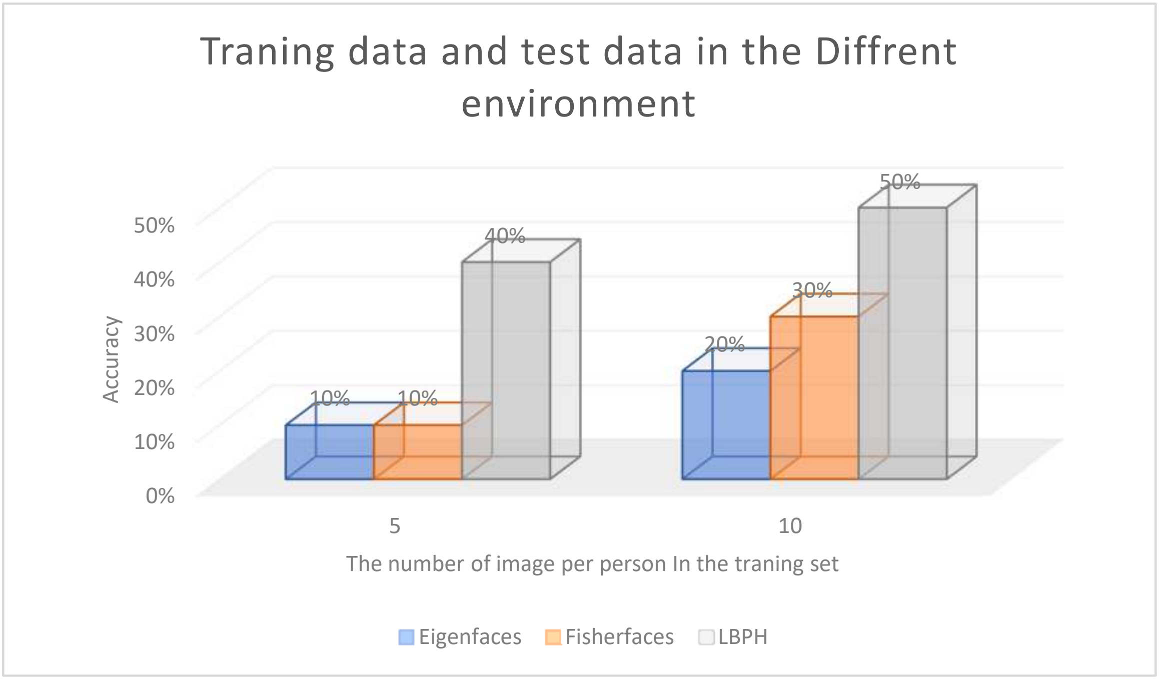


Face recognition testing results

The first figures show the result obtained by testing all the three algorithms in the same environment. The graphs represent the percentage of accuracy for each test. This percentage is computed by comparing the number of images that have been correctly recognized and the total number of images in the training set. The first observation that can be done in first figure is the increase of the accuracy when the number of images per person is growing. The first two algorithms which have statistical approach have less accuracy than LBPH. However, they still have an accuracy higher than 60%.



The second figure shows the result obtained by testing all the three algorithms in different environments. The graphs represent the percentage of accuracy for each test. This percentage is computed by comparing the number of images that have been correctly recognized and the total number of images in the training set. We can see that the performance of all the approaches dramatically decreases in relation to the first figure. The Eigenfaces and Fisherfaces recognized only the third row. LBPH recognized the third, fifth, sixth, seventh and tenth rows.


Traning data and test data in the Diffrent environment
Accuracy
0%
10%
20%
30%
40%
50%
10%
10%
40%
20%
30%
50%
5
10
The number of image per person In the traning set
Eigenfaces
Fisherfaces
LBPH

6. Conclusion

Eigenfaces, in both stages of the test, did not demonstrate remarkably good results in case of increasing the number of subjects. Nevertheless, growth in the size of the training set contributes to recognizing more faces. Eigenfaces consumed less memory and more rapid computation speed rather than Fisherfaces. The remarkable disadvantages of Eigenfaces were sensitivity to light and facial expressions.

Fisherfaces had better outcomes in the second stage with doubled amount of dataset. However, its behavior was relatively the same as Eigenfaces in the first stage of the test. This algorithm was relatively insensitive to light compared to Eigenfaces but consumed more memory.

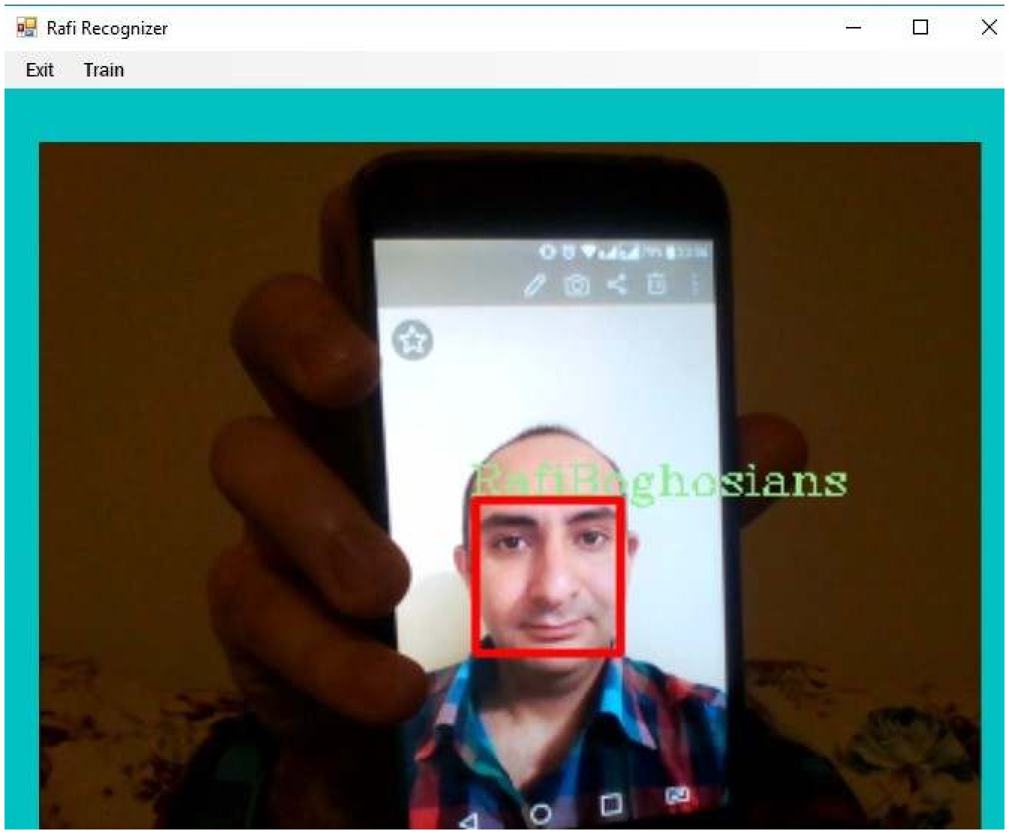
LBPH always had 100% in the first phase. In the second phase, this algorithm had a dramatic drop in its accuracy compared to the first stage. An increase in the number of subjects did not dramatically change its predictions. In each phase, an increase in the training data had a positive effect.

Taking into account the tests results, we can conclude that methodology of searching for patterns is more efficient for performing a facial recognition than a statistical approach. From the experiments conducted during the tests, it was evident that the Local Binary Pattern excelled Eigenface and Fisherface algorithms, regardless of multiple conditions. The results suggest that LBPH is the most excellent model for face recognition using EmguCV library. This model achieves 50% accuracy in the dynamically changing environment. We can improve the results by using a more complex classifier, and this may be the right way for EmguCV to boost the efficiency in future releases of the library.

A recognition model with an accuracy of 50% on a dataset of 100 could be helpful in applications if the interface provides multiple identity suggestions (at least 10). It will be very slow and challenge for a human to learn new identities of 100 individuals from photos to the point where they can achieve high recognition accuracy. However, the results demonstrate that there is still a lot of room for improvement in face recognition techniques.

Further Extensions

Unfortunately, we cannot use EmguCV library to develop facial biometric authentication systems. The primary obstacle is that none of the algorithms can differentiate the real face of a person from the picture.



But we can use them in CRM application to give a better user experience to customers or in attendance checking systems with not a significant amount of dataset and relatively stable lighting conditions.

References

1. https://newsroom.mastercard.com/videos/mastercard-identity-check-facial-recognition-biometrics/
2. https://www.nfcworld.com/2016/11/28/348741/jaguar-files-facial-gait-recognition-system-patent-vehicle-entry/
3. https://aicure.com/
4. https://techcrunch.com/2012/08/10/facedeals-check-in-on-facebook-with-facial-recognition-creepy-or-awesome/
5. Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.
6. Taigman, Yaniv, et al. "Deepface: Closing the gap to human-level performance in face verification." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014.
7. http://www.emgu.com/wiki/index.php/Main_Page
8. Pulli, Kari, et al. "Realtime computer vision with OpenCV." *Queue* 10.4 (2012): 40.
9. Laganière, Robert. *OpenCV Computer Vision Application Programming Cookbook Second Edition*. Packt Publishing Ltd, 2014.
10. Irani, Michal, Benny Rousso, and Shmuel Peleg. Recovery of ego-motion using image stabilization. Hebrew University of Jerusalem. Leibniz Center for Research in Computer Science. Department of Computer Science, 1993.
11. Rehm, Matthias, Nikolaus Bee, and Elisabeth André. "Wave like an Egyptian: accelerometer based gesture recognition for culture specific interactions." Proceedings of the 22nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction-Volume 1. British Computer Society, 2008.

12. Wilson, Phillip Ian, and John Fernandez. "Facial feature detection using Haar classifiers." *Journal of Computing Sciences in Colleges* 21.4 (2006): 127-133.
13. Viola, Paul, and Michael Jones. "Rapid object detection using a boosted cascade of simple features." *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE, 2001.

14. Turati, Chiara, et al. "Newborns' face recognition: Role of inner and outer facial features." *Child development* 77.2 (2006): 297-311.
15. Kanade, Takeo. "Picture processing system by computer complex and recognition of human faces." (1974).
16. Brunelli, Roberto, and Tomaso Poggio. "Face recognition through geometrical features." *European Conference on Computer Vision*. Springer, Berlin, Heidelberg, 1992.
17. Wiskott, Laurenz, et al. "Face recognition by elastic bunch graph matching." *IEEE Transactions on pattern analysis and machine intelligence* 19.7 (1997): 775-779.
18. Messer, Kieron, et al. "Performance characterisation of face recognition algorithms and their sensitivity to severe illumination changes." *International Conference on Biometrics*. Springer, Berlin, Heidelberg, 2006.
19. Ahonen, Timo, Abdenour Hadid, and Matti Pietikäinen. "Face recognition with local binary patterns." *European conference on computer vision*. Springer, Berlin, Heidelberg, 2004.
20. Sirovich, Lawrence, and Michael Kirby. "Low-dimensional procedure for the characterization of human faces." *Josa a* 4.3 (1987): 519-524.
21. Turk, Matthew, and Alex Pentland. "Eigenfaces for recognition." *Journal of cognitive neuroscience* 3.1 (1991): 71-86.
22. Belhumeur, Peter N., João P. Hespanha, and David J. Kriegman. "Eigenfaces vs. fisherfaces: Recognition using class specific linear projection." *IEEE Transactions on pattern analysis and machine intelligence* 19.7 (1997): 711-720.
23. Martínez, Aleix M., and Avinash C. Kak. "Pca versus lda." *IEEE transactions on pattern analysis and machine intelligence* 23.2 (2001): 228-233.
24. http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
25. Ojala, Timo, Matti Pietikainen, and Topi Maenpaa. "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns." *IEEE Transactions on pattern analysis and machine intelligence* 24.7 (2002): 971-987.
26. Ojala, Timo, and Matti Pietikäinen. "Unsupervised texture segmentation using feature distributions." *Pattern recognition* 32.3 (1999): 477-486.