Haaga-Helia
University of Applied Sciences

A comparison of facial recognition's algorithms

Nicolas Delbiaggio

Bachelor's Thesis
Degree Programme in BIT
2017

Abstract

26.05.2017

Author(s) Nicolas Delbiaggio	
Degree Programme in Business Information Technology	
Report/thesis title A comparison of facial recognition's algorithms.	**Number of pages and appendix pages** 41

The popularity of the cameras in smart gadgets and other consumer electronics drive the industries to utilize these devices more efficiently. Facial recognition research is one of the hot topics both for practitioners and academicians nowadays. This study is an attempt to reveal the efficiency of existing facial recognition algorithms through a case evaluation.

The purpose of this thesis is to investigate several facial recognition algorithms and make comparisons in respect of their accuracy. The compared facial recognitions algorithms have been widely utilized by industries. The focus is on the algorithms, Eigenfaces, Fisherfaces, Local Binary Pattern Histogram, and the commercial deep convolutional neural network algorithm OpenFace. The thesis covers the whole process of face recognition, including preprocessing of images and face detection. The concept of each algorithm is explained, and the description is given accordingly. Additionally, to assess the efficiency of each algorithm a test case with static images was conducted and evaluated.

The evaluations of the test cases indicate that among the compared facial recognition algorithms the OpenFace algorithm has the highest accuracy to identify faces.

Through the findings of this study, we aim to conduct further researches on emotional analysis through facial recognition.

Keywords
Computer vision, Face detection, Facial recognition algorithms, Neural Networks.

Table of contents

Abbreviations

CNN	Convolutional Neural Network
DNN	Deep Neural Network
HOG	Histogram of Oriented Gradients
LBPH	Local Binary Patterns Histograms
MSE	Mean squared error
PCA	Principal Components Analysis
ReLU	Rectified Linear Units Layer

1 Introduction

Nowadays we can observe more and more cameras in our society. They are present in streets, in supermarkets and many other public locations. The aim of their presence is most of the time related to security. However, these are not the only places where we can find cameras. The number of devices per person is constantly increasing, and they are often equipped with a camera. For example, all smartphones and tablets possess one. It becomes rare to see a laptop without a camera included, and when it is not the case, the owners often have a webcam instead. Even the latest versions of TV start to have an integrated camera. Many people also use cameras such as GoPro's for recording themselves doing extreme sports. The new tendency is purchasing a drone for filming from the air and taking pictures in some areas where humans cannot have access.

The image quality of all these cameras has considerably been improved during the last years and will continue to be enhanced in the future. Cameras are almost everywhere, but they are usually only used for taking pictures, videos or for video conversations. Their aim is only for recording a moment of the life. However, it is possible to use them for various other goals.

One possibility is computer vision which is used to understand digital images and videos. For example, the well-known Snapchat application (Weinberger, 2017) is based on computer vision. Via face detection, the user can add a filter which will be adjusted to the face. The most daily life applicable application is the search engine of Google. The engine is able to show pictures related to a research even if the pictures have not been tagged because the engine is able to recognize what is shown on the picture. Another example is the facial recognition performed by Facebook when a new picture is uploaded by a user. Facebook is able to detect the faces on the picture and recognize the persons. Then, the persons will automatically be suggested to the user for tagging him/her.

All the mentioned examples use computer vision, but additionally, it also requires machine learning. Some data are needed for teaching a machine to recognize an object, the larger the quantity of data is, the better the results are. Facebook and Google have accurate results and can provide you with information which seems to be unimaginable. The main reason of their success is due to their amount of data. They collect an enormous amount of data, and they can use them for training their artificial intelligence.

These companies have definitely the advantage of possessing a vast amount of data, but they also utilize really powerful algorithms. However, they do not have the only algorithms in the world for performing facial recognition, and they do not keep everything secret either. For example, Google made public the method of their facial recognition named FaceNet (Schroff, Kalenichenko, & Philbin) and Facebook did the same with their system DeepFace (Taigman, Yang, & Ranzato, 2014). Several algorithms are public. While some of them use a statistical approach or search for patterns, some other are using a neural network. What are the differences between these categories? Is it possible to recognize a face accurately through these algorithms? In the first part, this study provides an overview of the logic behind different facial recognition algorithms. In the second part, the results of the algorithms are elaborated through a case study.

2 Research Question

The goal of this chapter is to fix the objectives, define the scope and provide the plan of this thesis.

2.1 Objective of the project

The aim of this project is to test different algorithms for facial recognition. Each algorithm will be trained with the identic data set. Then, another data set will be used to test the accuracy of the algorithms. Finally, the test results will be compared, and the most accurate facial recognition algorithm will be pointed out. This thesis pursuit to answer the following question:

How efficient are the common facial recognition's algorithms to identify static images?

There are various commercial and non-commercial algorithms on the market, testing all the algorithms due to the time constrains is out of the question. Therefore, the evaluation is only delimited to those algorithms that are most used by the commercial sector, popularity, and were freely available (open source).

2.2 Scope of the project

The different steps for performing a recognition will be described briefly in order to get a sufficient understanding of the functioning and the necessity of each step. The environment where pictures are taken affects the quality of the image and can play a significant role. The environment which has been used for taking the pictures of the training data will be described.

It is recommended to pre-process a training dataset in order to improve the quality of the recognition. Different possibilities for correcting pictures will be demonstrated, and the effects of these corrections will be explained. Some modifications of the training set require the coordinates of the face in the picture to be performed. Thus, it will be shown how a face can be detected. Several ways to do so exist and they all have their advantages and disadvantages. Two different methods will be explained in order to get a better comprehension of the logic behind face detection, which is not the same as face recognition.

The biggest and most complex step is teaching the machine to recognize faces. Many pictures are needed in the training data, and the machine will have to learn how to differentiate faces. Different algorithms can be trained for that, some of them use a statistical approach or search for patterns, and some others use a neural network. We will go through these different concepts and their logics. As a result, the goal is to understand the process of the regarding algorithms. The different algorithms will then be compared, and their strengths and weaknesses will be discussed. Finally, the algorithms will be tested in a case study and the results will be compared. We will see which algorithms got the best results and try to identify the key points, which make them better.

3 Background study

Recognizing a face requires several pictures per subject. Each picture needs to be labeled with the name of the subject. First, the faces have to be detected in the picture. Then, they are pre-processed and used as input for training the machine learning. Finally, a picture can be used in the machine learning to predict the person. The process to recognize a face is represented in figure 1.

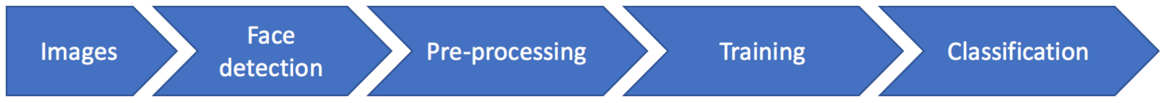


Figure 1. Facial recognition process

Facial recognition

Facial recognition is a technique used by computer algorithms to identify or verify a person or an object through images. The objective of facial recognition techniques is to get different features of human faces from images or different people (Lone, Zakariya, & Ali, 2011). In computer science, facial recognition is a part of computer vision. Facial recognition has been around for many decades mainly utilized by the army. Due to the popularity of social networks and smart gadgets, the importance of facial recognition becomes more evident.

Grayscale

An image can be represented in grayscale. Each pixel of a picture in a grayscale is represented by a number from 0 to 255, which corresponds to an intensity of gray of the pixel. This number is stored in one byte. 0 represents the color black and 255 is white. As it will be shown, grayscale is often used in computer vision and makes treatment of images easier when color is not essential.

Neural network

A neural network is an approach that consists of training computers through programming to analyze data for specific purposes. Pattern recognition, for example, is a subset of neural network that is applied to analyze the complex data. Neural network analysis is based on human brains unlike the conventional computer programming based on specific instructions(Bishop, 2005). Various techniques have been developed over time for neural networks, e.g. statistical technique, pattern recognition, deep learning, convolutional neural network.

3.1 Face detection

Before recognizing a face, it is first essential to detect and extract the faces from the original pictures. For recognizing a face, the algorithms compare only faces. Any other element in the picture that is not part of a face deteriorates the recognition. There are several existing algorithms for detecting faces.

3.1.1 Haar-cascade classifier

Haar-cascade (Wilson & Fernandez, 2006) is a method, invented by Viola and Jones (Viola & Jones, 2001), which trains a machine learning for detecting objects in a picture. In this context, it can be used to detect faces. The name of this method is composed of two important words, Haar and Cascade. Haar belongs to Haar-like features which is a weak classifier and will be used for the face recognition. A weak classifier is a classifier which is only slightly better than a random prediction. A Haar-like feature is a rectangle which is split into two, three or four rectangles. Each rectangle is black or white. Figure 2 shows the different possible features. A Haar-cascade needs to be trained with various positive and negative pictures. The objective is to extract the combination of these features that represents a face. While a positive picture contains the object which has to be recognized, a negative picture represents a picture without the object. In the context of face detection, a positive picture possesses a face, and a negative picture does not. This machine learning requires grayscale pictures. The intensity of gray will be used to detect which feature is represented. These features can be found by calculating the sum of the dark pixels in an area subtracted by the sum of the bright pixels.

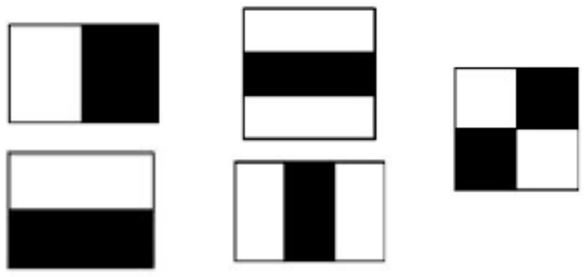


Figure 2. The 5 Haar-like features used for detecting faces

The extracted combination of features from the training part will be used for detecting faces in a picture. To detect a face in an unknown picture the combination of the features will be researched. The features are tried to be matched only in a block of pixels defined by a scale. The scale can be, for example, a square of 24x24 pixels. Each feature of the combination will be tried to be matched one by one in the block. If one of the features does not appear in the block, the research in it will be stopped. The remaining features will not be tested because the machine concludes that there is no face in this block. Then, a new block is taken, and the process will be repeated. This method tests all the blocks of pixels with the researched combination in cascade which explains the second word in the name of the method. This method is efficient to detect an image without faces because only a few tests need to be run to infer that the image does not contain a face. A face is consequently detected when each feature of the combination has been recognized correctly in a block. Figure 3 is a rough representation of the features combination which will be tried to be matched in each block. We can see that the eyes are darker than the cheeks and the middle of the nose is brighter. All these features which were extracted from the training are used to find a pattern to represent a face.

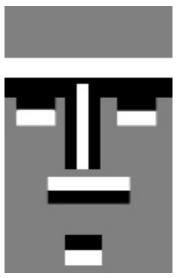


Figure 3 Example of a Haar-like features combination (Arubas)

The process will proceed block by block until the last one. After checking the last block, the scale is increased, and the detection process starts again. The process is repeated several times with different scales to detect faces of different size. Only few pixels are different between two neighbor blocks. Therefore, each time a face is detected in a picture, the same face is detected in different blocks. All the detected faces that concern the same person are merged and are considered as one at the end of the entire process. The accumulation of these weak classifiers builds a face detector able to detect faces very fast with a suitable accuracy. A Haar-cascade classifier has to be trained only once. Thus, it is possible to create one's own Haar-cascade or use one which has already been trained.

3.1.2 Histogram of Oriented Gradients (HOG)

HOG (Dalal & Triggs, 2005) (Geitgey, 2016) is another method for detecting objects which can also be used for detecting faces. In this case, we will use HOG for detecting any front face in a picture. This method requires an image in grayscale. Each pixel of the image is represented by an integer due to the grayscale. The HOG method compares each pixel with its neighbors. Most of the time, a pixel is surrounded by eight other pixels. The aim is to find the direction where the picture is getting darker. A white arrow will be drawn to represent this direction. So, the smaller is the number of the pixel then the darker is the pixel. This treatment is done for each pixel of the picture. The strength of this method is that it is not sensitive to a change in luminosity. If a picture is darker, all the pixels will be darker. The arrow representing the direction where it is getting darker will not change in a brighter picture. Indeed, this is not true if only a part of the picture is affected by a change in luminosity, which could be caused, for example, by a lamp.

This step gives us the shape of the analyzed face in detail, but too many details are caught. The aim is to recognize any face, but too many details can only detect one specific face. Thus, only the relevant directions need to be kept. The next step is to split the pictures into squares of 16x16 pixels. We count how many times each direction has been discovered beforehand, and only one arrow is drawn in the square with the direction which was the most frequently found. This action is done for each block of 16x16 in the picture. This treatment gives a better representation of a general face. All the steps will be performed with a vast number of frontal faces for determining the best pattern for face detection. Figure 4 represents an average face which has been trained with 3000 images by the library dlib (King, 2009)(King Davis, 2014).

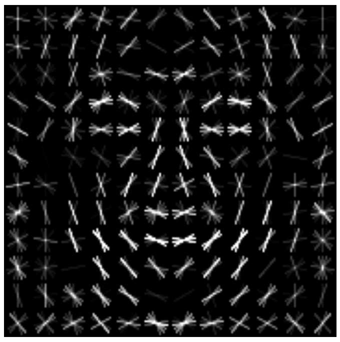


Figure 4 Visualization of the HOG detector from the library

When the detector has been trained, it is ready to use. For detecting a face, the detector will go through the picture, search for a face and check if the pattern matches somewhere. Each time the pattern is recognized somewhere in the picture, it means a face has been detected. It is also possible to find several faces in a picture with this method.

3.2 **Pre-processing of pictures**

Any of the previous methods can be used for extracting faces from input pictures. The next step is to pre-process these faces in order to make the training phase easier and improve the probability to recognize a person correctly. The training data will be standardized. Not all the pictures have the same zoom on the face and have maybe not all the same size. Most of the algorithms for facial recognition require the same size for the entire training set. Pre-processing includes different modifications. First of all, the faces need to be centered in the picture in the same way. The location of the two eyes and the nose is often used as a landmark for centering faces. The aim is to have the eyes at the same level and the nose at the same position for all images. To apply these modifications, the coordinates of the landmarks are needed. For that, it is possible to use a Haar-cascade classifier for detecting nose and eyes. It is also possible to use the facial landmarks detector, which is available inside the dlib library which is based on the work of Kazemi & Sulivan (Kazemi & Sullivan). The landmark detector has been trained to recognize 68 specifics points on a face which are shown in figure 5. This method has been trained with many pictures, which have been manually labeled before for each landmark.

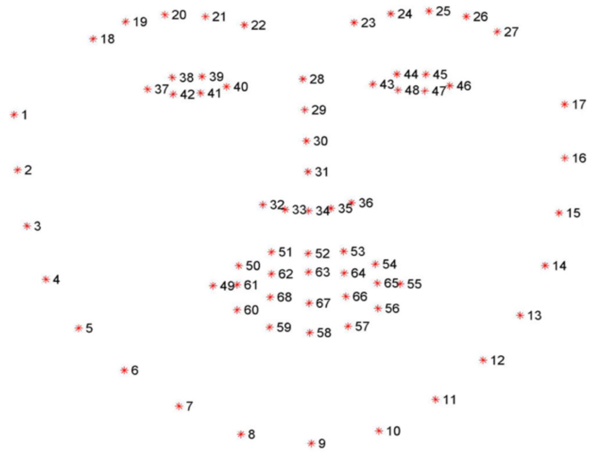


Figure 5. Visualization of the 68 landmarks(Pyimagesearch, n.d.)

This detector could be used to center the pictures based, for example, on the landmark 37, 46 which are present in the eyes and the number 34 which is the bottom of the nose. The faces can be rotated until the landmarks 37 and 46 are on the same level. This modification allows comparing pictures which do not have the face straight. Now, when all the pictures are in the same conditions, they can be resized to a common size and can also be cropped to the face edges. A small size such as 96x96 pixels is recommended because the pictures are lighter and thus it is faster to perform the machine learning. A small picture also contains less information. The aim is to find the appropriate size which is as small as possible in order to improve the computing time and keep enough information for recognizing a person. Several tests need to be performed to find the suitable size (Barnouti, 2016).

3.3 Facial recognition’s algorithms

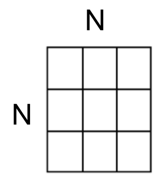
There are several approaches for recognizing a face. The algorithm can use statistics, try to find a pattern which represents a specific person or use a convolutional neural network. These different approaches can be observed through the explanations of different algorithms in this chapter.

3.3.1 Eigenfaces

Eigenfaces (Turk & Pentland, 1991) (Jaiswal, Gwalior, & Gwalior, 2011) (Morizet, Ea, Rossant, Amiel, & Amara, 2006) is a method for performing facial recognition based on a statistical approach. The aim of this method is to extract the principal components which affect the most the variation of the images. This is a holistic approach, the treatment for predicting a face is based on the entire training set. There is no specific treatment between images from two different classes. A class represents a person. Pre-processed pictures with grayscale are required for training the machine learning. Each pixel of a picture represents one dimension, it means a 96x96 pixels image is represented into 96 x 96 = 9216 dimensions. Eigenfaces is based on Principal Component Analysis (PCA) (Turk & Pentland, 1991) (Jaiswal & al., 2011) (Morizet & al., 2006) for reducing the number of dimensions while preserving the most important information. The training part of Eigenfaces is to calculate the eigenvectors and the related eigenvalues of the covariance matrix of the training set.

Step 1- transform images into matrix

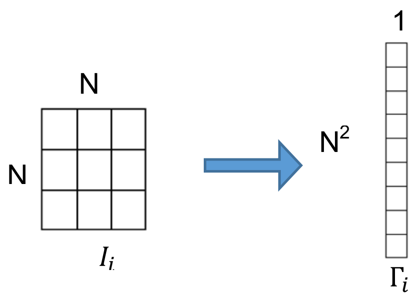
Each pixel of an image represents a number. Thus, we can easily represent them into a matrix NxN where each item of the matrix is a pixel. Each training image becomes a matrix I ($I_1, I_2, .., I_{m,}$ where m equals the number of images).



Matrix I_i NxN

Step 2 – adapt the matrix I_i into a vector Γ_i

A matrix is a high-dimensional space compared to a vector which is a lower-dimensional space. Therefore, each row of the matrix I_i will be concatenated and then transposed for representing the vector Γ_i.



Step 3 – calculate the average of the vectors Γ_i

$$\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$$

The sum of each vector Γ_i is calculated, and then the sum is divided by the number of images M which gives the vector Ψ representing the average.

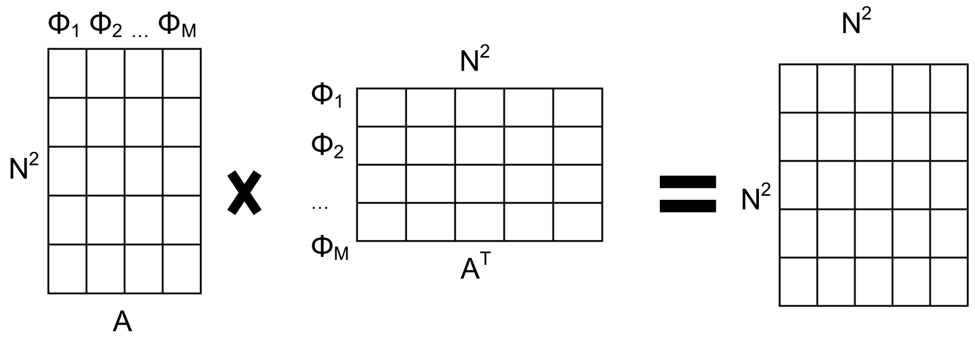
Step 4 – subtract the average from the vector Γ_i

$$\Phi_i = \Gamma_i - \Psi$$

Each image represented by the vector Γ_i will subtract the average of all the pictures. The result of the subtractions is represented by the vector Φ_i.

Step 5 – compute the covariance matrix C

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = AA^T$$



Then, the covariance is calculated based on the Φ_n. The vectors Φ_n are grouped to represent the matrix A. The covariance matrix C is computed by multiplying the matrix A with the transposition of the matrix A called A^T.

Computing the eigenvectors and the eigenvalues of a matrix

The relation between a matrix A, its eigenvectors and its eigenvalues is represented with the formula below:

$$A\vec{v} = \lambda\vec{v} \; or \; (A - \lambda I_n)\vec{v} = 0$$

$\vec{v} = eigenvector$

$\lambda = eigenvalue$

$I_n = identity\ matrix\ of\ A$

First, the eigenvalues need to be calculated. Then, the eigenvalues will be used for computing the eigenvectors. The following formula is used for obtaining the eigenvalues.

$\det(A - \lambda I_2) = 0$

The matrix A below is used as an example to demonstrate the process to calculate the eigenvalues.

$$A = \begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix}$$

The scalar λ is multiplied by the identity matrix of the matrix A. Thus, two eigenvalues can be calculated. The number of eigenvalues depends on the number of rows of the matrix A.

$$\begin{vmatrix} 2-\lambda & 3 \\ 2 & 1-\lambda \end{vmatrix} = 0$$

$$(2-\lambda) * (1-\lambda) - 2 * 3 = 0$$

The two calculations above represent the calculation of $\det(A - \lambda I_2) = 0$ When the equation above is solved, the eigenvalues -1 and 4 are obtained. Then the eigenvectors related to each eigenvalue can be calculated by solving the first equation $A\vec{v} = \lambda\vec{v}$ with the eigenvalues previously obtained. The following example shows the calculation with the eigenvalue -1.

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} = -1 \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix}$$

$$\begin{cases} 2x_{11} + 3x_{12} = -x_{11} \\ 2x_{11} + 1x_{12} = -x_{12} \end{cases}$$

The eigenvector below is related to the eigenvalue -1, the same process is executed for each eigenvalue.

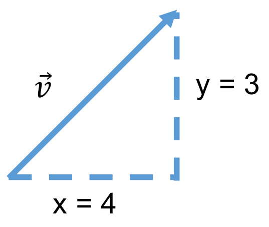
$$\vec{v} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

<u>Step 6 – calculate the eigenvectors with their related eigenvalues</u>

There are two options for computing the eigenvectors. Either we calculate the eigenvectors u_i of AA^T or the eigenvectors V_i of A^TA. These two ways have the same eigenvalues, and the relation between the eigenvectors is $u_i = Av_i$. AA^T gives a matrix N^2xN^2 as a result in contrast to A^TA which gives a matrix MxM. Most of the time we would prefer the second one because usually, the number of pictures in the training set is smaller than the number of pixels. Therefore, it is faster to accomplish the calculation. A^TA can have a maximum of

M eigenvalues and eigenvectors contrary to AA^T which can reach a number of N^2 eigenvalues and eigenvectors. The M eigenvalues represent the biggest eigenvalues included in N^2. The eigenvectors u_i need to be normalized, and the normalization has to equal 1, $\| u_i \| = 1$.

The norm of a vector represents the length of the vector. The Pythagorean theorem is used for calculating the norm. This theorem says that in a right triangle, the square of the hypotenuse equals the sum of the squares of the two other sides. It means $x^2 + y^2 = v^2$. In the example below the norm of the $\|\vec{v}\| = \sqrt{4^2 + 3^2} = 5$.



$$\boxed{\|\vec{v}\| = \sqrt{x^2 + y^2}}$$

Step 7 – K eigenvectors

The M eigenvectors are sorted in descending order based on the eigenvalues. Only K eigenvectors are kept. K is smaller than M and is decided by the user of the algorithm. All the training pictures can be represented by a combination of the K eigenvectors.

$$\Phi_i = \sum_{j=1}^{K} w_j u_j, (w_j = u_j^T \Phi_i)$$

K = number of eigenvectors

u_j = eigenvectors at the index j

Φ_i = Image i - the average

Each eigenvector which is also called eigenface represents a part of each image in the training data. An image can be decomposed through each eigenface. Each projection w is a vector that has been calculated with an image and an eigenvector. There are k projections, where k represents the number of relevant eigenvectors.

$$\Omega_i = \begin{bmatrix} w_1^i \\ w_2^i \\ w_{\dots}^i \\ w_k^i \end{bmatrix}, i = 1, 2, \dots, M$$

w_1^i = is the projection of the image i with the eigenvector at the index 1

k = the number of eigenvectors which have been kept

Face recognition with an unknown picture

First, the picture is converted into a vector Γ_i and subtract the mean as it has been done previously while training the machine learning ($\Phi = \Gamma - \Psi$). Then, the image is projected into the eigenspace with $\Phi = \sum_{i=1}^{k} w_i u_i, (w_i = u_i^T \Phi)$. The eigenspace contains all the eigenvectors. Afterwards, Φ is applied through the Eigenfaces in order to get the projections.

$$\Omega = \begin{bmatrix} w_1^i \\ w_2^i \\ w_{\dots}^i \\ w_k^i \end{bmatrix}$$

The last step is to find the picture in the training set that has the smallest distance to the test image. The formula below represents the comparison between Ω of the test image and Ω_i where i is the pictures of the training set. The class of the closest picture will predict the person of the unknown picture.

$$e_r = min \parallel \Omega - \Omega_i \parallel$$

A Euclidean distance can be used to find the closest face. However, it has been proved that the Mahalanobis distance (Turk & Pentland, 1991) is most of the time more accurate. $\mathrm{e_r}$ can be compared to a threshold and if e_r is smaller, we can determine that the face belongs to a person who is not in the training set, thus the image is added to the training set as a new person. Otherwise, the image is used as a new picture for training the recognized person and is also added into the training set with the related label. This algorithm is a constant machine learning and is supposed to become more reliable each time a new test is performed because the size of the training set increases constantly.

3.3.2 Fisherfaces

Fisherfaces (Jaiswal & al., 2011) (Morizet & al., 2006) (Belhumeur, Hespanha, & Kriegman, 1997) also uses a holistic approach. This algorithm is a modification of Eigenfaces, thus also uses Principal Components Analysis. The main modification is that Fisherfaces takes into consideration classes. As it has been said previously, Eigenfaces does not make the difference between two pictures from different classes during the training part. Each picture was affected by the total average. Fisherfaces uses the method Linear Discriminant Analysis (Jaiswal & al., 2011) (Morizet & al., 2006) (Belhumeur & al., 1997) in order to make the difference between two pictures from a different class. The aim is to minimize the variation within a class compared to the variation between classes. For that not only the total average of faces is used, but the average per class will also be an essential operation. The average is calculated with the following formula where c_i represents the class i and q_i is the number of pictures in the class c.

$$\Psi_{c_i} = \frac{1}{q_i}\sum_{k=1}^{q_i} \Gamma_k$$

The average is also subtracted from each vector as in Eigenfaces, but this time the average of the corresponding class is used.

$$\Phi_i = \Gamma_i - \Psi_{c_i}$$

Then the scatter matrixes are calculated. The Intra-class scatter matrix represented by s_w can be obtained with the formula below:

$$s_w = \sum_{i=1}^{c}\sum_{\Gamma_k \in c_i} \left(\Gamma_k - \Psi_{c_i}\right)\left(\Gamma_k - \Psi_{c_i}\right)^T$$

The Inter-class scatter matrix is represented by s_b which is calculated as follows:

$$s_b = \sum_{i=1}^{c} q_i \left(\Psi_{c_i} - \Psi\right)\left(\Psi_{c_i} - \Psi\right)^T$$

The next formula is used to obtain the total scatter matrix s_T.

$$s_T = \sum_{i=1}^{M} (\Gamma_i - \Psi)(\Gamma_i - \Psi)^T$$

After that, the goal is to find a projection of W which maximizes Fisher's optimization criteria.

$$W_{opt} = \arg max_w = \frac{|w^T s_b W|}{|w^T s_w W|}$$

The eigenvectors are found as follows:

$$s_b w_i = \lambda_i s_w w_i \, , i = 1,2, \dots m$$

Then the process is the same as Principal Components Analysis, the projection of the training picture will be compared with the projection of a test image, and the class of the picture which has the smallest distance will be the prediction of the algorithm.

3.3.3 Local Binary Patterns Histograms (LBPH)

This algorithm also requires grayscale pictures for processing the training. In contrast to the previous algorithms, this one is not a holistic approach. The aim of LBPH (Ahonen, Hadid, & Pietik, 2004) (Mäenpää, Pietikäinen, & Ojala, 2000) (Wagner, 2011) is to work by blocks of 3x3 pixels. The pixel in the center is compared to its neighbors. Each neighbor which is smaller than the pixel in the middle, the value 0 will be added to the thresholded square (figure 6) which is in charge to store the results, otherwise, a 1 will be added. The thresholded square and weights square are not present in the picture, they are only a representation to understand the process. When all the comparisons have been completed, each result will be multiplied by a weight. Each pixel has a weight to the power of two from 2^0 to 2^7. Each pixel in the center of a 3x3 square has 8 neighbors. These eight pixels represent one byte which explains the reason of using these weights. The weights are affected in a circular order. It does not matter which weight is affected to which pixel, however, the weight of a pixel does not change. For example, if the pixel top left has a weight of 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. Figure 6 shows the results of the comparisons and the weight which is related to each pixel.

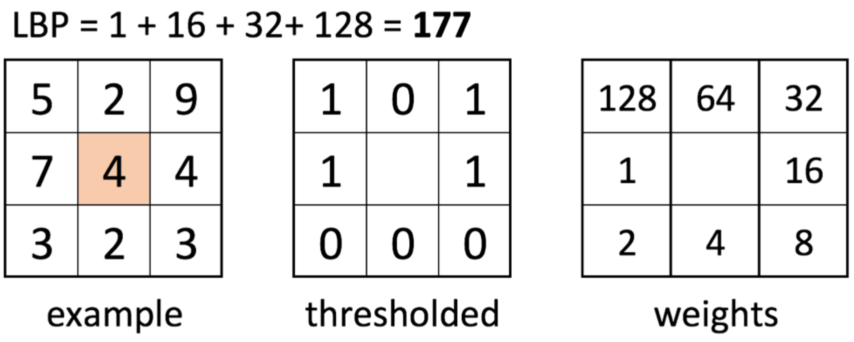


Figure 6. The example square represents a 3x3 pixels square. The thresholder square stores the result of the comparison made in the example square. The weights square is used to know the weight of each pixel in the example square. The calculation is the sum of each pixel in the thresholder according to their weight which provide the new value of the pixel in the middle of the example square.

When this process has been completed for each part of the picture, the picture is divided into a certain number of regions. Then, a histogram is extracted from each region and all the histograms are concatenated. For recognizing a face, exactly 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 hog detector, this algorithm is not sensitive to a variation of luminosity.

LBPH has been modified in different ways (what-when-how). One of them is called Extended LBPH. This extension is using a circular neighborhood which is composed of a radius and a number of sampling points. This approach allows a pixel to have more than eight neighbors. Depending on the radius (figure 7), the pixel in the middle could be compared to some pixels which are not next to it. Another extension is called uniform pattern (what-when-how). This extension takes into consideration the number of transition in the result byte. One transition is represented by a change in the byte from a 0 to a 1 or a1 to a 0. For example, 00000001 has one transition and, 00011000 has two transitions. It has been shown that patterns with a number of transitions from 0 to 2 are the most common (Ahonen & al., 2004). The patterns with two or fewer transitions usually have a specific signification how it can be seen in figure 7. All histograms with more than two transitions are regrouped together. This modification makes the vector representing the histograms smaller.

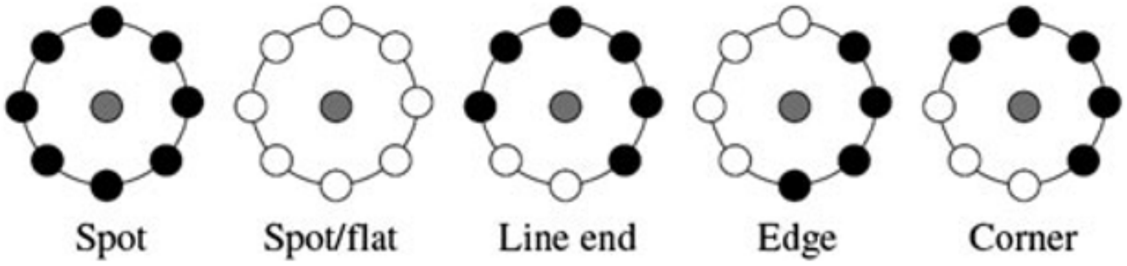


Figure 7. Visualization of the most common pattern on pictures with two or less transitions. Each pattern has a specific signification (what-when-how, n.d.).

Deep neural network

A neural network is divided into different layers. There is an input layer, one or two hidden layers and an output layer. A deep neural network (deeplearning4j) is a neural network with more than two hidden layers. It is able to use supervised and unsupervised data. A supervised data is a labeled data. For example, a picture of a person with its according label "Tom" that represents the name of the person, is a supervised data. An unsupervised data is an unlabeled data. A deep neural network is used to cluster the data according to their similarities. A deep neural network extracts features by itself. Thus thousands or millions of pictures are required for extracting relevant features.

3.3.4 Convolutional Neural Network

Using a convolutional neural network (CNN) (CS231n) (Deshpande, 2016c) (Deshpande, 2016a) (Deshpande, 2016b) is another way to perform face recognition. CNN has an architecture that enables it to use 2d pictures as inputs. A CNN consists of several layers called hidden layers. The layers are composed of several neurons. A neuron has specific weight and receives an input. After applying its logic to the input, it will provide an output. The input of the first layer is a picture of a face. The output of the last layer is the predicted class which is the person. In order to have a better comprehension of the recognition process, it is preferable to use an easy example. Therefore, "X" and "O" are used as classes instead of persons. The objective of this example is to predict whether the input is an "X" or "O".

Convolution layer

Each pixel in the picture is represented by a number between -1 and 1. -1 is a dark pixel, and 1 is a bright pixel, 0 is gray. By observing figure 8 which represents an "X" we can notice three features which are used to draw the "X". There is a pattern representing a backward slash, another one a forward slash and the last one is a cross. These patterns are used as filters and will try to be matched through the pictures.

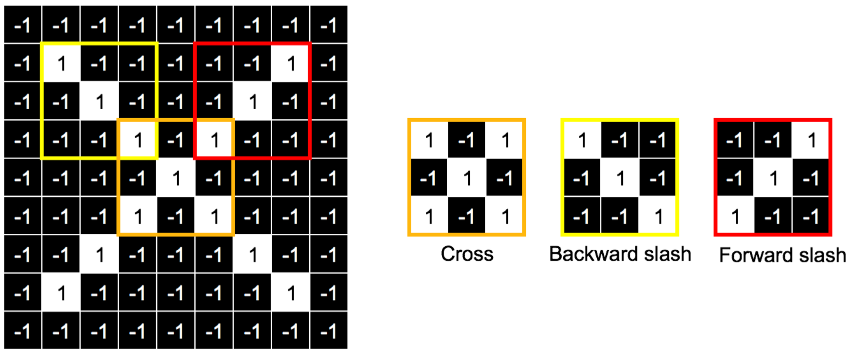


Figure 8. Representation of a picture of an “X” with the values corresponding to the color of each pixel. Numbers goes from -1 to 1 where the smallest number is the darkest. The patterns for detecting an “X” are extracted from the picture.

One of the filters is chosen and is compared to others blocks in the picture. A filter is compared to a square with the same number of cells. The filter is first compared to the top left square of the picture with the same size. The process will be repeated through the entire picture. After a comparison of a block, we move from the current position to the next block.

For determining the move from one block to another, the stride has to be taken into consideration. A stride represents the number of pixels that are shifted on from the current position. For example, if overlapping should be avoided when performing the comparison, the stride has to be at least as big as the length of the filter. With a filter 3x3, the stride has to be at least 3 for not having one pixel used in more than one block (figure 9). A filter is not necessary a 3x3 square, different sizes are possible. It is also possible to add a padding zero to the image. It adds some extra pixels around the picture with the value 0. The value of the padding zero represents the number of zero layer around the image. For example, in figure 9 with a stride of 3 the next shift after the comparison in blue needs a padding of 2 on the right in order to process to the next comparison. Figure 10 shows a visualization of a padding zero. The objective of a padding zero is to control the size of the output. By adding a padding, it is possible to obtain an output with the same size as the input size.

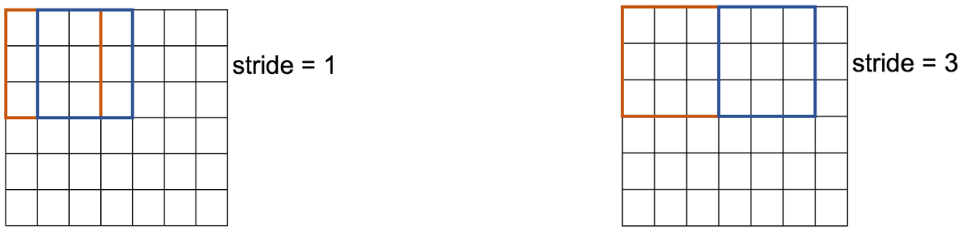


Figure 9. Visualization of a shift after a comparison with a stride 1 and a stride 3. A stride of the same size of the filter avoids overlapping.

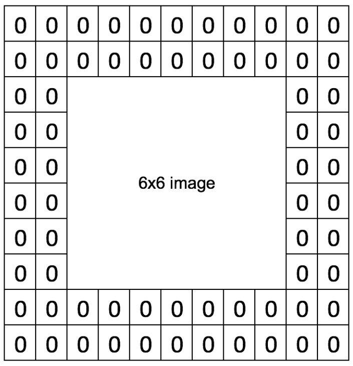


Figure 10. Visualization of a padding 2 surrounding a picture.

Each comparison will give one number as a result. It means after all the comparisons we have a new square much smaller than before. The size of the resulting square depends on different factors: the size of the image, the size of the filter, the value of the stride and the size of the padding. The following formula (figure 11) can be used to predict the size of the resulting square:

$$O = \frac{(w - k + 2p)}{s} + 1$$

$O = the\ ouput\ height/length$

$w = the\ input\ height/length$

$k = the\ filter\ size$

$p = the\ padding\ size$

$s = the\ value\ of\ the\ stride$

Figure 11. Output size formula

For example, if there is an input square of 9x9 with a filter 3x3, without any padding and a stride of 1, it will have an output of 7x7. Assumed that this square represents an “X”. We will try to match the forward slash filter (figure 8) to this picture. For comparing a filter with another block, each pixel in the filter is multiplied by the related pixel in the comparison block. The result is stored in a new array. Then, the sum of all numbers is calculated and finally, we get the average by dividing the sum with the number of pixels in the filter. The average is the final result for the comparison block and will be stored in the output square. Figure 12 illustrate the comparison between a filter and the comparison block. In this example, the comparison block is identical to the filter. The comparison of two identical pixels gives the value 1, thus comparing to an identical square will have a square full of 1 as a result, and the average is obviously 1. As a reminder, the pixels can be between -1 and 1, but for making the example easier to understand, only black and white pixels have been used.

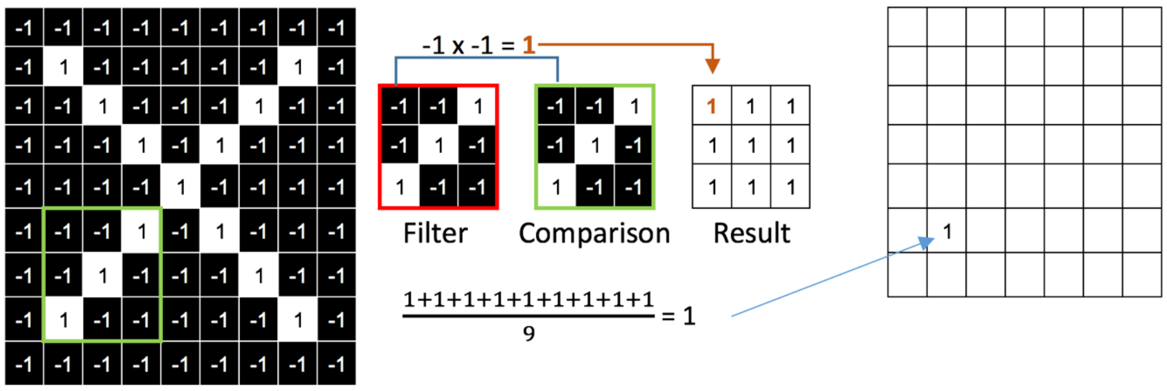


Figure 12. Visualization of a comparison between a filter and a block which is identical. The multiplication between each corresponding pixel is computed and the average of the resulting square is calculated. The average goes to the related pixel in the output square.

In figure 13 the backward slash filter is compared to a block which is not the same. Once again, the multiplication between the filter and the other square is performed. Then, the sum is calculated, and we finally get the average by dividing the sum by 9. The compared square was different this time, and we notice that the result is smaller with a value of 0.33.

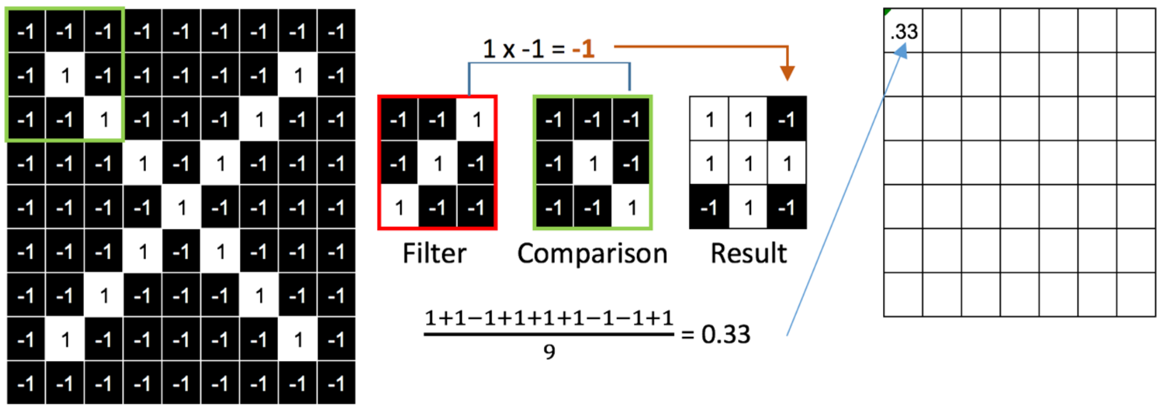


Figure 13. Visualization between a filter which is different to the compared block.

Once the process has been repeated for each block, figure 14 is obtained. The pattern backward slash is visible in the result which is the same pattern as the filter used for the comparisons. The result is a square 7x7 as predicted by the formula. The entire process is also performed for the other filters (figure 15). The result obtained with this process is called the convolution layer. This output can be used as input in another layer. There is no rule for choosing the hyperparameters which are the different elements used in the formula (figure 11). They depend on the type of data, the complexity of the images and other factors.

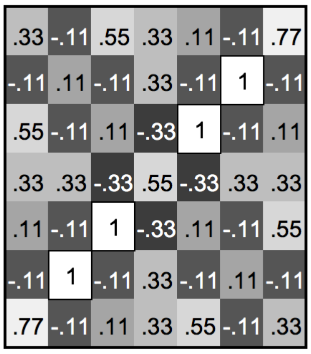


Figure 14. Visualization of the output square after comparing the filter backward slash through each block of the picture

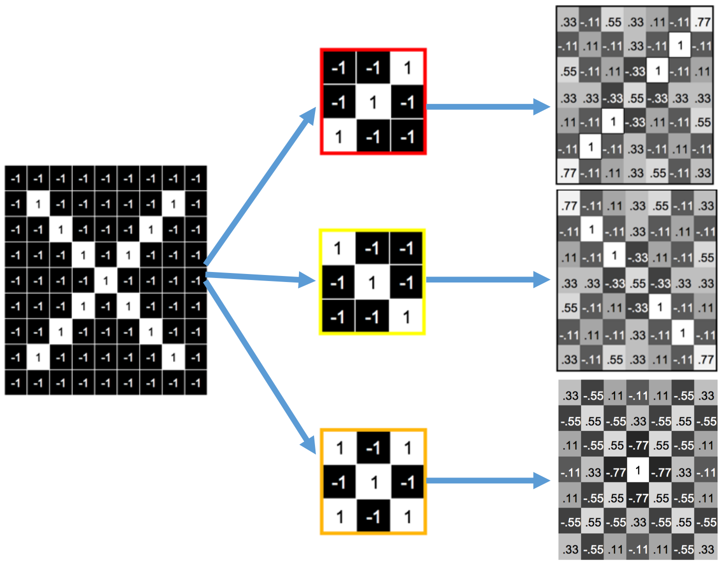
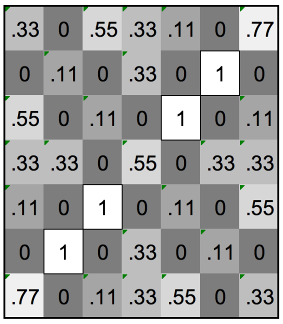


Figure 15. Visualization of the outputs square after comparing all the filters

Rectified Linear Units layer (ReLU)

After each convolutional layer, it is recommended to use an activation function. An activation function (Mestan, 2008) is a nonlinear function with the goal to add nonlinear properties in the network. If only linear functions are used in a network, the process could be summarized into one linear function. However, recognizing faces is not a linear problem, this is the reason why activation functions are required into the CNN. Generally, an activation function provides a number between 0 and 1. When the value is close to 1, it is called active and inactive when it is close to 0. The most popular are Sigmoid, Tanh and ReLu. In this example, a rectified linear units layer (Nair & Hinton, 2010) is used. The aim of this layer is to replace all the negative numbers by the value 0 (figure 16).

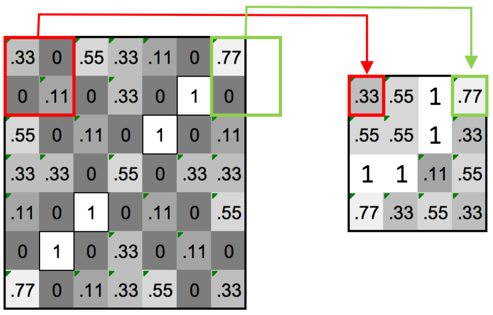


.33	0	.55	.33	.11	0	.77
0	.11	0	.33	0	1	0
.55	0	.11	0	1	0	.11
.33	.33	0	.55	0	.33	.33
.11	0	1	0	.11	0	.55
0	1	0	.33	0	.11	0
.77	0	.11	.33	.55	0	.33

Figure 16. Visualization of the ReLU layer. Replacement of the negative values by 0.

Maximum Pooling layer

This maximum pooling layer shrinks the previous layer that it receives as input. This image is split into 2x2 windows and only the maximum number in each window is kept (figure 17). A stride of 2 is used for this process. The max pooling reduces considerably the number of pixels while keeping the main information.



.33	0	.55	.33	.11	0	.77
0	.11	0	.33	0	1	0
.55	0	.11	0	1	0	.11
.33	.33	0	.55	0	.33	.33
.11	0	1	0	.11	0	.55
0	1	0	.33	0	.11	0
.77	0	.11	.33	.55	0	.33

.33	.55	1	.77
.55	.55	1	.33
1	1	.11	.55
.77	.33	.55	.33

Figure 17. Visualization of the max pooling process

If the window for the comparison is smaller than 2x2, we simply take the biggest number in the window. It is the same as adding only one column of zero padding for making the comparison. As in figure 17, we notice the green window contains only two pixels. Thus, the calculation is done as if there is a column of zero padding on the right of the image. The formula for predicting the size of the output square can be used, but needs to be adapted in this case. Instead of using $2p$ in the formula only p is used because only one side of the picture needs a padding zero. Maximum pooling is another layer of the convolutional neural network. As the other layers, it uses the output of the previous layer as input and then provides an output for another layer. Each layer can be repeated several times in a CNN.

Fully connected layer

After passing our initial picture through a certain number of layers, the data of the last layer is used in a fully connected layer. The three squares in figure 18 are the results of our previous example for each filter which went through another maximum pooling layer. In the fully connected layer, all the pixels do not have the same weight. Since the beginning, the picture of an "X" was used as input in the first layer. It means that the results in figure 18 are the results for predicting an "X". The bigger is the pixel bigger is the weight. The thickness of a line represents the number of votes. Thus, the bigger the value of a pixel is, the thicker is the line.

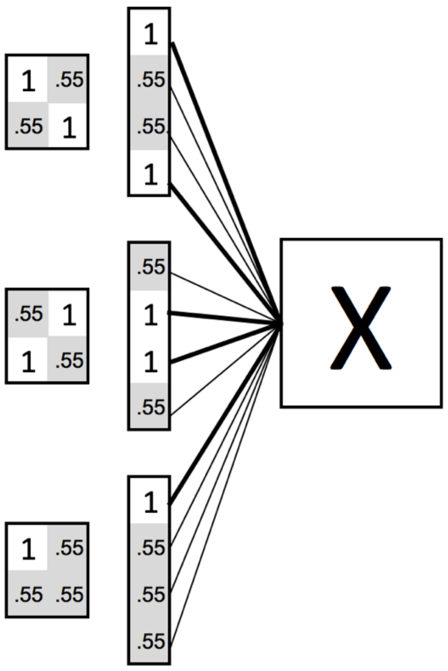


Figure 18. Visualization of the fully connected layer

The entire process is executed for all the classes. In this example there are only two class "X" and "O", therefore only "O" is missing. The process is exactly the same, but instead of having an "X" picture as input for the first layer it is an "O" (figure 19). The fully connected layer will give the most important cells for predicting an "O".

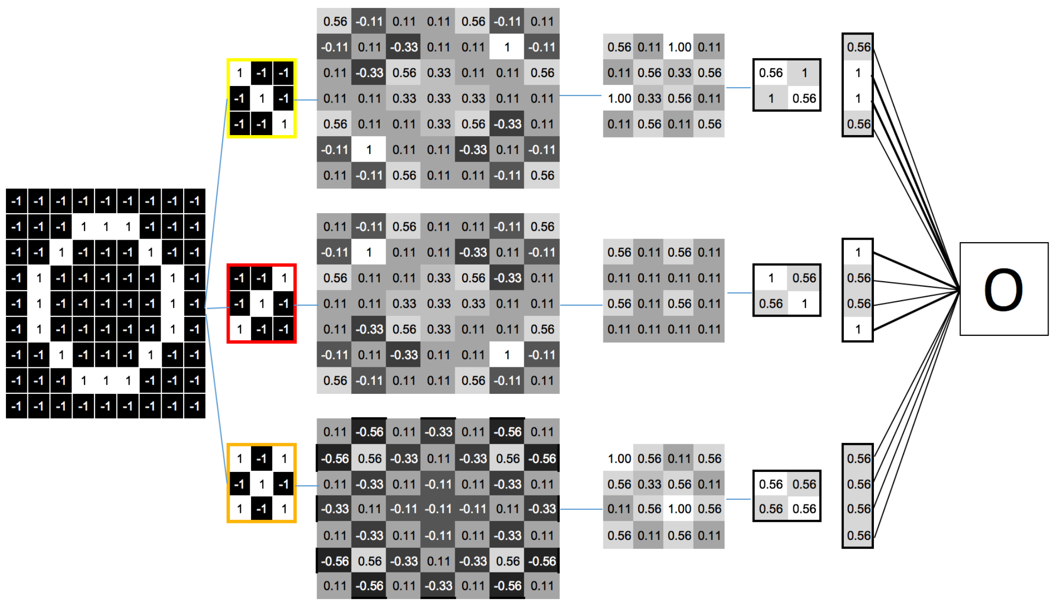


Figure 19. Full process for a picture "O" as input. The three filters are applied in a convolutional layer. Then, there is a ReLU layer which is not shown in this figure. After that, there are two layers of max pooling and finally one fully connected layer.

Then, for predicting if a new picture is an “X” or an “O”, the picture is passed through the convolutional neural network. In the fully connected layer, the average of the thickest lines for representing an “X” is calculated, and the same is done for the thickest lines concerning “O”. The average for “X” is represented by $\bar{x}$ and $\bar{o}$ for the average of “O”. Figure 20 shows the pixels with the most votes for each class. The symbol which has the biggest average is the prediction of the CNN.

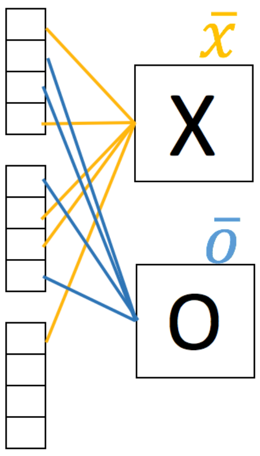


Figure 20. Calculation of the average for each class

Backprobagation

Backprobagation is a training process for adjusting the weight of the filters that have also been seen as votes for determining the representation of the image. The backprobagation can be divided into four parts: the forward pass, the loss function, the backward pass and the weight update. The forward pass is just the principle to pass an image through the convolutional neural network. The weights of the CNN are not really accurate during the first training because the weights have been chosen randomly and are almost the same everywhere. Obviously, a CNN will not give a correct answer if it has not been trained. The loss function is used to determine how wrong is our CNN. For the training process only labeled pictures are used. It means we can check if the predicted class of the CNN for the training machine has been identified correctly and also know how confident was the machine for this prediction. In our example of classifying the pictures into our two categories “X”, “O”, only one class represents in reality the object with a probability of 100% and 0% for the other one. A loss function is used in order to know if the machine is reliable or needs to be corrected. A function which is often used is called mean squared error (MSE):

$$E_{total} = \sum \frac{1}{2} (target - output)^2$$

$$target = the\ real\ result$$

$$output = \ the\ prediction$$

As we said before, the number of errors represented by E_{total} is high for the training. The aim is to get a result which is the closest to reality. We can take for example a picture which represents an “X” and gives as output the value 0.9 for “X” and 0.55 for “O”. Figure 21 is a representation of this situation where the errors have been calculated with the MSE. The error value for “O” is considerably high, the goal is to reduce the total number of errors.

	target	output	error
X	1	0.9	0.005
O	0	0.55	0.15125
Total			0.15625

Figure 21. Representation of the MSE for the two classes

However, 100% success, thus a total error equals to 0 doesn't always mean that the training has been performed correctly. The problem that might occur when the machine has a of result 100% is that the machine could have learned the training data by heart and might be completely wrong with a picture external to the training set. However, the aim is to minimize the number of errors in order to get as close as possible to this percentage. For that it is needed to change the weights, but not all the weights. Thus, it is necessary to identify which weights need to be updated.

The number of errors is directly related to the weights. Figure 22 is the gradient descent which shows the correlation between error and weight. If a weight is too high or too small the number of errors will be extremely high. The aim is to adjust the initial weight in order to reach the desired weight that offers the smallest number of errors. The change of weight must not be too big. Otherwise, the desired weight will be missed. But if the change is too small, it will be very expensive in calculus for the machine. The role of the backward pass is to determine which weights are the most responsible for the number of errors. Finally, the aim of the last step "weight update" is, as the name says it to update the weights in order to move in the direction of the desired weights. The desired weight as shown in figure 22 is the weight which is related to the minimum error. The combination of these four steps is called epoch, a certain number of epoch is performed for each picture of the training set. Afterwards, the CNN should be trained enough and should provide reliable results.

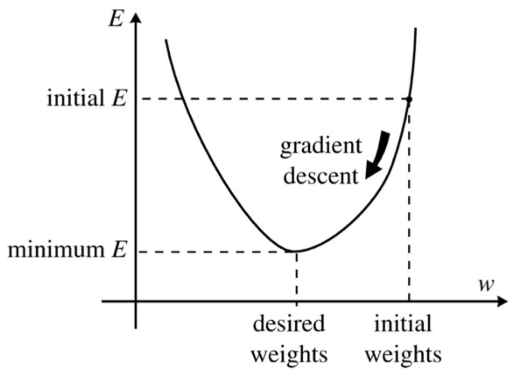


Figure 22. Gradient descent. This graph represents the relation between the errors and the weights. The aim is to move slightly from the initial point until reaching the desired point.

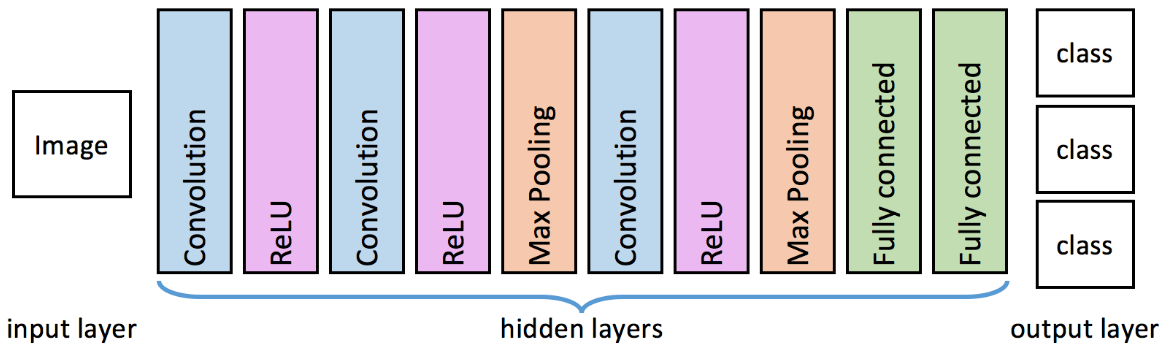


Figure 23. Example of the structure of a potential Convolutional neural network. Each layer can be used several times.

This was an overview of the main principles of a convolutional neural network. Each network can have a different structure, as we can see in figure 23. Each type of layer can be repeated several times in the process. Some can use different pooling than maximum pooling, another loss function, an inception module.

This whole process was used for predicting whether the picture is more an “X” or “O”, but we are interested in recognizing faces which are more complex than these two symbols. The concept is exactly the same if the inputs are faces. Filters will be applied through the picture, and the rest is the same. However, it is easier to find filters for representing an “X” than representing a face. It is crucial to use suitable filters for face recognition. The human is more concentrated on the mouth, the nose and the eyes for recognizing a person, but this is not the case for the machine. The machine is more capable of identifying by itself the main features representing a face. A deep neural network (DNN) is used for extracting the main features (Lee, Grosse, Ranganath, & Ng, 2009). Then, the outputs of the DNN is used as filters in the convolutional neural network.

Transfer learning

Training a deep neural network requires a huge number of faces which is not accessible to everyone. However, there is an alternative if we cannot have enough data for training a DNN which is called transfer learning. The concept of transfer learning is to use a pre-trained model and customize it with our data. A DNN is also based on layers and the first layers are used for detecting curves and edges which is necessary for almost all networks. The last layer is replaced with our own data. Each weight present in the pre-trained model will stay unchanged. The new version can be trained normally, and expected outputs can be retrieved.

3.3.5 OpenFace

OpenFace (Amos, Ludwiczuk, & Satyanarayanan, 2016) is a face recognition library. It is based on Google's FaceNet (Schroff & al.) systems. OpenFace is using a deep convolutional neural network for performing a facial recognition. It uses a modified version of the network nn4 from FaceNet. OpenFace was trained with 500k images. As shown in figure 24 the structure is divided into two parts. On the left, there is a deep neural network used for extracting the features. This part needs to be performed only once. OpenFace has already done this phase. Thus, the result can directly be used in the second part of the structure. The second part of the structure requires several pictures per subject. The faces are detected and extracted from the pictures with dlib's pre-trained detector that uses a HOG. Then, the faces go through a preprocessing phase and are finally used in the convolutional neural network. The CNN uses the features extracted in the deep neural network from the left part of the structure (figure 24) as filters. During the training phase, the CNN is adapted according to the different classes. For predicting the person of an unknown phase, a picture is passed through the right part of the structure, and the class which represents the person will be given as output.

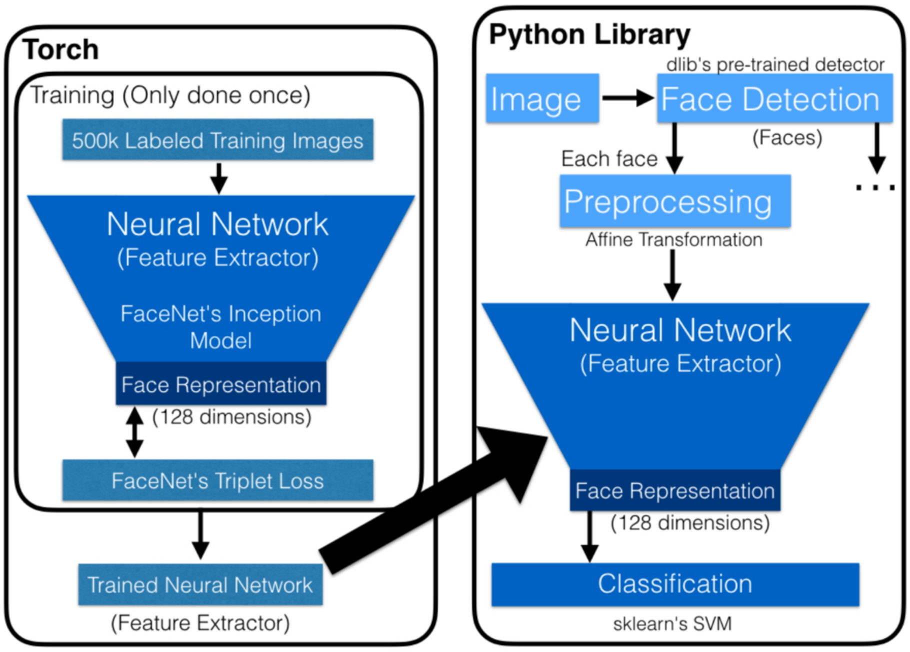


Figure 24. OpenFace's project structure (Amos & al., 2016)

4 Tests design

For comparing the algorithms, different factors can be tested. Several tests will be run with different modifications in the training set such as the number of subjects, the number of pictures per subject and a change in the luminosity. The pictures in the training set are faces with various emotional gestures. The algorithms which will be used for the test are Eigenfaces, Fisherfaces, local binary patterns histograms and a project using a deep convolutional neural network.

The same training set and the same test set are used for training the algorithms. The test set consists of some pictures from each subject that were not included in the training set. The test will be run with two types of test data. In the first one, the training data of a person will be in the same environment as the test data. It means the pictures are taken in the same room and with the same light. In the second one, the pictures from the training data and the test data will not be from the same environment. The pictures could be taken in another location and where the luminosity is variable. The aim of splitting the tests into these two types is in a first phase testing how accurate the algorithm is in the same environment with the same luminosity. In the second phase, it will be interesting to see the reaction of the algorithms with pictures from another place. The robustness of the algorithms will be tested this way. For these two phases, it will be interesting to analyze the evolution of the predictions concerning the pictures present in all the tests.

All the pictures in the test set are labeled. Thus, it will be possible to calculate the percentage of error each algorithm has made concerning this test set. If the prediction of the algorithm is equivalent to the label of the test picture, the algorithm predicted the person correctly. Otherwise, it is considered as an error. The aim of these tests is to compare the accuracy of these algorithms through the different factors which have been enumerated before and find the most reliable algorithm.

5 Tests implementation

All the pictures from the training data have been taken with a camera Nikon D3100. They have been pre-processed in the same way. The faces have been detected with a Histogram of Oriented Gradients. Then, the landmark detector of the dlib library was used to center the face into the picture based on the nose, and the picture was rotated to have the eyes on the same level. After that the images were cropped based on the external landmarks of the faces and resized into a 96x96 dimension.

The algorithms used for the tests are Eigenfaces, Fisherfaces and local binary patterns histograms which all come from the library OpenCV. Eigenfaces and Fisherfaces are used with a Euclidean distance to predict the person. The last algorithm which is using a deep convolutional neural network is the project called OpenFace (Amos & al., 2016). Each algorithm had three categories of tests (figure 25). The first category has 5 subjects where all the pictures have been taken in the same environment and with the same light. The second category has 10 subjects and the third one has 15 subjects. The number of selected subjects per sample is based on an interval five to make the comparisons more visible. The pictures from the categories two and three were not all taken in the same environment with the same luminosity, but all the pictures of a class were in the same conditions. Each category is divided into three tests. The difference between these tests is the number of pictures per subject in the training set. One has 10 pictures per person, another one with 20 pictures per person and the last one with 40 pictures per person.

First, for each test, 5 pictures per subject were used. The pictures from the test set were taken at the same time as the one for the training data. Thus, the environment of the training set and the test set is the same concerning a person. Figure 25 shows the representations of all the tests. Then, all tests have been performed a second time with two pictures per subject. These pictures were taken in a completely different environment than the training data.

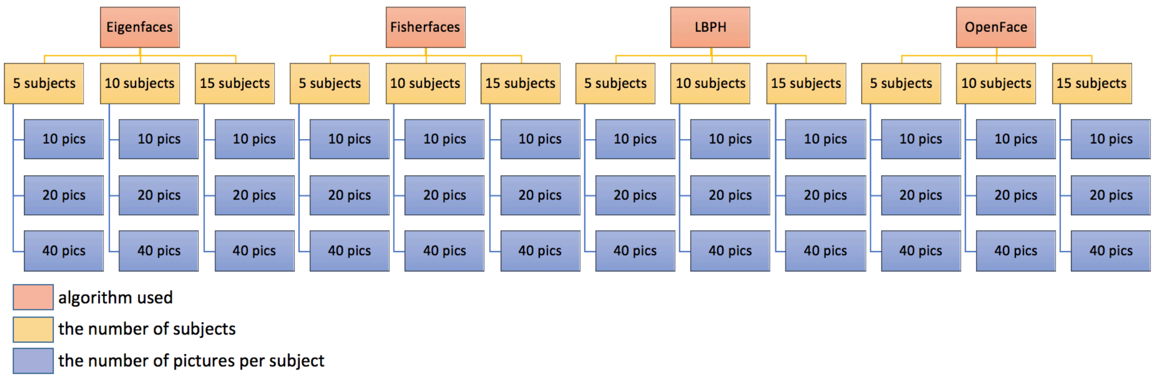


Figure 25. Representation of the planned tests

6 Tests results and analysis

Figures from 26 to 29 represent the result obtained by the test implementation part. Figures from 30 to 33 concern the tests that have the same environment of the training pictures and the tests pictures. These tests are part of the first phase explained in chapter 4. The graphs show the percentage of accuracy for each test. This percentage is calculated by comparing the number of pictures that have been correctly recognized and the total number of pictures tested. The first observation that can be done in figure 26 is the increase of the accuracy when the number of pictures per person is also increasing. All the algorithms were able to recognize all the pictures correctly when at least 20 pictures per person were used to train the algorithms. The two algorithms with a statistical approach were a bit less accurate than the others with 10 pictures per person for the training. However, they still get an accuracy higher than 90%.

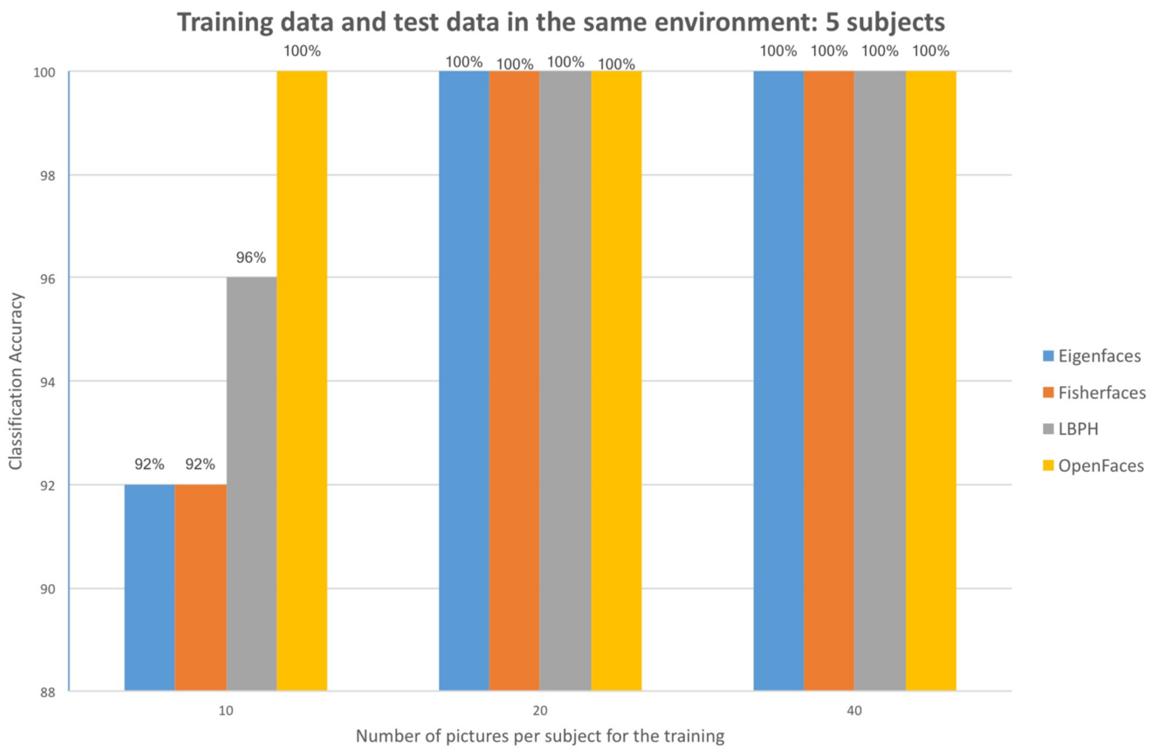


Figure 26. Tests results with 5 subjects and the same environment of the training data and test data. 5 pictures were tested per person.

In figure 27 the accuracy with 10 pictures for the training has increased compared to the previous one (figure 26). It is important to notice that the number of tests doubled because 10 subjects were used for those tests (figure 27). Fisherfaces is the only algorithm that did not recognize correctly all the subjects with 20 pictures for the training phase. The ranking of the algorithm related to the accuracy is the same as the previous figure. The same observation can be noticed on figure 26, the bigger the training set is, the more accurate the result is. Figure 28 represents only the results of the test data present in the 5 subjects tests which are also in the other tests. This figure shows how the same pictures have been affected by a change of the number of subjects. An increase from 5 subjects to 10 transformed a correct prediction into a wrong one for Fisherfaces (training: 20 pics per subj). However, the other predictions are stable.

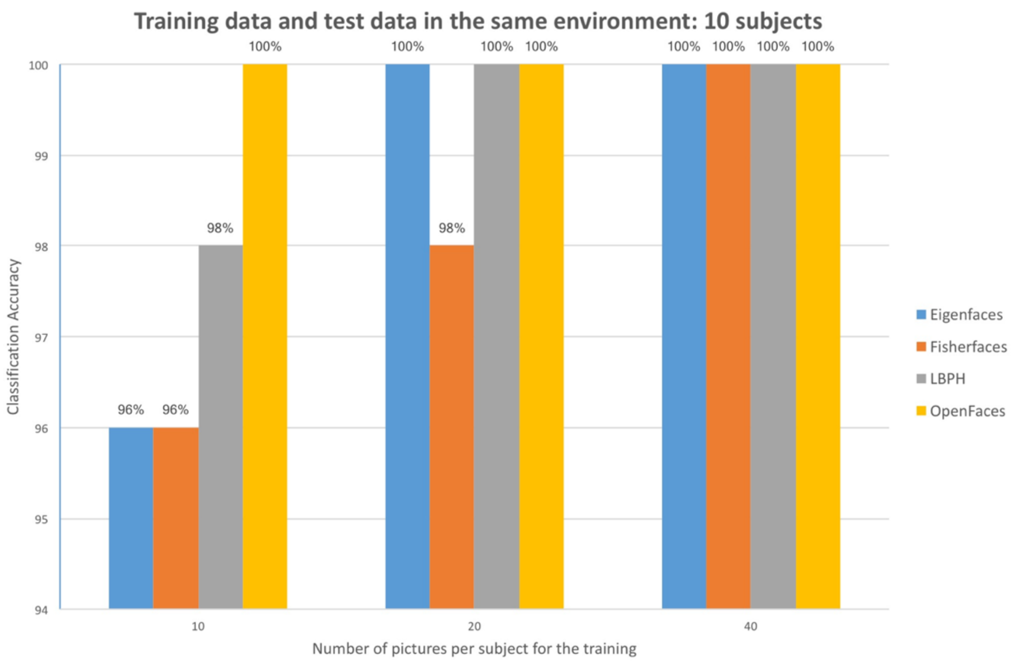


Figure 27. Tests results with 10 subjects and the same environment of the training data and test data. 5 pictures were tested per person.

	5 subjects								
	Training: 10 pics per subj.			Training: 20 pics per subj.			Training: 40 pics per subj.		
	Correct	Error	Result	Correct	Error	Result	Correct	Error	Result
Eigenfaces	23 pics	2 pics	92 %	25 pics	0 pics	100 %	25 pics	0 pics	100 %
Fisherfaces	23 pics	2 pics	92 %	25 pics	0 pics	100 %	25 pics	0 pics	100 %
LBPH	24 pics	1 pics	96 %	25 pics	0 pics	100 %	25 pics	0 pics	100 %
OpenFace	25 pics	0 pics	100 %	25 pics	0 pics	100 %	25 pics	0 pics	100 %

	10 subjects								
	Training: 10 pics per subj.			Training: 20 pics per subj.			Training: 40 pics per subj.		
	Correct	Error	Result	Correct	Error	Result	Correct	Error	Result
Eigenfaces	23 pics	2 pics	92 %	25 pics	0 pics	100 %	25 pics	0 pics	100 %
Fisherfaces	23 pics	2 pics	92 %	24 pics	1 pics	96 %	25 pics	0 pics	100 %
LBPH	24 pics	1 pics	96 %	25 pics	0 pics	100 %	25 pics	0 pics	100 %
OpenFace	25 pics	0 pics	100 %	25 pics	0 pics	100 %	25 pics	0 pics	100 %

	15 subjects								
	Training: 10 pics per subj.			Training: 20 pics per subj.			Training: 40 pics per subj.		
	Correct	Error	Result	Correct	Error	Result	Correct	Error	Result
Eigenfaces	23 pics	2 pics	92 %	25 pics	0 pics	100 %	25 pics	0 pics	100 %
Fisherfaces	23 pics	2 pics	92 %	24 pics	1 pics	96 %	24 pics	1 pics	96 %
LBPH	24 pics	1 pics	96 %	25 pics	0 pics	100 %	25 pics	0 pics	100 %
OpenFace	25 pics	0 pics	100 %	25 pics	0 pics	100 %	25 pics	0 pics	100 %

Figure 28. These tables represent the results obtained only for all the pictures contained in the tests with 5 subjects where the pictures were taken in the same environment. All these images are also present in the tests with more subjects.

Figure 29 consolidates the idea that a large amount of data improves the accuracy. Once again, the two algorithms with a statistical approach are the least accurate. Fisherfaces had the worst result. LBPH is the second best, and the leader is OpenFace with an accuracy of 100% in each test. Only Fisherfaces has been affected by a change in the number of subjects for the same pictures (figure 28). However, the accuracy of all the algorithms has never been below 92%.

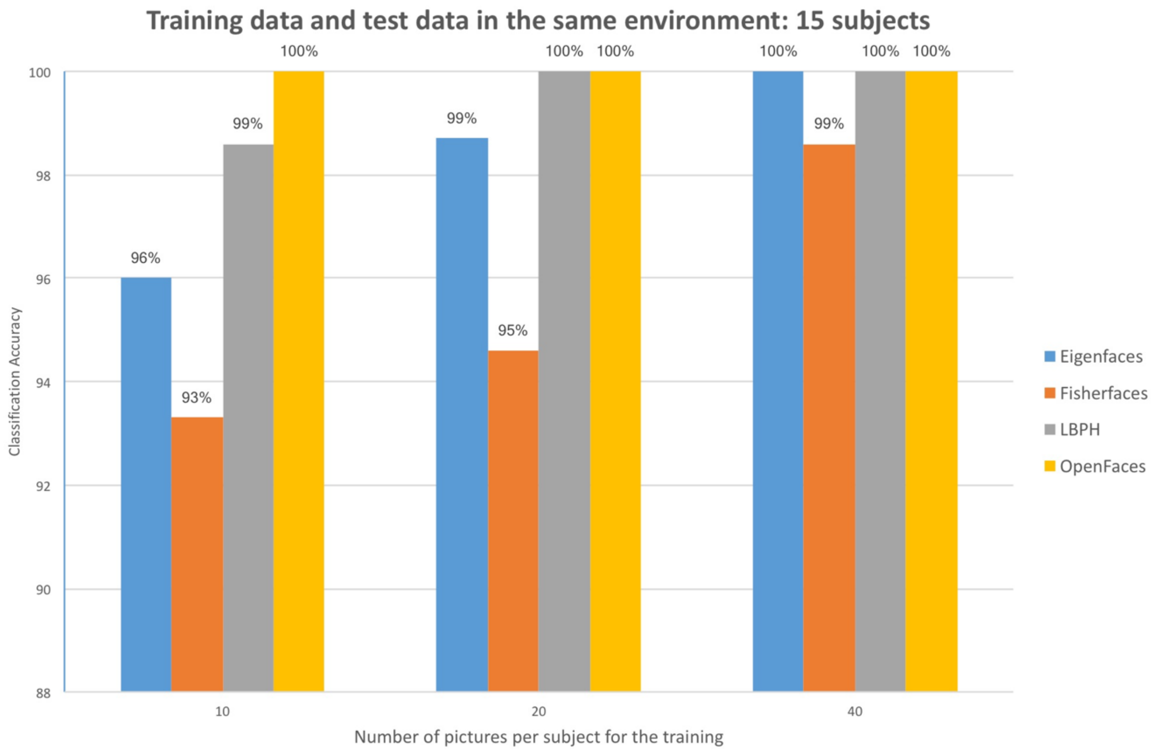


Figure 29. Tests results with 15 subjects and the same environment of the training data and test data. 5 pictures were tested per person.

Figures from 30 to 33 concern the results of tests which have a different environment between the training set and the test set. These figures represent the second phase explained in chapter 4. These tests were performed with two pictures per subject in the test set. A big drop can be noticed in figure 30 compare to the first phase for all the algorithms except OpenFace. OpenFace is constant and keeps an accuracy of 100%. This algorithm has not been affected by the change of the environment for the moment. For the first time, an algorithm got worse accuracy after an increase in the training data. This concerned only Fisherfaces. The others stayed the same or had an improvement when this factor became bigger. The two algorithms with a statistical approach were better than LBPH. LBPH had the worst results with less than 50%. A consistent gap between OpenFace and the other algorithms can be noticed.

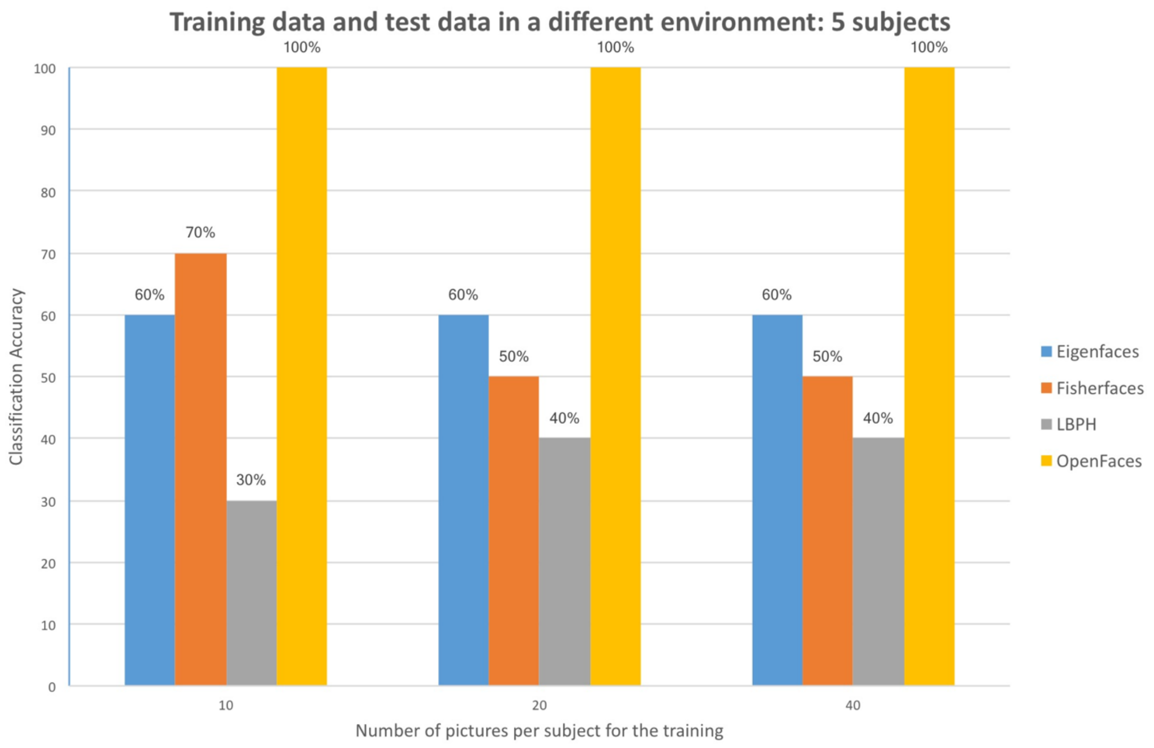


Figure 30. Tests results with 5 subjects and a different environment of the training data and test data. 2 pictures were tested per person.

The results continued to drop in figure 31. The results of the three algorithms affected by the change were almost halved compared to the previous tests (figure 30). These three algorithms did not have an accuracy higher than 30%. Fisherfaces even reached an accuracy of 10 percent with 10 pictures as training. This result represents the same percentage as picking up a person randomly and use it as a prediction. OpenFace is still great with a perfect recognition. An increase in the training data had a positive effect for Fisherfaces and LBPH.

The change from 5 to 10 subjects had considerably affected the predictions for the same pictures (figure 32). Eigenfaces lost 20% of accuracy, Fisherfaces lost 50% of accuracy concerning the test with a training of 10 pictures per subject and LBPH was not able to recognize any picture. An increase in the size of training set improved slightly the accuracy of LBPH. Concerning Fisherfaces, its behavior is not constant when the training set becomes bigger. The reaction was positive with 20 pictures per subject. However, the accuracy decreased with 40 pictures. Eigenfaces was not affected by this factor.

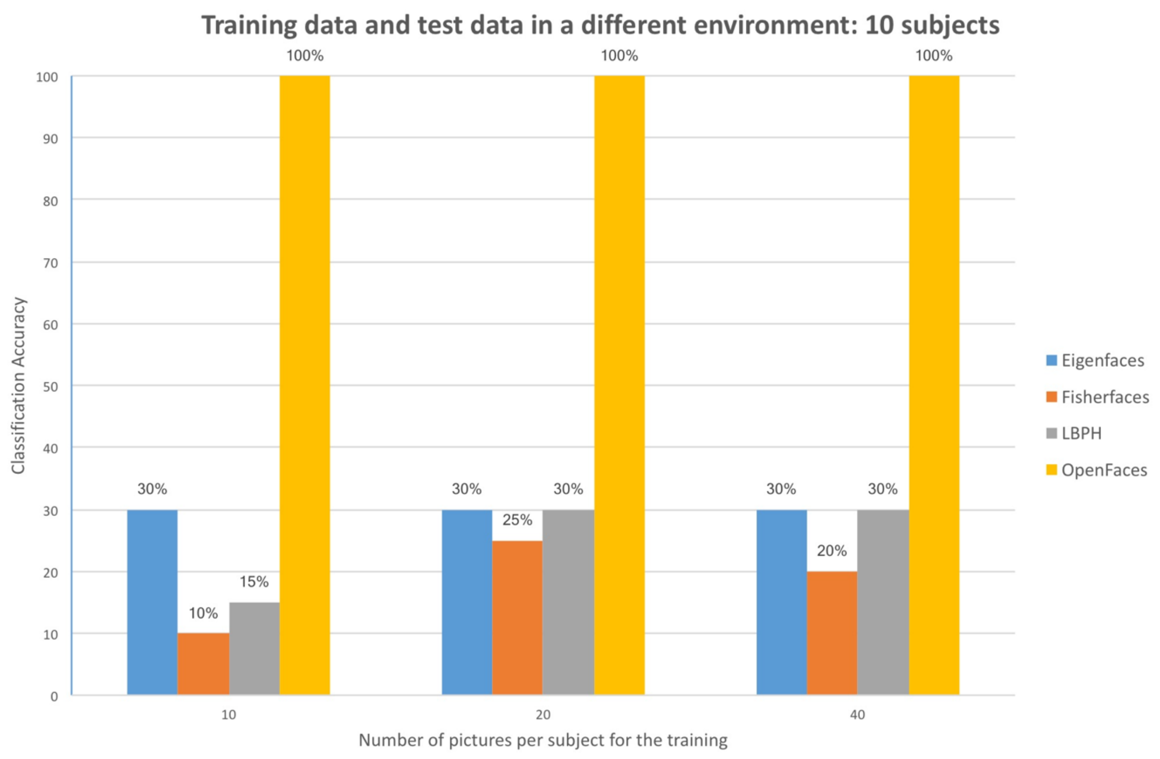


Figure 31. Tests results with 10 subjects and a different environment of the training data and test data. 2 pictures were tested per person.

	5 subjects								
	Training: 10 pics per subj.			Training: 20 pics per subj.			Training: 40 pics per subj.		
	Correct	Error	Result	Correct	Error	Result	Correct	Error	Result
Eigenfaces	6 pics	4 pics	60 %	6 pics	4 pics	60 %	6 pics	4 pics	60 %
Fisherfaces	7 pics	3 pics	70 %	5 pics	5 pics	50 %	5 pics	5 pics	50 %
LBPH	3 pics	7 pics	30 %	4 pics	6 pics	40 %	4 pics	6 pics	40 %
OpenFace	10 pics	0 pics	100 %	10 pics	0 pics	100 %	10 pics	0 pics	100 %

	10 subjects								
	Training: 10 pics per subj.			Training: 20 pics per subj.			Training: 40 pics per subj.		
	Correct	Error	Result	Correct	Error	Result	Correct	Error	Result
Eigenfaces	4 pics	6 pics	40 %	4 pics	6 pics	40 %	4 pics	6 pics	40 %
Fisherfaces	2 pics	8 pics	20 %	5 pics	5 pics	50 %	3 pics	7 pics	30 %
LBPH	0 pics	10 pics	0 %	1 pics	9 pics	10 %	2 pics	8 pics	20 %
OpenFace	10 pics	0 pics	100 %	10 pics	0 pics	100 %	10 pics	0 pics	100 %

	15 subjects								
	Training: 10 pics per subj.			Training: 20 pics per subj.			Training: 40 pics per subj.		
	Correct	Error	Result	Correct	Error	Result	Correct	Error	Result
Eigenfaces	2 pics	8 pics	20 %	2 pics	8 pics	20 %	2 pics	8 pics	20 %
Fisherfaces	2 pics	8 pics	20 %	1 pics	9 pics	10 %	1 pics	9 pics	10 %
LBPH	1 pics	9 pics	10 %	3 pics	7 pics	30 %	3 pics	7 pics	30 %
OpenFace	8 pics	2 pics	80 %	8 pics	2 pics	80 %	9 pics	1 pics	90 %

Figure 32. Tests results with 10 subjects and a different environment of the training data and test data. Two pictures were tested per person.

For the first time, OpenFace did not reach 100% of accuracy (figure 33). However, OpenFace is better than the three other algorithms together. The others became even less accurate with less than 25%. All the algorithms have better results with a larger amount of data except Fisherfaces, which sometimes tends to decrease slightly. Figure 32 demonstrated that some pictures are tested with a few number of people will become harder to be correctly recognized when the number of subjects is bigger.

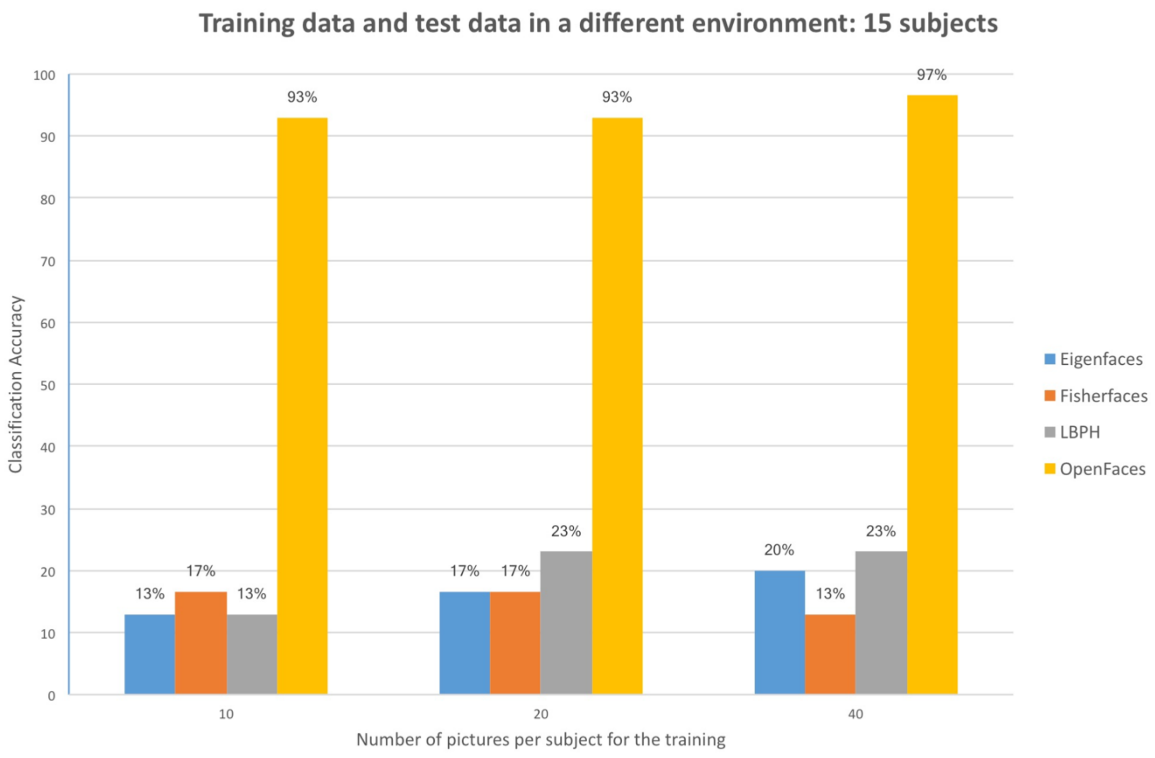


Figure 33. Tests results with 15 subjects and a different environment of the training data and test data. 2 pictures were tested per person.

7 Discussion and conclusion

The experiment demonstrates that all these algorithms are relatively accurate. However, OpenFace is more accurate and reliable in comparison with the others.

Eigenfaces was not very sensitive to a change in the number of subjects during the first phase, however an increase in size of the training set helped the algorithm to correct its wrong prediction. This reflection is the opposite with the tests in another environment (figure 32). An increase in the data set did not help to recognize more subjects, but it turned correct predictions into wrong ones. Eigenface was not accurate in the second phase.

Fisherfaces had better results in the first phase with a larger amount of data. However, its behavior was different in the second phase. Sometimes with 20 pictures, the results were better, but with 40 pictures the results were the same or worse. This algorithm was most of the time the worst in the two phases. It also had a very low accuracy in the second phase.

LBPH always had at least 96% in the first phase and was the second-best algorithm. In the second phase, this algorithm had a consequent drop in its accuracy compared to the first phase. It was slightly less accurate than Eigenfaces. An increase in the number of subjects dramatically changed its prediction. In each phase, an increase in the training data had a positive effect or no effect.

OpenFace was so far the best algorithm. It recognized perfectly all the pictures in the first phase and did the same with 5 and 10 subjects in the second phase. It only had some errors in the second phase with 15 subjects. These errors were minimized with a larger training data. OpenFace was only lightly impacted by a change in the environment compared to the others which had difficulties to recognize the pictures.

Based on these results, we can conclude that using a convolutional neural network is more efficient for performing a facial recognition than a statistical approach or searching for patterns. As it has been explained before, a CNN does not have a unique structure, but can be customized. It is likely that OpenFace can be optimized by testing different hyperparameters and different combinations of hidden layers. Increasing the amount of data in the data set for extracting the features is also a way to potentially improve the results. Another topic, which is using the same approach as facial recognition is emotion recognition. Instead of differentiating persons, the aim is to recognize the emotion of a person.

The concept of class is also used, although instead of having a person representing a class it is a facial expression such as happy, sad, surprised, fear, anger or neutral. Fisherface or a convolutional neural network could be used for recognizing emotions due to their approach to cluster pictures. However, what accuracy can be expected for emotion recognition? Could Fisherface or OpenFace provide satisfying results? This question will try to be answered in another paper.

References

Ahonen, T., Hadid, A., & Pietik, M. (2004). Face Recognition with Local Binary Patterns, 469–481.

Amos, B., Ludwiczuk, B., & Satyanarayanan, M. (2016). OpenFace: A general-purpose face recognition library with mobile applications, (June). Retrieved from http://cmusatyalab.github.io/openface/

Arubas, E. (n.d.). haar-all.jpg (100×175). Retrieved 17 May 2017, from http://eyalarubas.com/images/face-detection-and-recognition/haar-all.jpg

Barnouti, N. H. (2016). Improve Face Recognition Rate Using Different Image Pre-Processing Techniques Nawaf Hazim Barnouti American Journal of Engineering Research (AJER), (4), 46–53.

Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs . Fisherfaces : Recognition Using Class Specific Linear Projection, *19*(7), 711–720.

Bishop, C. M. (2005). Neural networks for pattern recognition. *Journal of the American Statistical Association*, *92*, 482. https://doi.org/10.2307/2965437

CS231n. (n.d.). CS231n Convolutional Neural Networks for Visual Recognition. Retrieved 12 May 2017, from http://cs231n.github.io/convolutional-networks/

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, *I*, 886–893. https://doi.org/10.1109/CVPR.2005.177

deeplearning4j. (n.d.). Introduction to Deep Neural Networks - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM. Retrieved 19 May 2017, from https://deeplearning4j.org/neuralnet-overview#concept

Deshpande, A. (2016a). A Beginner's Guide To Understanding Convolutional Neural Networks – Adit Deshpande – CS Undergrad at UCLA ('19). Retrieved 11 May 2017, from https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/?utm_source=mybridge&utm_medium=email&utm_campaign=read_more

Deshpande, A. (2016b). A Beginner's Guide To Understanding Convolutional Neural Networks Part 2 – Adit Deshpande – CS Undergrad at UCLA ('19). Retrieved 11 May 2017, from https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

Deshpande, A. (2016c). The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3) – Adit Deshpande – CS Undergrad at UCLA ('19), 1–25. Retrieved from https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html face_fhog_filters.png (180×180). (n.d.). Retrieved 17 May 2017, from http://1.bp.blogspot.com/-pPgDErLVJ_k/UvBGZk22ZXI/AAAAAAAAALs/c0mJmAVZnQE/s1600/face_fhog_filters.png

Geitgey, A. (2016). Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning. Retrieved 11 May 2017, from https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78

Jaiswal, S., Gwalior, M. I. T. S., & Gwalior, M. I. T. S. (2011). Comparison between face recognition algorithm-eigenfaces, fisherfaces and elastic bunch graph matching, *2*(7), 187–193.

Kazemi, V., & Sullivan, J. (n.d.). One Millisecond Face Alignment with an Ensemble of Regression Trees. https://doi.org/10.13140/2.1.1212.2243

King, D. E. (2009). Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*, *10*, 1755–1758. https://doi.org/10.1145/1577069.1755843

King Davis. (2014). dlib C++ Library: Dlib 18.6 released: Make your own object detector! Retrieved 12 May 2017, from http://blog.dlib.net/2014/02/dlib-186-released-make-your-own-object.html

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, 1–8. https://doi.org/10.1145/1553374.1553453

Lone, M. A., Zakariya, S. M., & Ali, R. (2011). Automatic face recognition system by combining four individual algorithms. In *Proceedings - 2011 International Conference on Computational Intelligence and Communication Systems, CICN 2011* (pp. 222–226). https://doi.org/10.1109/CICN.2011.44

Mäenpää, T., Pietikäinen, M., & Ojala, T. (2000). Texture classification by multi-predicate local binary pattern\noperators. *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, *3*, 3–6. https://doi.org/10.1109/ICPR.2000.903699

Mestan, A. (2008). Introduction aux Réseaux de Neurones Artificiels Feed Forward. Retrieved 20 May 2017, from http://alp.developpez.com/tutoriels/intelligence-artificielle/reseaux-de-neurones/#LIII-2

Morizet, N., Ea, T., Rossant, F., Amiel, F. D. R., & Amara, A. (2006). Revue des algorithmes PCA, LDA et EBGM utilisés en reconnaissance 2D du visage pour la biométrie, 13. Retrieved from papers3://publication/uuid/1CA253B2-0344-45CF-9F40-1E3A0ECF2F18

Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, (3), 807–814. https://doi.org/10.1.1.165.6419

Pyimagesearch. (n.d.). facial_landmarks_68markup-1024x825.jpg (1024×825). Retrieved 17 May 2017, from http://www.pyimagesearch.com/wp-content/uploads/2017/04/facial_landmarks_68markup-1024x825.jpg

Schroff, F., Kalenichenko, D., & Philbin, J. (n.d.). FaceNet : A Unified Embedding for Face Recognition and Clustering.

Taigman, Y., Yang, M., & Ranzato, M. A. (2014). Deepface: Closing the gap to humal-level performance in face verification. *CVPR IEEE Conference*, 1701–1708. https://doi.org/10.1109/CVPR.2014.220

Turk, M., & Pentland, A. (1991). Eigenfaces for Face Detection / Recognition. *Journal of Cognitive Neuroscience*, *3*(1), 1–11. https://doi.org/10.1162/jocn.1991.3.1.71

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition (CVPR)*, *1*, I--511--I--518. https://doi.org/10.1109/CVPR.2001.990517

Wagner, P. (2011). Local Binary Patterns. Retrieved 11 May 2017, from http://bytefish.de/blog/local_binary_patterns/

Weinberger, M. (2017). Facebook vs. Snapchat in computer vision - Business Insider. Retrieved 11 May 2017, from http://www.businessinsider.com/facebook-vs-snapchat-in-computer-vision-2017-3

what-when-how. (n.d.). Local Representation of Facial Features (Face Image Modeling and Representation) (Face Recognition) Part 1. Retrieved 11 May 2017, from http://what-when-how.com/face-recognition/local-representation-of-facial-features-face-image-modeling-and-representation-face-recognition-part-1/

Wilson, P. I., & Fernandez, J. (2006). Facial Feature Detection Using Haar Classifiers. *Journal of Computing Sciences in Colleges*, *21*(4), 127–133. https://doi.org/10.1109/CVPR.2001.990517