AGH

University of Science and Technology in Cracow

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

Department of Biocybernetics and Biomedical Engineering

Master of Science Thesis

Magda Nowak-Trzos

A comparative analysis and evaluation of various machine learning algorithms for facial recognition

Analiza porównawcza wybranych algorytmów uczenia maszynowego do rozpoznawania twarzy

Supervisor:
Adrian Horzyk Ph.D

Cracow 2018

Uprzedzona o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzona o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałam osobiście i samodzielnie i że nie korzystałam ze źródeł innych niż wymienione w pracy.

I would like to thank my mentor Adrian Horzyk Ph.D for his motivation and the continuous support of my master thesis and research.

Abstract

This paper provides an evaluation of various machine learning algorithms concerning the problem of facial recognition. A survey of available facial recognition machine learning algorithms will be presented. The work will focus on comparing four algorithms: PCA (so-called eigenfaces algorithm), Multilayered Perception algorithm, Convolutional Neural Network and Support Vector Machine. After the proper algorithm analysis the results of tests will be presented and compared.

Contents

1. Introduction

Facial recognition systems are applications capable of verifying or identifying a person based on a digital image. They have a wide range of uses such as identity authentication or security and access control. Interest in this topic has increased significantly over the past decade because it has potential to be the least invasive and user-friendly approach to human identification. Facial recognition applications are becoming less expensive, making their use more widespread. Benefits of automatic facial recognition systems can be seen not only in security systems but also in commercial identification systems and marketing tools. We can unlock our phones using the facial recognition system, social media can tag people automatically on the uploaded pictures, even some dating sites match people with the same facial features using the theory that people are most attracted to those that look like them.

Although people seem to recognize faces relatively easy, automatic recognition is a much more daunting task. A capable face recognition system should be able to deal with variations of face images in viewpoint, illumination, and facial expression. This makes face recognition a great challenge for contemporary computer systems.

To visualize how hard this problem can be, some pictures of the same individual can be seen in *Figure 1.1*. Even though they all belong to the same person, they are hardly recognized as such, even by a human.

Figure 1.1: Very different pictures of a single individual

Facial recognition is an active area of research, spanning several disciplines such as image processing, pattern recognition, computer vision, and neural networks. Engineers started to show interest in face recognition in the mid-1960's. One of the pioneers in this topic was Woodrow W. Bledsoe. He developed a semi-automatic system that could classify photos of faces by hand using so-called a "RAND tablet" - a device that people could use to input horizontal and vertical coordinates on a grid using a stylus that emitted electromagnetic pulses. The system was used to record the coordinate locations of various facial features such as the eyes, nose, hairline or mouth. Computers used gathered information in further facial recognition.

In 1988, Sirovich and Kirby [10] applied the linear algebra to the problem of facial recognition, introducing the Eigenface approach. They showed that feature analysis on a collection of facial images could form a set of basic features. They were also able to show that less than one hundred values were required in order to code a normalized face image accurately. A few years later Turk and Pentland [7] expanded the approach by discovering how to detect faces within images. It led to the first automatic face recognition. It was a real milestone in proving the feasibility of automatic facial recognition.

In the meantime, another branch of study was being developed - the artificial neural networks. This, inspired by the human brain structure system turned out to be a breakthrough solution to many engineering problems.

Nowadays, the deep neural network approach is one of the most actively investigated methods regarding the facial recognition problem. Deep Learning is at the cutting edge of what machines can do. It seems to provide the best results among all other facial recognition algorithms.

2. Face Recognition Process

2.1. Face Detection

The process of recognizing a face in an image consists of two phases:

1. **Face detection** – detecting the pixels in the image which represent the face,
2. **Face recognition** – the actual task of recognizing the face by analyzing the part of the image identified during the face detection phase.

Numerous algorithms have been introduced and claimed to have the accurate performance to tackle face detection problems. E.g. Principle Component Analysis, Local Binary Pattern, Fisherface algorithm, Gabor Wavelet method, Viola-Jones algorithm or the Artificial Neural Networks. In this work, the Local Binary Pattern Algorithm (*LBP*) was used to extract the faces from the given image dataset. LBP is not the most efficient algorithm among the others listed above, however, it is good enough to gather the required amount of training data for the face recognition process, which is the main topic of this thesis.

The problem of face detection is almost as complex as face recognition itself. The framework for both face detection and recognition is very similar, hence, the difficulties that may be encountered are basically the same. Differences such as various facial expressions, presence or absence of structural components (e.g. beard, mustaches, and glasses), illumination variations, pose, size, rotation complicate both face detection and recognition.

2.2. Face Recognition

The face recognition phase can be applied in two different types of applications:

1. Verification - the process of affirming that a claimed identity is correct by comparing the offered claims of identity with one or more previously enrolled templates. Verification systems are generally described as a 1-to-1 matching system because the algorithm tries to match the biometric presented by the individual against a specific biometric already present in the system.
2. Identification - the system attempts to determine the identity of an individual. The application must check the biometric presented against all others already in the database. Identification systems are described as a 1-to-n matching system, where n is the total number of samples in the database.

Facial recognition can be approached by using many different kinds of machine learning algorithms. At a high level, these different algorithms can be classified into two groups based on the way they "learn" about data to make predictions: supervised and unsupervised learning.

The goal of both methods is to find a specific relationship or structure in the input data that allow us to effectively produce correct output data. The fundamental factor that differs those two algorithm groups is that the supervised algorithm is being "taught" from a given training dataset, whereas an unsupervised algorithm is deriving the inherent structure of the data without using explicitly-provided labels.

2.2.1. Unsupervised learning

Principal Component Analysis

The purpose of PCA is to reduce the large dimensionality of data space to the smaller dimensionality of feature space, that nonetheless retains most of the sample's information. The method uses linear algebra to yield the directions (principal components) that maximize the variance of the data.

Self-organizing Map

The self-organizing map is a type of artificial neural network that belongs to a category of competitive learning networks adapted using an unsupervised approach. During training, the output unit that provides the highest activation to a given input sample is declared the winner and is moved closer to the input sample, whereas the rest of the neurons remain unchanged.

In short, SOM algorithm implements a mapping from the high dimensional space to map units. The map units, so-called neurons, usually form a two-dimensional grid. Therefore a SOM method provides a dimensionality reduction.

Independent Component Analysis

Independent component analysis, as the name suggests, has a lot in common with Principal Component Analysis. Both PCA and ICA seek a set of basis vectors that the inputs data is projected against. The difference between those two algorithms is that PCA finds the set of vectors that best explains the variability of the input data under the constraint that it is orthogonal to the preceding components, whereas in ICA each vector represents an independent component of the input data.

2.2.2. Supervised learning

Nowadays, the supervised machine learning is much more more common across a wide range of industry use cases. The most popular supervised algorithms in terms of facial recognition are briefly described below.

Support Vector Machine

SVM belongs to the class of maximum margin classifiers. It performs a pattern recognition between two classes by finding a hyperplane that separates the largest possible fraction of points of the same class on the same side while maximizing the distance from either class to the hyperplane. In terms of face recognition, the PCA is first used to extract features of face images, and then discrimination functions between each pair of images are learned by the support vector machine algorithm.

Artificial Neural Network

Artificial Neural Network is an approach, based on a large collection of neural units, so-called neurons, structurized in layers. The deep learning models are inspired by the way biological neural networks in the human brain process information. In case of face recognition, we have to deal with the problem of

processing very high-dimensional inputs. To face this problem the Convolutional Neural Network (*CNN*) was introduced.

3. Test scenario

In this thesis we will focus on the verification problem and take a closer look at the Principal Component Analysis, the Support Vector Machine, and Artificial Neural Networks.

The tests were performed on two different databases: – Chicago Face Database - good quality pictures captured in a controlled environment

Figure 3.1: Examples of one person pictures in the Chicago Face Database

The database consists of 100 individuals, with 4 samples each. Tests were performed using 3 pictures as a training dataset and 1 picture as a testing sample. Due to a small amount of data cross-validation was applied. All test were performed on 20 and 40 individuals from the database.

Labeled Faces in Wild database - poor quality pictures captured in an uncontrolled environment, with various lightning conditions, pose, facial expression etc.

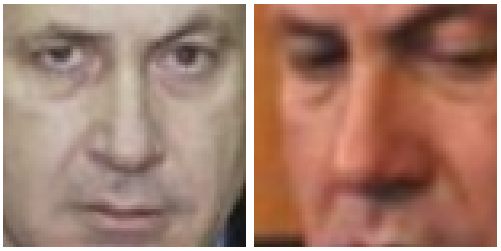


Figure 3.2: Exemplary samples from Labeled Faces in Wild database

The database consists of 5746 individuals. The number of pictures for each person is different. 20 individuals with the greatest number of samples were chosen to test the algorithms.

Table 3.1: List of individuals taken for testing the algorithm

Name	Number of pictures
George W. Bush	100
Colin Powell	100
Tony Blair	100
Donald Rumsfeld	100
Gerhard Schroeder	100
Ariel Sharon	77
Hugo Chavez	71
Junichiro Koizumi	60
Jean Chretien	55
John Ashcroft	53
Serena Williams	52
Jacques Chirac	52
Vladimir Putin	49
Luiz Inacio Lula da Silva	48
Gloria Macapagal Arroyo	44
Arnold Schwarzenegger	42
Jennifer Capriati	42
Laura Bush	41
Lleyton Hewitt	41
Hans Blix	39

Data was randomly separated into two sets:

- 80% training set,
- 20% testing set.

Due to early stopping approach that was used in CNN algorithm, in this case, the data was randomly divided into three sets:

- 75% training set,
- 5% validation set,
- 20% testing set.

The exact testing scenario for CNN algorithm was explained in chapter 7.

4. Principal Component Analysis

Principal Component Analysis is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. In fact, this approach reduces the number of feature vectors. The main advantages of the PCA are its low sensitivity to noises, the reduction of the requirements of the memory, and the increase in efficiency due to the operation in the space of smaller dimensions. In the face recognition problem, PCA is based on extracting the characteristic features of the face and representing it as a linear combination of so-called eigenfaces obtained from the feature extraction process. PCA for face recognition is usually called eigenfaces method.

4.1. Algorithm Background

The input data to the algorithm is a square, N by N pixels image that can be expressed as an N^2 dimensional vector.

$$X = (x_1, x_2, ..., x_{N^2}) \tag{4.1}$$

where the rows of pixels in the image are placed one after the other. The values in the vector are the intensity values of the image - a single greyscale value.

Say we have M images in our training dataset containing NxN pixels images. For each image, we create an image vector as described above. Later on, we built our database matrix representation.

$$ImagesMatrix = \begin{pmatrix} ImageVector_1 \\ ImageVector_2 \\ \vdots \\ Image1vector_M \end{pmatrix} \tag{4.2}$$

In the next step, the average of the image set is calculated as:

$$averageFace = \frac{1}{M} \sum_{i=1}^{M} ImageVector_i \tag{4.3}$$

The graphical representation of $avarageFace$ is presented in *Figure 4.1.*

Figure 4.1: Avarage Face graphical representation

$averageFace$ is calculated and subtracted from each face in the training set, which results in the set of normalized training images - matrix A. The normalization is computed as follows:

$$\phi_i = ImageVector_i - averageFace \tag{4.4}$$

$$A = [\phi_1, \phi 2 \cdots \phi_M] \tag{4.5}$$

In the next step the covariance matrix C should be computed, from which the eigenvectors need to be derived. The basic formula for the covariance matrix is given by:

$$C = AA^T \tag{4.6}$$

Since we know that the dimension of matrix A is N^2xM, we can derive the dimension of such a covariance matrix C:

$$C \sim N^2xN^2 \tag{4.7}$$

It turns out that using a matrix as high-dimensional as matrix C would be highly inefficient for further calculations. As a result, we would get N^2 eigenvectors. The solution to this problem is dimensionality reduction.

The matrix C' is computed as follows

$$C' = A^T A \tag{4.8}$$

$$C' \sim MxM \tag{4.9}$$

The dimension of the matrix C' is significantly smaller than the dimension of the matrix C. In result of such an operation, we get M eigenvectors (v_i) with M elements each.

$$Cv_i = \lambda_i v_i \tag{4.10}$$

where λ_i are eigenvalues and v_i are eigenvectors.

The next step is to choose the K eigenvectors such as $K < M$. The eigenvectors that correspond to zero-eigenvalues are discarded.

The set of extracted significant eigenvectors has to be mapped back into the original dimensions, which provides the set of eigenfaces.

Using linear algebra principles, we can perform the mapping as described in formula 4.11

$$u_i = Av_i \tag{4.11}$$

$$||u_i|| = 1 \tag{4.12}$$

The eigenfaces u_i may be considered as a set of features which characterize the global variation among images. The graphical representation of eigenfaces can be seen below:

Figure 4.2: Image representation of eigenfaces

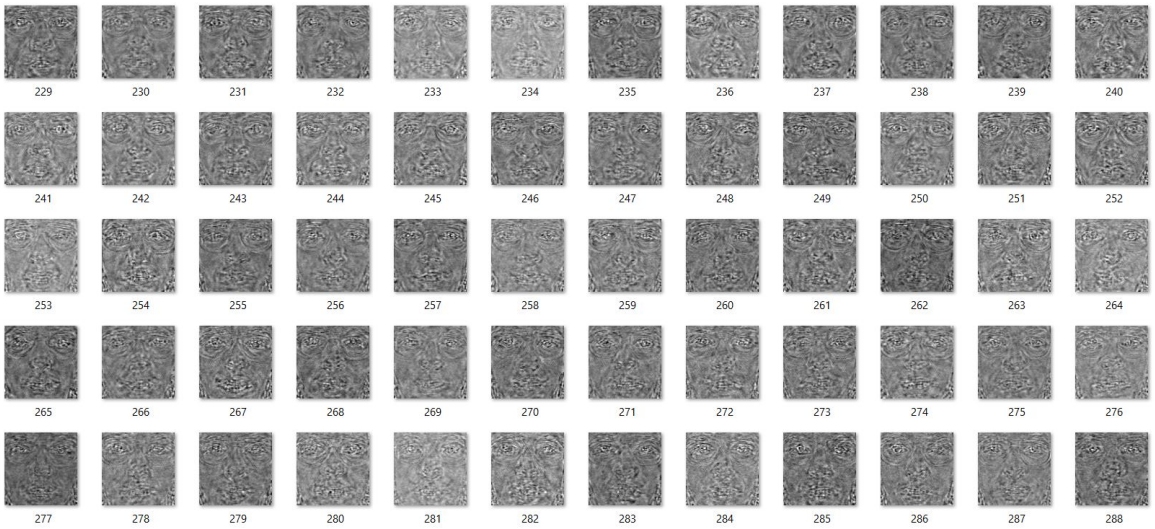


Figure 4.3: Less significant eigenfaces

In figure 4.2, we can observe 50 of the most significant eigenvectors - the ones with the highest eigenvalues, whereas figure 4.3 presents the least significant eigenfaces.

Figure 4.7 presents the image representation of three chosen eigenfaces from CFD database.

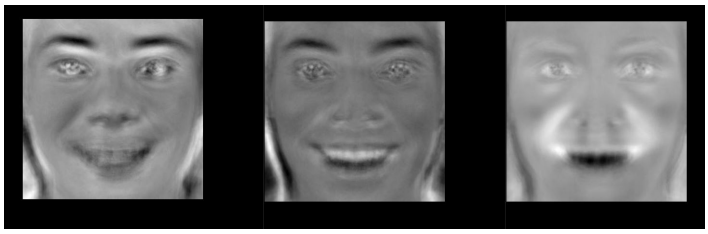


Figure 4.4: CFD eigenfaces

The next step is to represent each normalized face in a training set as a linear combination of previously extracted eigenfaces. In this way, we can represent each face in a training set as a one-dimensional vector with elements corresponding to each eigenface.

$$\Omega_i = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{pmatrix} \tag{4.13}$$

The last step in the PCA algorithm is to project the unknown face x into a space of eigenfaces as:

$$\Omega = U^T(x - averageFace) \tag{4.14}$$

where U is the set of significant eigenvectors.

One simple way to determine to which class x belongs is minimizing the Euclidean distance given by:

$$\epsilon_k = ||\Omega - \Omega_k|| \tag{4.15}$$

where Ω_k is the weight vector representing the k_{th} face class. The face x is considered as belonging to class k if the minimum ϵ_k is smaller than some threshold t. Otherwise, face x is classified as unknown.

Since to apply MLP algorithm we need to perform the PCA for dimensionality reduction, the algorithm results are presented in chapter 6.

5. Artificial Neural Networks

One of the most powerful methods for solving the face recognition problem are Artificial Neural Networks. A good definition of ANN is given by Haykin [1], describing ANN as a parallel combination of simple processing units which can acquire knowledge from the environment through a learning process and store the knowledge in its connections. The idea of Artificial Neural Network was inspired by biological neural networks, in particular, the brain.

5.1. Biological background

A neuron, in a biological sense, is a cell that carries and processes information - the electric signal. The typical neuron is composed of a cell body (perikaryon) and two types of branches: axons and dendrites. The cell body consists of plasma and a nucleus that holds information about hereditary traits. Each neuron receives information from other neurons through the dendrites and transmits the processed information via axons. The connection between an axon and a dendrite is called synapse.

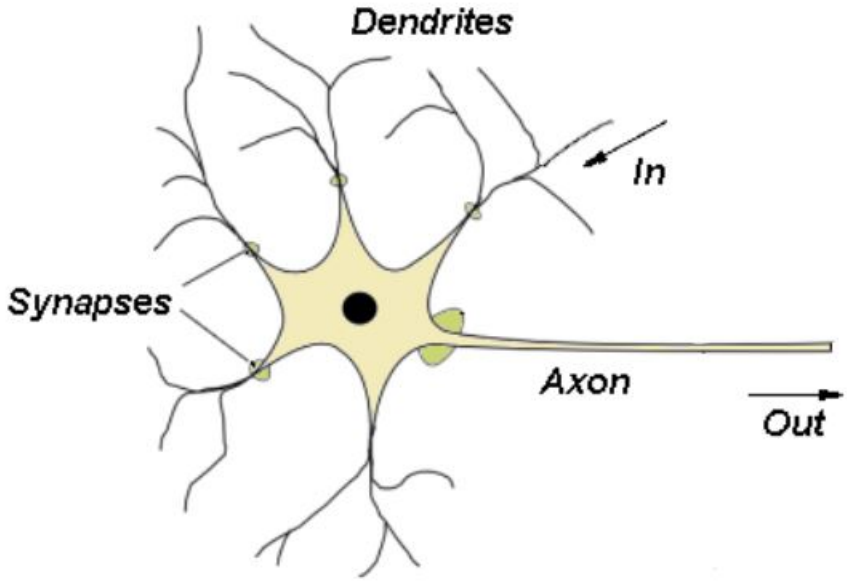


Figure 5.1: Biological Neuron Schema

The synapses' effectiveness can be adjusted by the signals passing through it so that it can learn from the activities in which it participates.

A typical person is capable of making complex perceptual decisions such as face recognition within a few hundred milliseconds. These decisions are made by a network of neurons whose operational speed

is only a few milliseconds. This implies that the computations cannot take more than about 100 serial steps. It is known as the hundred-step rule: the brain runs parallel programs that are at most 100 steps long for such a task.

It can be deduced that the amount of information sent from one neuron to another is very small. Hence, the critical information is not transmitted directly but captured and distributed in the neuron interconnections.

That is also one of the features of Artificial Neural Networks. The system, inspired by the human brain, is hoped to possess some of the most desired brain's characteristics such as:

- massive parallelism,
- distributed representation and computation,
- learning ability,
- adaptability,
- fault tolerance.

5.2. Artificial Neural Network architecture

5.2.1. Forward pass

The smallest unit of computation in a neural network is the neuron, also called a node or unit. It receives an input from the other neurons or (in case of input neurons) from the external source. Each input has an associated weight, which is assigned proportionally to its relative importance to other inputs. The neuron produces the output by applying the so-called activation function to the weighted sum of its inputs.

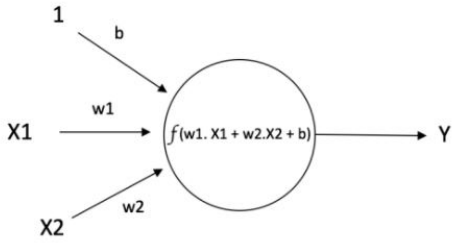


Figure 5.2: Single Neuron

The activation function is applied in order to introduce non-linearity into the neurons' output. The choice of activation function is wide, and it remains as an active area of research. The ones most commonly used are presented below:

- Sigmoid: squashes the input into the range between 0 and 1,

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{5.1}$$

- Hyperbolic tangent: squashes the input into the range between -1 and 1,

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.2}$$

- Rectified Linear Unit (ReLU): takes a real-valued input and thresholds it at zero.

$$f(x) = max(0, x) \tag{5.3}$$

According to the recent publications the ReLU function is believed to perform better than the other activation functions. Its simplicity reduces the computational complexity - it does not involve expensive operations such as exponentials. Another big advantage of ReLU is non-saturation of its gradient, which greatly accelerates the convergence of stochastic gradient descent compared to the sigmoid / tanh functions [2].

The general equation describing the neural network forward pass is given by:

$$\vec{o} = f(\vec{w} \cdot \vec{x} + b) \tag{5.4}$$

where o is the output vector, f is an activation function, $\vec{w}$ is a weights vector, $\vec{x}$ is the vector of layer input, b is a bias.

5.2.2. Backpropagation

One of the main requirements for training neural networks is a big amount of input data. All learning algorithms use data in their training processes, but ANN requires more than most.

Backpropagation, also called gradient descent, is a common method for training a neural network. It is used for training the Multilayer Perception Network as well as the Convolutional Neural Network.

The basic idea behind gradient descent method is to adjust the parameters of the neural networks in the way that the computed error between predicted and expected values will be minimized.

The choice of error functions (cost functions) is wide. One of the simplest approaches is to calculate the Euclidean distance between the predicted o and expected o' values:

$$E_1(o, o') = \frac{||o - o'||^2}{2} \tag{5.5}$$

The error can also be calculated using Mean Squared Error:

$$E_2(o, o') = \frac{\sum_{k=1}^{K}(o_i - o'_i)^2}{K} \tag{5.6}$$

where K is the size of the output vector.

When using a neural network in order to perform classification and prediction, it is usually better to use cross-entropy error than classification error or mean squared error, due to the better convergence of the gradient.

The cross-entropy error function for multi-class output is defined as:

$$E_3(o, o') = -\sum_{i=1}^{K} o'_i \ln(o_i) \tag{5.7}$$

Since the goal of backpropagation is to minimize the error E, which depends on the network weights, we have to deal with all weights in the network one at a time. To perform the backpropagation, the output of each unit from the forward pass must be stored. The gradients with respect to each parameter w_{ij} are being calculated.

1. As a first step we calculate so-called errors $\delta_i^{(o)}$ for the output units:

$$\delta_i^{(o)} = \frac{\partial E}{\partial o_i} f'(z_i) \tag{5.8}$$

where z_i is the input to the output layer.

2. We determine $\delta_i^{(l)}$ for all hidden layers in the network:

$$\delta_i^{(l)} = f'(z_i^{(l)}) \sum_{k=1}^{m^{(l+1)}} w_{i,k}^{(l+1)} \delta_k^{(l+1)} \tag{5.9}$$

where $m^{(l)}$ is the number of units in layer l.

3. We calculate the derivative:

$$\frac{\partial E}{\partial w_{j,i}^{(l)}} = \delta_j^{(l)} y_i^{(l-1)} \tag{5.10}$$

4. Once all partial derivatives have been computed, the gradient descent is performed by subtracting the increment from each weight:

$$\Delta w_{ij} = \lambda z_j^{(l-1)} \delta_j^{(l)} \tag{5.11}$$

where λ is the learning rate, which determines how fast the network coefficients change.

After randomly choosing the initial weights of the network, the backpropagation algorithm is used to compute the necessary corrections. The neural network training process can be decomposed in the following steps:

1. Feed-forward pass,
2. Backpropagation of each layer,
3. Weight updates.

The algorithm is stopped when the value of the error function is sufficiently small.

5.2.3. ANN topology

The architecture of an Artificial Neural Network defines how its several neurons are arranged, or placed, in relation to each other. In general, an artificial neural network architecture can be divided into three parts:

1. Input layer - responsible for receiving normalized information, signals, features, or measurements from the user,
2. Hidden layer (or layers) - responsible for further information processing such as extracting patterns associated with the process,
3. Output layer - responsible for producing and presenting the final network outputs, which results from the previous information processing.

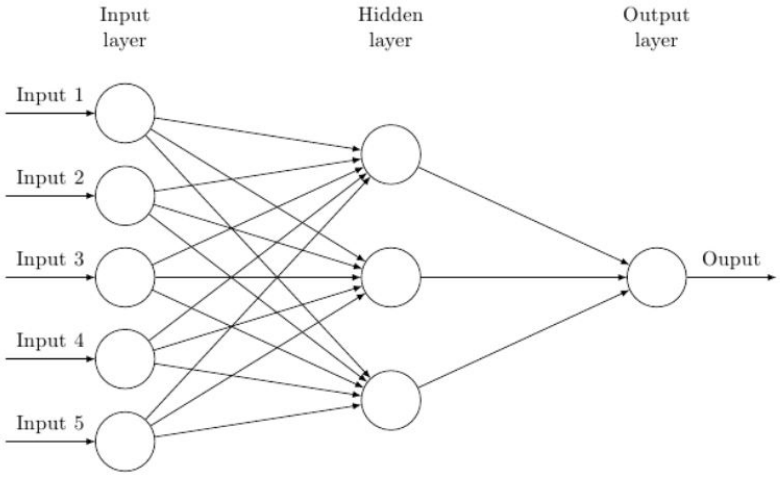


Figure 5.3: Basic Artificial Neural Network Schema

The presented ANN structure performs a feed-forward process. The connections only project forward, which means, that neurons in a layer send the signal only to the subsequent layer. Furthermore, there are no connections between neurons in the same layer or between non-adjacent layers.

There are plenty various configurations of feed-forward ANNs, which can differ in the number of neurons in each layer as well as in the number of layers.

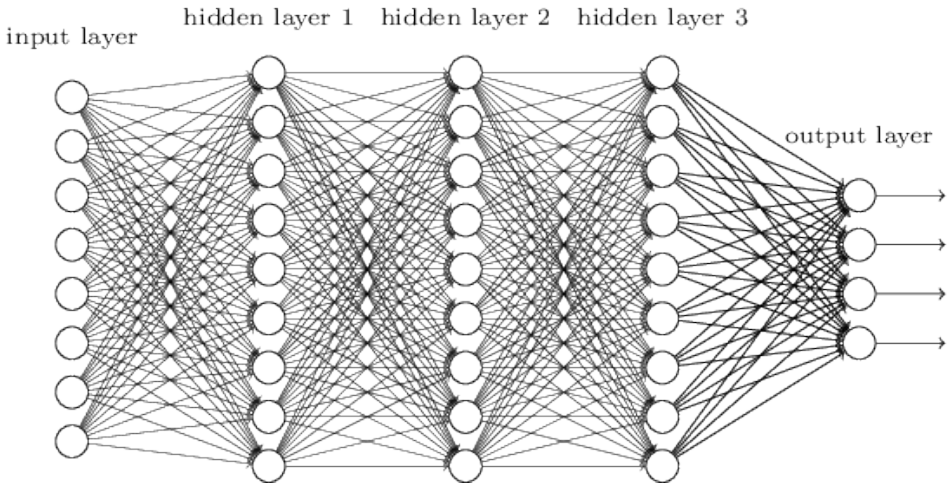


Figure 5.4: Example of Deep Neural Network architecture

A neural network with more than one hidden layer is a so-called Deep Neural Network (Figure 4.4). This type of ANN is suitable for processing very high-dimensional data, hence, it seems to be a good choice for the facial recognition system design.

6. Multi-Layer Perception

The multi layer perception is the most known and most frequently used type of neural network. It consists of at least three layers. The difference between Multi and Single Layer Perception is, that the first one can learn non-linear functions, whereas the second one is only suitable for solving linear problems.

Multilayer perceptrions are often applied to supervised learning problems: they are being trained on a set of input-output pairs and learn to model the dependencies between the given inputs and outputs. MLP performs well with relatively low dimensional inputs.

6.1. Implementation and test result

Due to the big number of parameters, it is rather a poor solution for working with the images as its inputs. To proceed with the MLP algorithm we need relatively low dimensional data. The idea to implement Face Recognition system using MLP is to connect the Multilayer Perception model with the Principal Component Analysis algorithm.

The first step in this approach is to perform PCA on every image in a training dataset. PCA produces a weight vector (3.13) for an image so we can represent each image in a low dimensional space. The dimension depends on the number of eigenvectors chosen for the Principal Component Analysis algorithm.

$$X = \begin{bmatrix} w_{11} & w_{21} & \cdots & \cdots & w_{m1} \\ w_{12} & w_{22} & \cdots & \cdots & w_{m2} \\ w_{13} & w_{23} & \cdots & \cdots & w_{m3} \\ w_{14} & w_{24} & \cdots & \cdots & w_{m4} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ w_{1k} & w_{2k} & \cdots & \cdots & w_{mk} \end{bmatrix} \tag{6.1}$$

where k is a number of eigenvectors and m is the amount of data in a training dataset.

Since the MLP is a supervised algorithm, the data labels are required to train the model. Each label was represented as a k-dimensional one-hot vector.

$$Y = \begin{bmatrix} 0 & 1 & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & 1 \\ 1 & 0 & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \cdots & 0 \end{bmatrix} \tag{6.2}$$

The goal is to train the model in a way that given the matrix X, MLP will produce close approximation of the matrix Y.

For training the model backpropagation algorithm was used.

```
def back_prop(epochs, X, Y, Wh, Wp, activation, l_rate):
    for i in range(epochs):
        H = activation(np.dot(X, Wh))
        Z = activation(np.dot(H, Wp))
        E = error(Y, P)
        dP = E * activation(Z, deriv=True)
        dH = dP.dot(Wp.T) * activation(H, deriv = True)
        Wz += l_rate * H.T.dot(dP)
        Wh += l_rate * X.T.dot(dH)
        if i % 10000 == 0:
            per = check_performance(Z)
            if per > 0.98:
                break
```

The neural network performance varies depending on the change of various parameters such as:

- amount of input data,
- number of neurons in hidden layers,
- initial weights,
- activation function,
- learning parameter.

The tests were performed on CFD and LFW databases described in chapter 3.

6.2. Tests on pictures captured in a controlled environment

The tests were performed using a MLP topology presented in figure 6.3.

6.2.1. 20 individuals

The first step is to set a number of eigenvectors that the image will be represented with. To do that we use the previously implemented PCA algorithm.

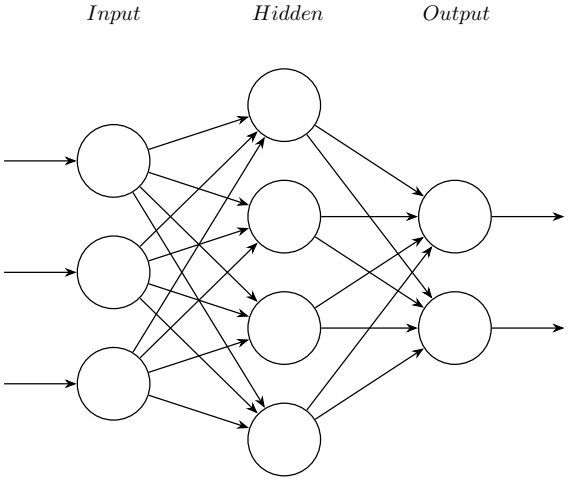


Figure 6.1: Neural Network with one hidden layer

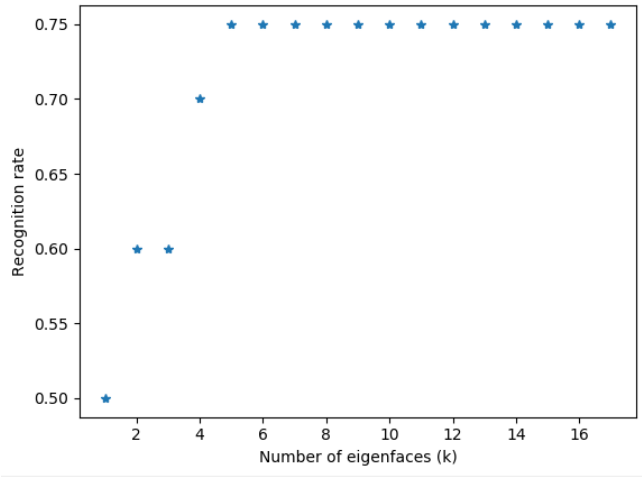


Figure 6.2: Recognition rate with a varying number of eigenvectors for 20 individuals

From these results, we can conclude that $k = 6$ is the sufficient number of eigenvectors to properly represent each sample image.

The test was performed on a dataset with 20 individuals. The MLP algorithm was run with following parameters:

- number of hidden layers = 1,
- number of neurons in hidden layers = 10,
- initial weights chosen randomly in a range [0:1],
- sigmoid activation function,
- learning parameter = 1,
- number of eigenvectors $k = 6$.

In order to examine the accuracy of MLP algorithm, we check if the index of the obtained outputs' maximum value is equal to the index of the maximum value of the desired output. The accuracy is a value representing the percentage of pictures that fulfills this requirement.

As the first step the influence of a learning rate parameter was examined.

Learning rate = 1

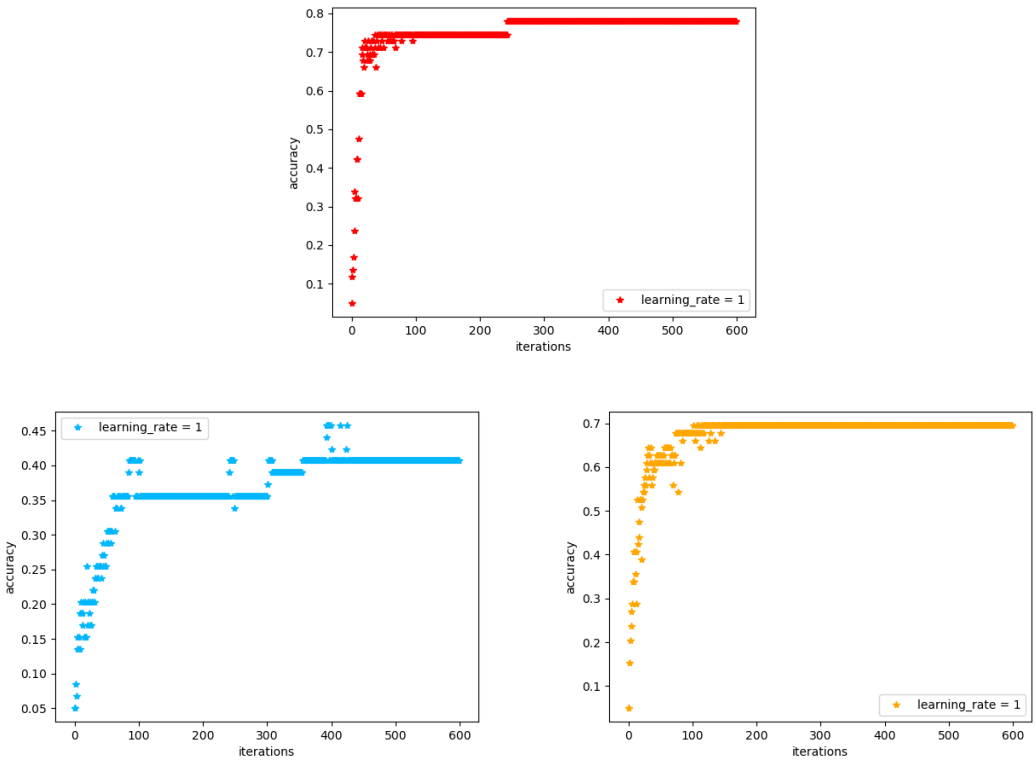


Figure 6.3: Learning rate = 1, initial weights chosen randomly

The neural network with the learning rate = 1 is very unstable and is strongly dependent on initial weights.

The tests were performed for different learning rate values:

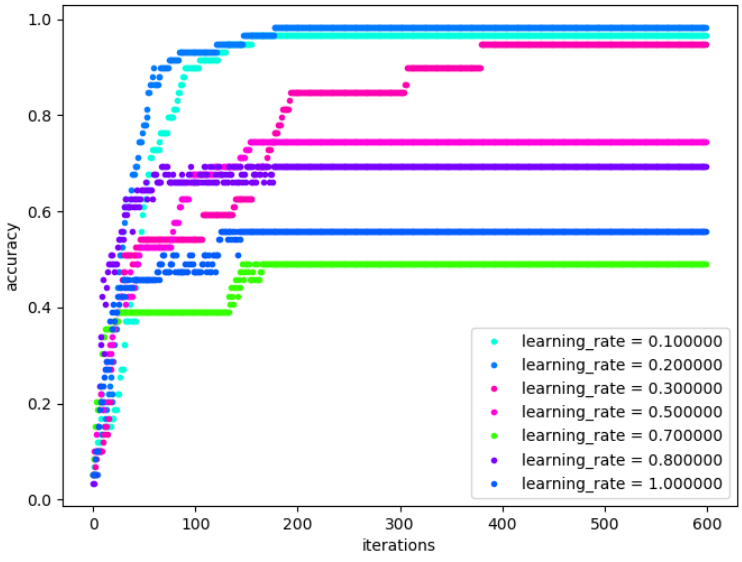


Figure 6.4: Algorithm accuracy achieved for different learning rates

The learning rate is one of the most important hyper-parameters to tune for training neural networks. If the learning rate is low, then training is more reliable, but optimization will take more time because the weights' changes are tiny.

For each experiment approximately 400 iterations were enough to sufficiently train the network.

The testing phase was made using the model that was trained with following parameters:

- number of hidden layers = 1,
- number of neurons in hidden layers = 10,
- initial weights chosen randomly in a range [0:1],
- sigmoid activation function,
- learning parameter = 0.2,
- number of eigenvectors $k = 6$.

Obtained results are good. 18 out of 20 people were recognized correctly - the recognition rate reached 90%, which is 15 percentage points better than the result of the PCA algorithm, that reached 75% of accuracy (Figure 6.4).

6.2.2. 40 individuals

Following the same procedure as in section 6.2.1. we can derive the optimal (for PCA algorithm) k value.

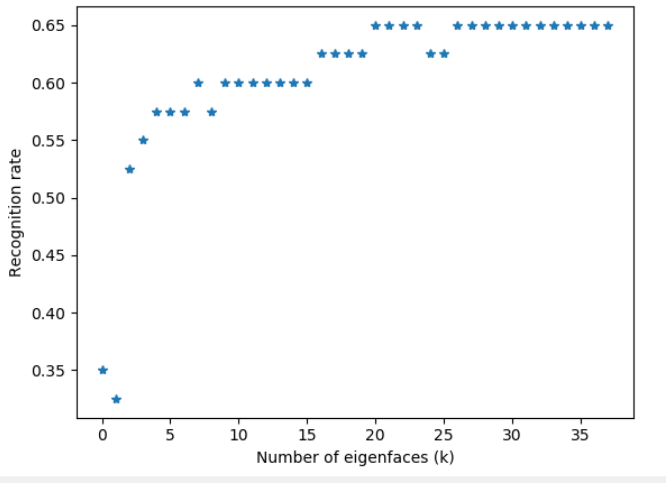


Figure 6.5: Recognition rate with varying number of eigenvectors for 40 individuals

When dealing with a bigger database, we have to find a balance between a sufficient number of eigenfaces to represent an image and an amount of data that can be applied as the input to a multilayer perception network, so that we will obtain acceptable training result in a reasonable time. The bigger input dimension, the harder it is to train the network.

In this case, input dimension = 27 is still low enough to be processed with the MLP network, hence the parameter k was set to 27.

The test was performed with following parameters:

- dimension of input data = 27,

- number of hidden layers = 1,
- number of neurons in hidden layers = 27,
- initial weights chosen randomly in a range [0:1],
- sigmoid activation function,
- learning rate = 0.2.

The result of training algorithm.

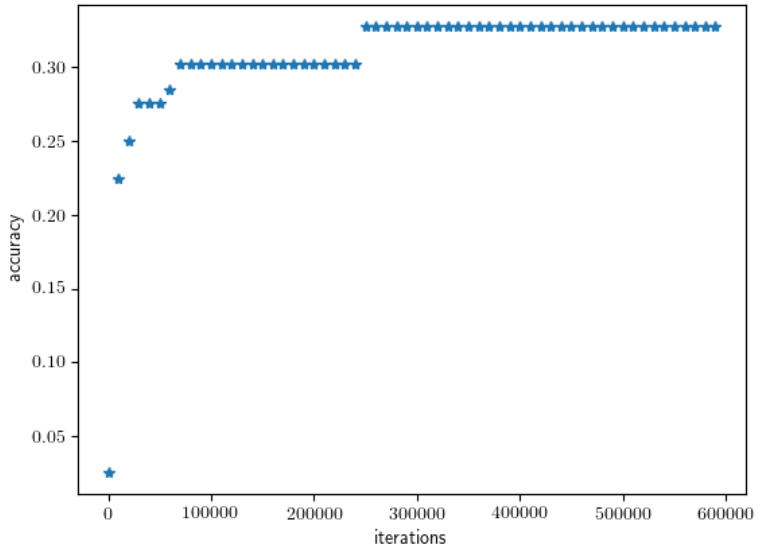


Figure 6.6: Number of eigenvectors = 27, number of hidden neurons = 27.

The obtained training accuracy of the algorithm in this configuration reached 35%. The results are not satisfying.

The next interesting parameter in terms of neural network performance is a number of neurons in a hidden layer. This topic is still an active area of research, and the major answers are driven by tests and experience. In the previous experiment, the number of hidden neurons was equal to the number of input neurons and it does not seem to be a good solution. The recognition rate in the network trained as such reached 22,5%, which is very poor. The next step is to check if increasing or decreasing the number of neurons in a hidden layer will help to improve the network performance.

Using too little neurons in the hidden layers will result in underfitting, which means that the network is not capable of adequately detecting the signals in a complicated dataset.

On the other hand, too many neurons in the hidden layer can result in several problems. First of all, overfitting which occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers. Second of all, there is another problem that can be encountered even if the training data is sufficient. A large number of neurons in the hidden layer can increase the time it takes to train the network. A trade off must be reached between too many and too few neurons in the hidden layer.

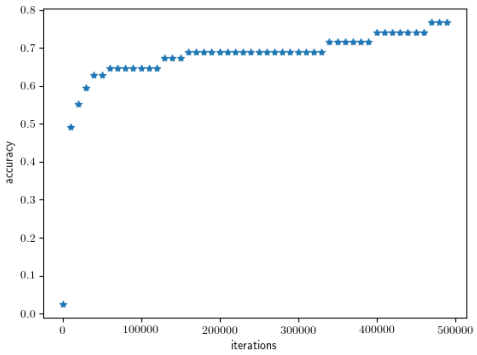
There are several rule-of-thumb methods for determining a sufficient number of neurons in the hidden layers, such as the following:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer,
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer,
- The number of hidden neurons should be less than twice the size of the input layer.

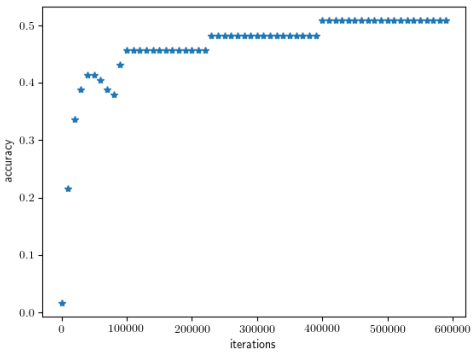
In our case, we have 27 input neurons and 40 output neurons. According to the listed rules we have 3 possibilities:

1. the number of hidden neurons should be in the range between 27 and 40,
2. the number of hidden neurons should be equal to $0.6 * 27 + 40 \sim 56$,
3. the number of hidden neurons should be less than $\frac{27}{2} = 13.5$.

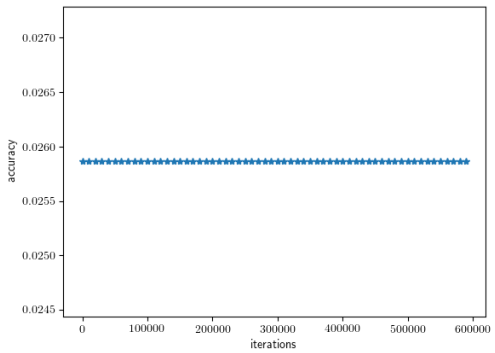
Each of the possibilities was examined. The results not only differ in the accuracy of the algorithm but also in the execution time.



(a) 10 hidden layers



(b) 33 hidden layers



(c) 56 hidden layers

Figure 6.7: Training accuracy for the different number of hidden neurons

Number of hidden neurons	10	33	56
Execution time[sec]	114	173	216
Recognition Rate	70%	43.5%	2.5%
Training accuracy	76.5%	50%	2.5%

The best results were achieved with 10 hidden layers. Comparing these results with the result of the PCA algorithm on the same data, it seems that the MLP algorithm performs better than the PCA algorithm. The PCA algorithm achieved 65% of recognition rate (Figure 6.7) whereas MLP algorithm reached 70%.

6.3. Tests on pictures captured in an uncontrolled environment

In the previous section, the algorithm was tested with Chicago Face Database, where all the individuals were taken in a controlled environment. The samples of each individual differs only in terms of facial expression. It is much more challenging to build a system that is not sensitive to changing the light conditions, the picture quality, face pose etc.

To test the algorithms' robustness to these factors, the Labeled Faces in Wild database was used. The quality of images is much worse than in the previous examples, so the tests result are expected to be worse.

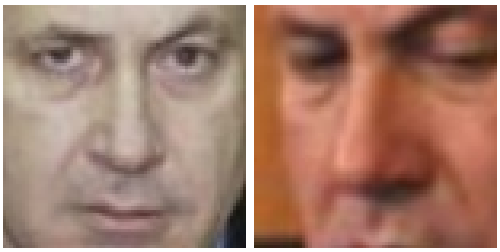


Figure 6.8: Exemplary samples from the LFW database

6.3.1. One hidden layer

The tests were performed in the same topology as presented in figure 6.3

As in the previous examples PCA was performed as a first step. The results (figure 6.13) are very poor as expected.

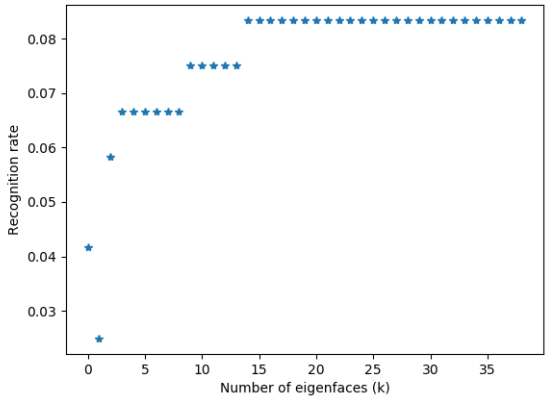


Figure 6.9: Recognition rate with varying number of eigenvectors for 20 individuals

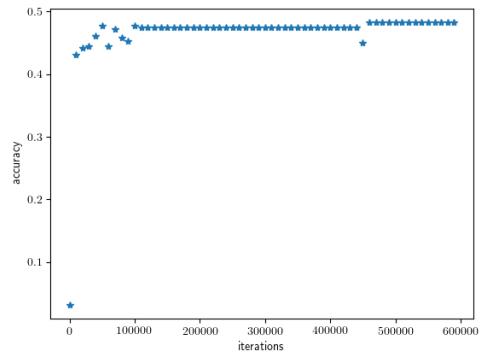
The question is how much the results can be improved with applying MLP network.

The model was trained with following parameters:

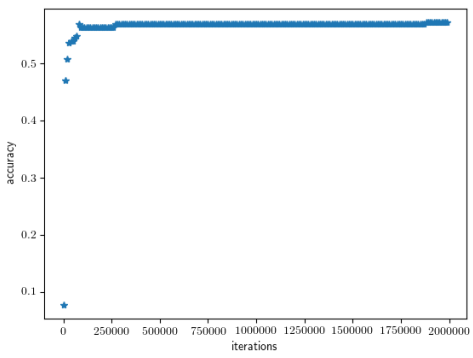
- number of hidden layers = 1,
- initial weights chosen randomly in a range [0:1],
- sigmoid activation function,
- learning parameter = 0.2.

and various dimensions of input data (number of eigenfaces) and different number of neurons in a hidden layer.

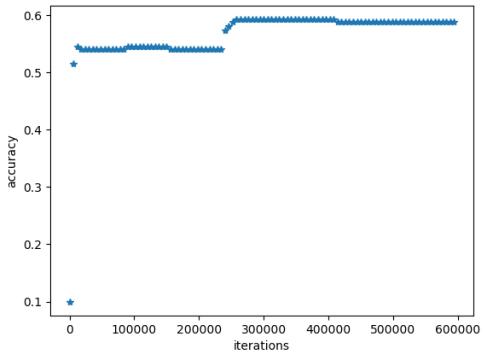
The training results are presented in figure 6.14:



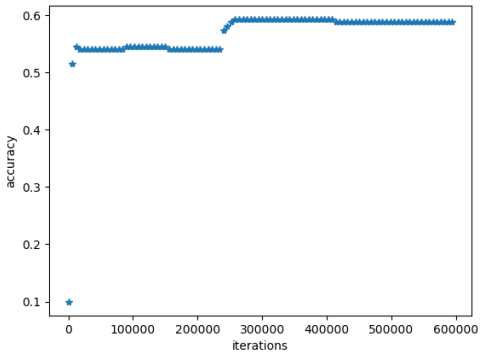
(a) 10 hidden neurons, k = 15



(b) 20 hidden neurons, k = 15



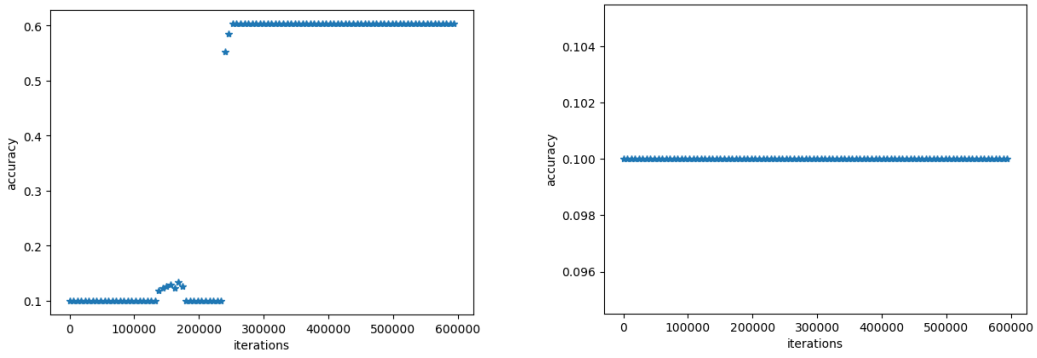
(c) 10 hidden neurons, k = 10



(d) 20 hidden neurons, k = 10

Figure 6.10: Training accuracy for different number of hidden neurons and eigenfaces

During the experiment, it was noticeable that the training accuracy grows with increasing the number of hidden neurons. However, the time to train the network also grows and sometimes it took too long to see the performance:



(a) 25 hidden neurons, k = 15 (b) 30 hidden neurons, k = 15

Figure 6.11: Training accuracy for different number of hidden neurons

After several experiments, the best result was achieved with the model trained with 13 eigenfaces and 20 hidden neurons.

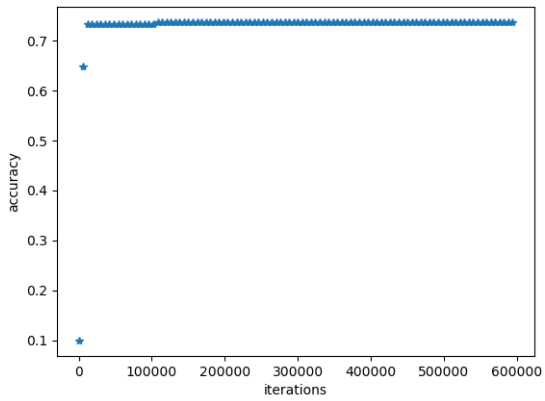


Figure 6.12: 20 hidden neurons, k = 13

The obtained recognition rate was 66.6%, which is much better than PCA result but still not acceptable for facial recognition system. However, taking into account the quality and variety of images used for these tests, the result is better than expected.

6.3.2. Two hidden layers

Further experiments were performed on network topology presented in figure 6.17. The goal is to check if an additional hidden layer improves the network performance.

The first tests were run with following parameters:

- number of hidden layers = 2,
- initial weights are chosen randomly in a range [0:1],
- sigmoid activation function,
- learning parameter = 0.2,
- number of eigenfaces = 15.

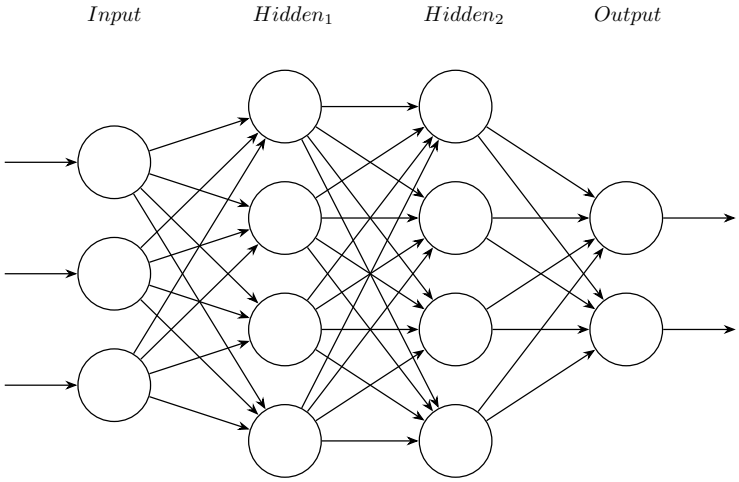


Figure 6.13: Neural Network with two hidden layers

Results are presented in figure 6.18.

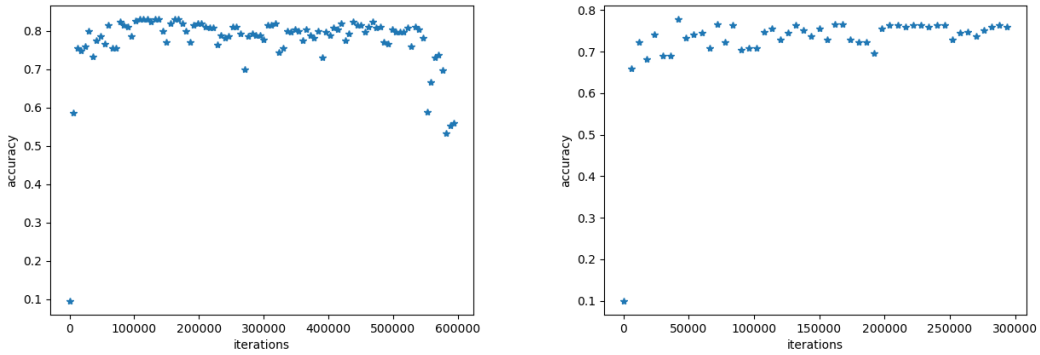


Figure 6.14: 10 hidden neurons in each hidden layers, learning rate = 0.2

The system seemed to be very unstable. The learning rate was reduced to 0.05 with the following results:

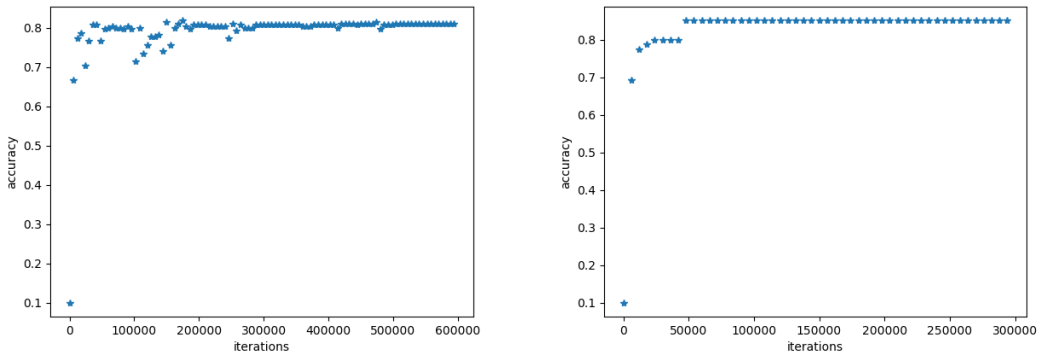


Figure 6.15: 10 hidden neurons in each hidden layers, learning rate = 0.05

Better results were obtained with a slightly bigger number of neurons in the second hidden layer.

The network with following parameters gave the best training results:

- number of hidden layers = 2,
- 10 neurons in first hidden layer,
- 15 neurons in second hidden layer,
- initial weights are chosen randomly in a range [0:1],
- sigmoid activation function,
- learning parameter = 0.05,
- number of eigenfaces = 15.

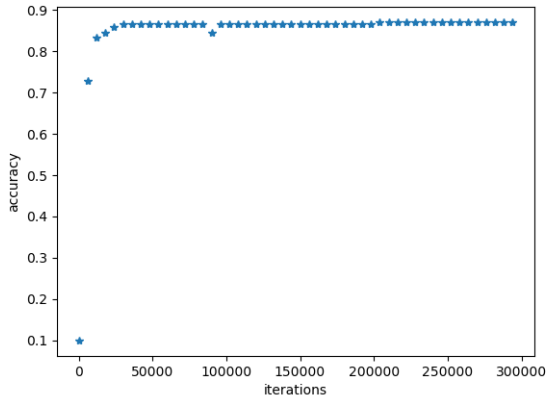


Figure 6.16: 10 neurons in the first hidden layer, 15 in the second hidden layer

The obtained recognition rate is 73.3%.

The presented results are varying significantly depending on the values of initial weights. Since the weights were chosen randomly, it was a matter of trial and error to find an appropriate network configuration and train the network properly. It is possible that this score still can be improved.

6.4. Summary

A tabular comparison of PCA and MLP algorithm is presented in Table 6.1.

Table 6.1: Tabular comparison of PCA and MLP algorithms

Algorithm	CFD 20	CFD 40	LFW
PCA	75%	65%	8%
MLP	90%	70%	73.3%

It can be clearly noticed that the performance of the MLP algorithm is much better than the PCA, especially for the images that were captured in an uncontrolled environment.

7. Convolutional Neural Networks

Convolutional Neural Network is a class of deep, feed-forward neural networks, that can be successfully applied to systems with high dimensional input such as images.

It is a pattern recognition mechanisms that is inspired by the way in which mammals visually perceive the world around them, using a layered architecture of neurons in the brain.

Convolutional Neural Networks leverage three ideas:

1. local connectivity,
2. parameter sharing,
3. pooling hidden units.

CNNs exploit spatially-local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. In other words, each unit in the hidden convolutional layer is not connected with all units in a previous layer, instead it is connected to a subset of units from a previous layer. This approach helps to solve the problem of having an unmanageable number of parameters.

The primary purpose of convolution in case of a CNNs is to extract features from the input image. In convolutional layer, a set of filters (typically square matrices with random values) is generated.

Each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature (activation) map. Activation maps indicate 'activated' regions, i.e. regions where features specific to the filter have been detected in the input.

Computation of those feature maps corresponds to computation of discrete convolution of a i_{th} channel of input and filter matrix $W \in \Re^{2h_1+1 \times 2h_2+1}$.

The convolution operation is given by

$$(X * W)_{r,s} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} W_{u,v} X_{r+u,s+v} \tag{7.1}$$

$$W = \begin{bmatrix} w_{-h_1,-h_2} & \cdots & w_{-h_1,h_2} \\ \vdots & w_{0,0} & \vdots \\ w_{h_1,-h_2} & \cdots & w_{h_1,h_2} \end{bmatrix} \tag{7.2}$$

The behavior of this operation towards the borders of the image is handled by 0-padding if needed.

The i_th feature map in layer l is computed as

$$z_i^{(l)} = \sum_{j=1}^{m} K_{i,j}^{(l)} * Y_j^{l-1} \tag{7.3}$$

where m is the size of the l layer input.

The output of the layer l, denoted $Y_i^{(l)}$ is given by

$$Y_i^{(l)} = f\left(z_i^{(l)}\right) \tag{7·4}$$

where f is an activation function.

The values of the filter matrix change with each learning iteration over the training set, indicating that the network is learning to identify which regions are of significance for extracting features from the data.

Another important concept of CNNs is max-pooling, which is a form of non-linear down-sampling. Max-pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value. Max-pooling is useful in vision for two reasons:

- By eliminating non-maximal values, it reduces computation for upper layers,
- It provides a form of translation invariance.

Since it provides additional robustness to the position, max-pooling is a smart way of reducing the dimensionality.

After a sequence of these operations, the output is flattened and applied to the fully connected layer. At the very end of the network, the output is squashed with the softmax layer so that each element in the output vector corresponds to the probability value for each class.

$$\sigma(o_j) = \frac{e^{o_j}}{\sum_{k=1}^{K} e^{o_k}} \tag{7·5}$$

where K is the number of classes and o_j is the j_{th} value of the output vector.

The whole process is presented in figure 7.1

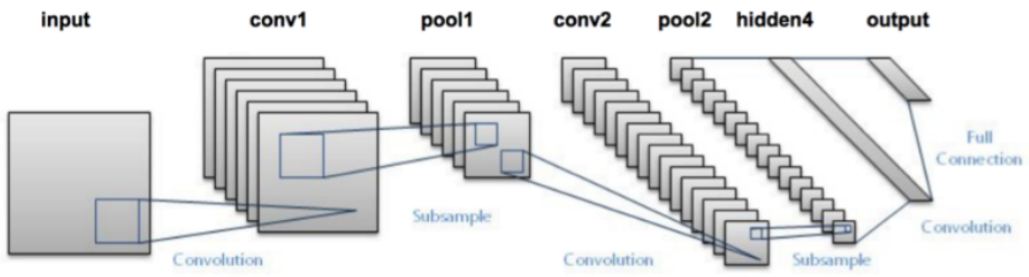


Figure 7.1: A typical convolutional neural network for face recognition

The algorithm is trained with backpropagation. The backpropagation for Convolutional Neural Networks is slightly more complicated than the one for Multilayered Perception. The general idea is exactly the same, but in CNN we have to face with backpropagation through flattening, max-pooling and convolution, which requires more effort in practice.

The first step, as in MLP, is to calculate the error between predicted and expected class probabilities. To do so, we calculate the cross-entropy error. Cross-entropy measure is widely used when the network output represents the class probabilities. Thus it is used as a loss function in neural networks which have softmax activation in the output layer.

The softmax activation of the i_th output unit is:

$$o_i = \frac{e^{s_i}}{\sum_{k=1}^{K} e^{s_k}} \tag{7.6}$$

where K is the number of classes and s_i is the i_{th} value of the last dense layer output vector.

The cross entropy function for multi-class output is given by:

$$E(o, o') = -\sum_{i=1}^{K} o'_i \ln(o_i) \tag{7.7}$$

$$E = -\sum_{i=1}^{K} o'_i \ln(o_i) \tag{7.8}$$

Computing the gradient yields

$$\frac{\partial E}{\partial o_i} = -\frac{o'_i}{o_i} \tag{7.9}$$

$$\begin{aligned} \frac{\partial o_i}{\partial s_k} &= \begin{cases} \frac{e^{s_i}}{\sum_{j=1}^{K} e^{s_j}} - \left(\frac{e^{s_i}}{\sum_{j=1}^{K} e^{s_j}}\right)^2 & i \neq k \\ -\frac{e^{s_i} e^{s_k}}{(\sum_{j=1}^{K} e^{s_j})^2} & i = k \end{cases} \\ &= \begin{cases} o_i(1 - o_i) & i \neq k \\ -o_i o_k & i = k \end{cases} \end{aligned} \tag{7.10}$$

$$\begin{aligned} \frac{\partial E}{\partial s_i} &= \sum_{k}^{K} \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial s_i} \\ &= \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial s_i} - \sum_{k \neq i} \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial s_i} \\ &= o'_i(1 - o_i) + \sum_{k \neq i} o'_k o_i \\ &= o' t_i + o_i \sum_{k} o'_k \\ &= o_i - o'_i \end{aligned} \tag{7.11}$$

The gradient for weight in the last dense layer can be computed as follows:

$$\begin{aligned} \frac{\partial E}{\partial w_{ji}} &= \sum_{i} \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial w_{ji}} \\ &= (o_i - o'_i) h_j \end{aligned} \tag{7.12}$$

where h_j is the output of the first dense layer in the network and w_{ij} is the weight connecting the units from second dense layer to the output layer.

The error is backpropagated further as described in section 5.2.2.

The next step is to perform the reversed flatten layer, so that the error can be backpropagated further to the maxpooling layer. Backpropagating through the maxpooling layer implements the idea, that the gradient from the flatten layer is passed back to only that neurons which achieved the maximum value in forward pass during the maxpooling computation. All other neurons get zero gradient.

The gradient prepared in such way is ready to be backpropagated through convolutional layers. The backpropagation for convolutional layer can be applied as described in chapter 5, however equation 5.12 changes to:

$$\Delta w_{ij} = \lambda z_j^{(l-1)} \sum_{k=1}^{m^{(l)}} \delta_k^{(l)} \tag{7.13}$$

As in every other Artificial Neural Network training process can be decomposed in the following steps:

1. Feed-forward pass,
2. Backpropagation of each layer,
3. Weight updates.

The algorithm is stopped when the value of the error function is sufficiently small.

DeepFace [Taigman et al., 2014] and FaceNet [Schroff, Kalenichenko, and Philbin, 2015] are two of the most successful applications of CNNs in the face recognition problem. These two have provided state-of-art results in recent years, with the best results being obtained by the latter.

7.1. Implementation and test results

The Convolutional Neural Network may differ in a number of layers and their connections. The one that was implemented has the following architecture:

```
def forward_pass(self, X):
    h = self.relu(self.cnn_layer(X, layer_i=0, border_mode="full"))
    h = self.relu(self.cnn_layer(h, layer_i=1, border_mode="valid"))
    h = self.maxpooling_layer(h)
    h = self.relu(self.cnn_layer(h, layer_i=3, border_mode="valid"))
    h = self.maxpooling_layer(h)
    h = self.flatten_layer(h)
    h = self.relu(self.dense_layer(h, layer_i=6))
    h = self.dense_layer(h, layer_i=7)
    h = self.softmax_layer2D(h)
    max_i = self.classify(h)
    return max_i[0]
```

The first two and the 4th layer in the network performs the convolutional operation followed by activation function RELU.

The input to the network is an image, that can be treated as a matrix with three dimensions, each corresponding to every color channel (RGB). In this case, every image has the same width and height k equal to 76 pixels.

$$R = \begin{bmatrix} r_{11} & r_{21} & \cdots & \cdots & r_{k1} \\ r_{12} & r_{22} & \cdots & \cdots & r_{k2} \\ r_{13} & r_{23} & \cdots & \cdots & r_{k3} \\ r_{14} & r_{24} & \cdots & \cdots & r_{k4} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{1k} & r_{2k} & \cdots & \cdots & r_{kk} \end{bmatrix} G = \begin{bmatrix} g_{11} & g_{21} & \cdots & \cdots & g_{k1} \\ g_{12} & g_{22} & \cdots & \cdots & g_{k2} \\ g_{13} & g_{23} & \cdots & \cdots & g_{k3} \\ g_{14} & g_{24} & \cdots & \cdots & g_{k4} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ g_{1k} & g_{2k} & \cdots & \cdots & g_{kk} \end{bmatrix} B = \begin{bmatrix} b_{11} & b_{21} & \cdots & \cdots & b_{k1} \\ b_{12} & b_{22} & \cdots & \cdots & b_{k2} \\ b_{13} & b_{23} & \cdots & \cdots & b_{k3} \\ b_{14} & b_{24} & \cdots & \cdots & b_{k4} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{1k} & b_{2k} & \cdots & \cdots & b_{kk} \end{bmatrix} \quad (7.14)$$

In convolutional layer each of those matrices was convolved with each of 32 randomly chosen filters. The convolutional layer has an additional parameter *border mode*, which can be set to either *full* or *valid*, which defines if the convolution operation involves padding around the input (valid border mode sets padding to 0). The convolution across each channel is summed up. As a result of the first two layers, we obtain a matrix with dimensions $= (1, 32, 76, 76)$.

The 3rd and the 5th layer are maxpooling layers, with the purpose to reduce the dimensionality of the data. No activation function is applied. The 3rd and 5th output matrix dimensions are accordingly: $(1, 32, 38, 38)$ and $(1, 32, 18, 18)$.

One of the last stages of a Convolutional Neural Network is a dense layer. It is just a simple MLP layer, as described in chapter 6. It requires the one-dimensional input - the output of convolutional layers has to be converted into a one-dimensional feature vector. This operation is called flattening and is performed with the flattening layer. After flattening the data, we obtain a vector with $10368 (18 * 18 * 32)$ elements. The MLP layer used in this example has one hidden layer with 6000 units. The output layer consist of a number of units equals the number of classes (number of people in the database) - in our case 20.

The last part of the network is a softmax layer, which allows us to calculate the probability value for each class.

The above construction allows classifying an image only if the filter parameters and dense layer parameters are set correctly. The choice of those coefficients is made randomly, and they are trained with backpropagation algorithm.

During the algorithm execution, it turned out that training the entire network on a reasonably sized dataset would be unfeasible on a personal laptop. Even though the dimensionality reduction (max-pooling)is applied, the total number of parameters that need to be trained is equal to approximately $1,25 * 10^9$.

According to the convolutional networks theory, we know that the goal of the lower layers is to identify lower-level features such as colors, shapes or textures, and only the top layers distinguish the specific, higher-level features of each class. Based on this idea, instead of training the whole convolutional network, we can take an existing convolutional network model and train the last layer of the network so that it can recognize the given data.

The first experiment was performed with Google Inception-v3 model. The Inception v3 model has nearly 25 million parameters and uses 5 billion multiply-add operations for classifying a single image.

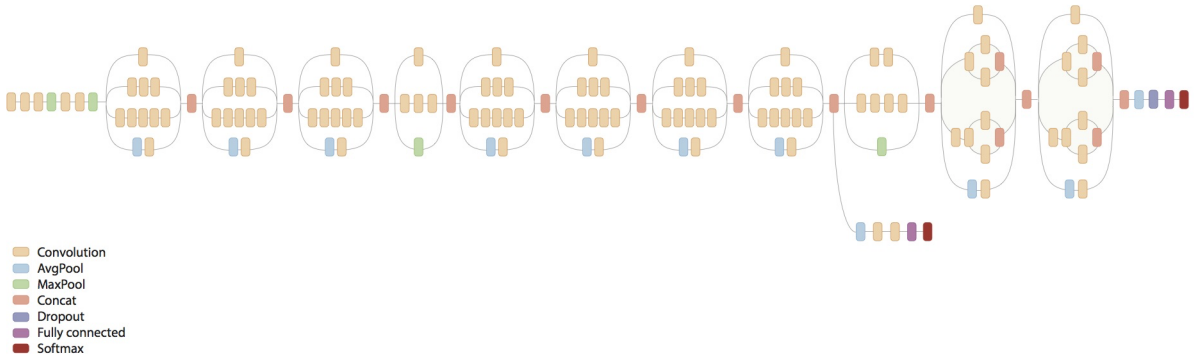


Figure 7.2: Google Inception v3 model architecture

This model structure is much more complicated than the general convolutional network architecture. Instead of applying all convolutional, max-pooling and dense layers one by one, some of the layers consist of the composition of those operations. The model was trained on the ImageNet database with 1000 image categories and turns out to outperform the human recognition abilities. The goal of this experiment is to find out how good this model can be adjusted to recognize human faces, taking into account that there were no facial images in its training dataset.

The tensorflow library provides the function that returns the outputs from a different layers of inception network. We take the last layer with 2048 neurons and connect it to the fully connected layer, creating a new output with the number of neurons corresponding to the number of individuals in the dataset. To train only a single layer, we specify the list of trainable variables in the training operation. The network trains only the parameters that connect the 2048 inception neurons with 10 output neurons of our network. At the end, the softmax activation function is applied in order to calculate probabilities for each class.

7.2. Test results

The algorithm was implemented in Python 3.4 with the usage of numpy and tensorflow libraries.

The first experiment was performed on 10 individuals with the biggest number of images from the Labeled Faces in Wild database.

Data was randomly separated into three sets:

- 75% training set,
- 5% validation set,
- 20% testing set.

The algorithm implements an early stopping approach. It requires periodically testing the network on a validation set to obtain the score on the cost function (average cross entropy). If the loss does not decrease for a specified number of iterations, training is interrupted. It prevents the network from overfitting.

The test results visualization were generated by tensorboard module (from tensorflow library).

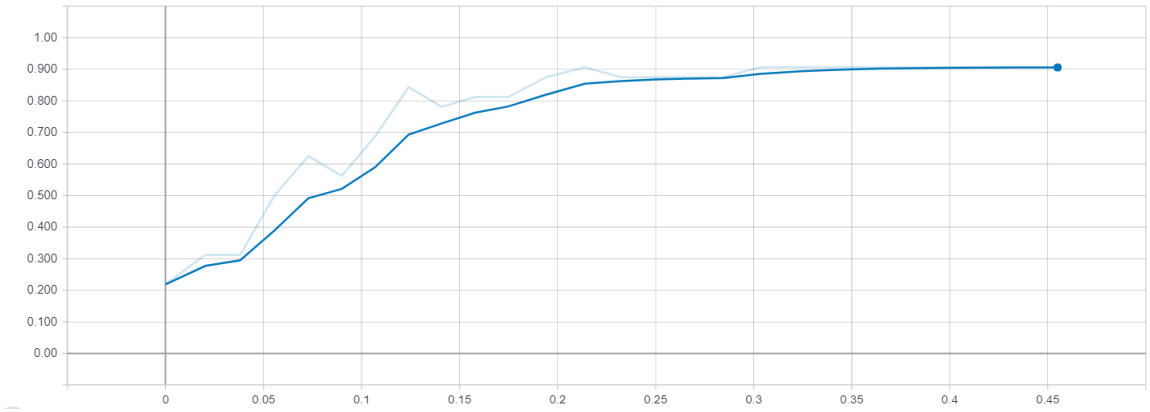


Figure 7.3: Training accuracy

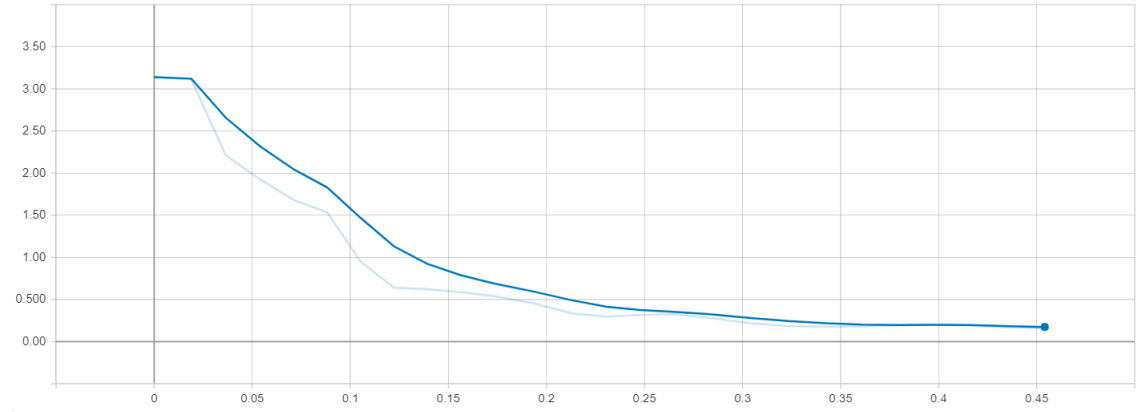


Figure 7.4: Validation loss

Taking into account that these results were obtained via training only the last layer of the network, we can consider them better than expected. The training accuracy reached over the 90%.

The recognition rate obtained from testing dataset was 83,8%.

The same test was performed on 20 individuals from the LFW database. Except of the 10 of them listed in table 7.1, 10 further individuals were added to the training data:

Test results are presented below:

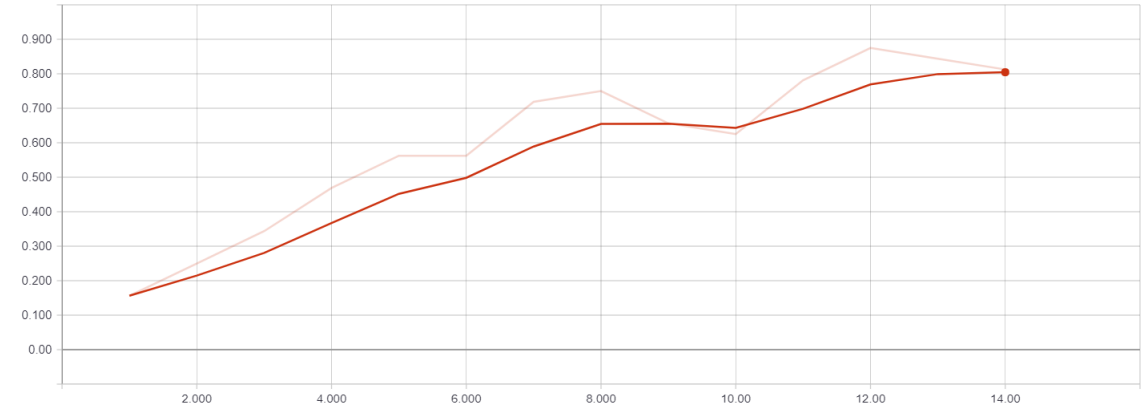


Figure 7.5: Training accuracy

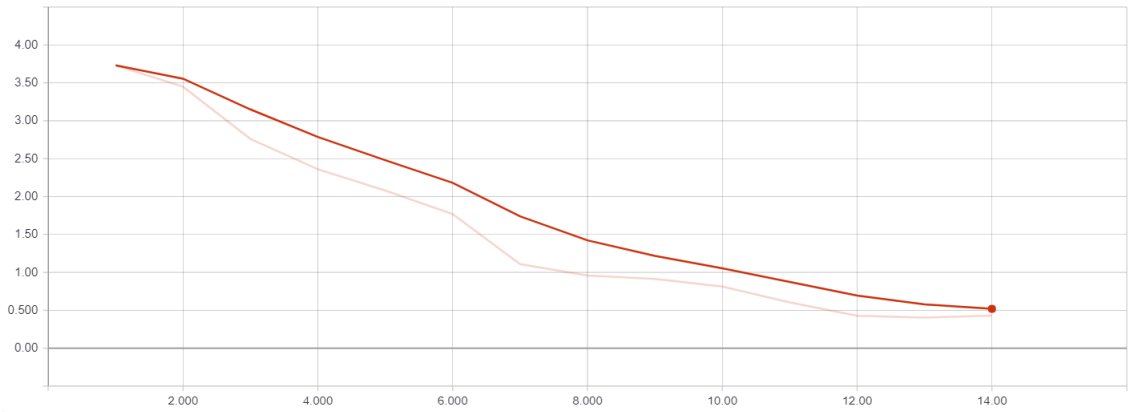


Figure 7.6: Validation loss

The training accuracy reached 80%, the recognition rate was 75,4%.

Test results are worse, not only because of the bigger number of classes to classify, but also due to not enough amount of training data for each class.

The pictures from Chicago Face Database were not tested, because of not sufficient training data that is required for this algorithm.

8. Support Vectors Machine

8.1. Algorithm background

SVM belong to the class of maximum margin classifiers. They perform pattern recognition between two classes by finding a decision surface that has the maximum distance to the closest points in the training set which are termed support vectors.

We start with a training set of points $x_i \in R$, $i = 1, 2...N$ where each point x_i belongs to one of two classes identified by the label $y \in 1, -1$ Assuming that data is linearly separable, the goal of SVM is to separate the two classes by a hyperplane.

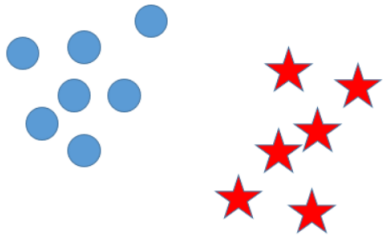


Figure 8.1: Linearly separable dataset and its hyperplane

The hyperplane is a set of points satisfying equation 8.1

$$f(x) = w^T x + b = 0 \tag{8.1}$$

The vector w is known as the weight vector, and b is called the bias. Suppose the weight vector can be expressed as a linear combination of the training examples, i.e.

$$w = \sum_{i=1}^{N} \alpha_i x_i \tag{8.2}$$

Then:

$$f(x) = \sum_{i=1}^{N} \alpha_i x_i^T x + b \tag{8.3}$$

The representation in terms of the variables α_i is known as the dual representation of the decision boundary.

The hyperplane divides the space into two: the sign of the discriminant function denotes the side of the hyperplane a point is on.

8.1.1. The geometric margin

For a given hyperplane we denote by x_+ the closest point to the hyperplane among the positive examples and by x_- the closest point to the hyperplane among the negative examples. These points are commonly called support vectors. The margin of a hyperplane f can be described as:

$$m(f) = \frac{1}{2}\hat{\mathbf{w}}^T(x_+ - x_-) \tag{8.4}$$

where $\hat{\mathbf{w}}$ is a unit vector in the direction of w. We assume that x_+ and x_- are equidistant from the hyperplane:

$$f(x_+) = w^T x_+ + b = af(x_-) = w^T x_- + b = -a \tag{8.5}$$

where $a > 0$

Adding the two equations and dividing by $||w||$ we can describe the geometric margin as:

$$m(f) = \frac{1}{||w||} \tag{8.6}$$

The purpose of the algorithm is to maximize the geometric margin. This optimization problem can be also seen as minimizing $||w||^2$

$$\begin{aligned} &\min_{w,b} \frac{1}{2}||w||^2 \\ &s.t. \forall i, y_i(w^T x_i + b) \geq 1 \end{aligned} \tag{8.7}$$

The constraints in this formulation ensure that the maximum margin classifier classifies each example correctly. However, in our optimization problem, we want to introduce a kind of "penalty" for data misclassification. This can be done with changing our optimization problem to so-called soft-margin SVM:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i \\ &s.t. \forall i, y_i(w^T x_i + b) \geq 1 - \xi_i \end{aligned} \tag{8.8}$$

where $\xi_i \geq 0$ are so-called *slack variables* that allow an example to be in the margin ($0 \leq \xi_i \geq 1$, also called a margin error) or to be misclassified ($\xi_i > 1$).

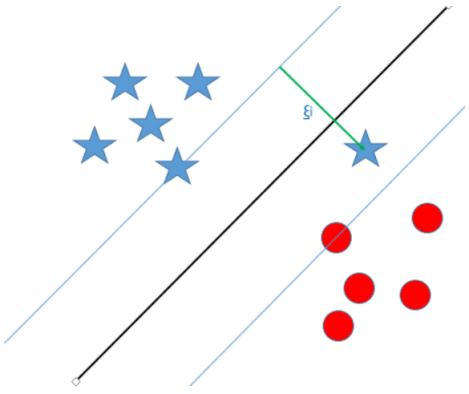


Figure 8.2: Illustration of ξ value

The parameter C is a called *slack penalty* and is one of the SVM "tuning" parameters.

The solution to this optimization problem can be solved using Lagrange duality principle.

Classification of a new data point is performed by computing equation 8.9 with previously optimized parameters.

$$h_{w,b}(x) = g(w^T x + b) \tag{8.9}$$

where w and b are coefficients of the hyperplane trained by the SVM algorithm and $g(z) = 1$ if $z \geq 0$, and $g(z) = -1$ otherwise.

8.1.2. Non-linear separable dataset

The described approach works only for a linearly separable dataset. Unfortunately, in many applications a non-linear classifier is required. Linear classifiers have several advantages, one of them being that they often have simple training algorithms. This begs the question: Can we extend the entire linear-classifier construction to generate non-linear decision boundaries?

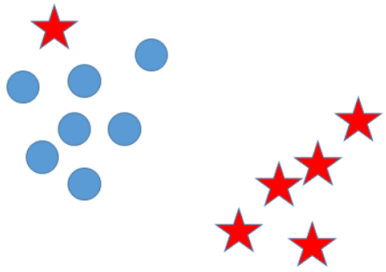


Figure 8.3: Nonlinearly separable dataset and its hyperplane

The solution to this problem is to map our data from the input space X to a feature space F using a non-linear function $\phi : X \to F$, so that the entire construction can be extended to the case of nonlinear separating surfaces.

Each point x_i in the input data is mapped to a point $z = \phi(x)$ of higher dimensional space, called the feature space.

Rather than applying SVMs using the original input attributes x, we teach the algorithm using new features $\phi(x)$ instead.

To map our training set into the higher dimensional space, we choose the similarity

For each x_i, we calculate the similarity value between this and every other sample in the dataset. The similarity function is also called a kernel.

$$\begin{aligned} f_{i1} &= K(x_i, x_1) \\ f_{i2} &= K(x_i, x_2) \\ &\vdots \\ f_{iN} &= K(x_i, x_N) \end{aligned} \tag{8.10}$$

In the new high dimensional space each point x_i is represented as a vector:

$$\vec{f_i} = [f_{i1}, f_{i2}, \cdots, f_{iN}] \tag{8.11}$$

In the feature space F, expression 8.3 is given by:

$$f(x) = \sum_{i=1}^{N} \alpha_i \phi(x_i)^T \phi(x) + b \tag{8.12}$$

The feature space F may be very high dimensional, hence computing the whole equation becomes computationally expensive unless the kernel function K, defined by the equation 8.13 can be computed efficiently.

$$K(x, z) = \phi(x)^T \phi(z) \tag{8.13}$$

Using the kernel function, the decision boundary function can be defined as:

$$f(x) = \sum_{i=1}^{N} \alpha_i K(x, x_i) + b \tag{8.14}$$

The two most widely used kernels are:

1. Gaussian kernel
$$K(x, x_i) = exp(-\frac{||x - x_i||^2}{2\sigma^2}) \tag{8.15}$$
where σ is a parameter that controls the width of Gaussian
2. Polynomial kernel
$$K(x, x_i) = [(x \cdot x_i + 1]^d \tag{8.16}$$
with degree d

8.1.3. Multi-class classification problem

There are two basic approaches for solving n-class problems with SVMs:

1. one-vs-all approach - q SVMs are trained. Each of the SVMs separates a single class from all remaining classes.
2. pairwise approach - q(q-1)/2 SVMs are trained. Each SVM separates a pair of classes. The pairwise classifiers are arranged in trees, where each tree node represents an SVM.

8.1.4. Face recognition

The system has a linear SVM for every person in the database. Each SVM is trained to distinguish between all images of a single person and all other images. Given a set of q people and a set of q SVMs, each one associated with one person, it is relatively easy to perform recognition. Specifying the threshold, the distance between input and given SVM models are computed as follows:

$$d(x) = \frac{\sum_{i=1}^{l} \alpha_i y_i x_i \cdot x + b}{||\sum_{i=1}^{l} \alpha_i y_i x_i||} \tag{8.17}$$

The sign of d is the classification result for x, and $||d||$ is the distance from x to the hyperplane.

The class label of the input picture is computed as follows:

$$y = \begin{cases} n \text{ if } d_n(x) > t \\ \text{not recognized if } d_n(x) \leq t \end{cases} \quad \text{where } d_n(x) = max(d_{i=1}^{q})$$

8.2. Implementation and test results

The SVM algorithm was implemented with Python 3.6.1 with the usage of numpy, PIL and sklearn libraries.

The first step in the algorithm is data preprocessing. As in the Multilayered Perception approach, the images were preprocessed with the PCA algorithm in order to reduce the dimensionality of input data, hence reducing the computational complexity of the algorithm. The algorithm is using a one-vs-one approach.

The tests were performed on Chicago Face Database and Labeled Faces in Wild database.

The main goal of the experiments was to examine the influence of parameter C and different kernels on the accuracy of the algorithm.

8.3. Tests on pictures captured in a controlled environment

8.3.1. 20 individuals

The first experiment was performed on 20 individuals from the CFD database. The Gaussian and Polynomial kernels were tested.

The Gaussian kernel as a similarity function that measures the "distance" between a pair of examples, (x_i, x_j). The Gaussian kernel is also parameterized by a bandwidth parameter σ (8.15), which determines how fast the similarity metric decreases to 0 as the examples are further apart.

The kernel function (8.15) from can be also interpreted as:

$$exp(-\gamma * ||x - x_i||^2) \tag{8.18}$$

where:

$$\gamma = \frac{1}{2\sigma^2} \tag{8.19}$$

The influence of the parameter γ on the Gaussian kernel function can be visualized as presented in figure 8.4. (with the smallest value of γ on the left graph and the greatest on the right)

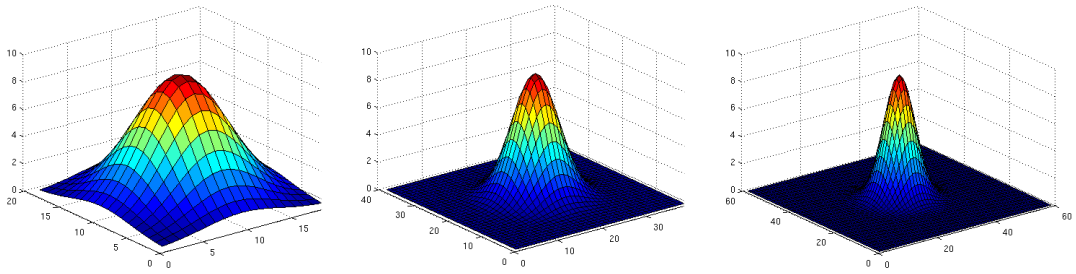


Figure 8.4: Gaussian kernel visualization

The figure 8.5 presents the SVM training accuracy with varying γ and C parameter:

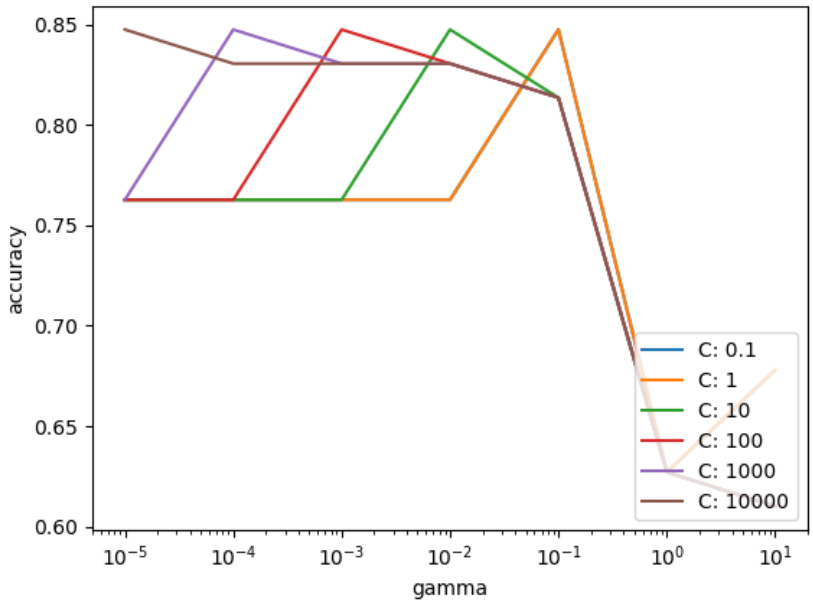


Figure 8.5: Training score with varying gamma and C parameter for 20 individuals

The best score (0.85) was obtained with several configurations of C and γ parameters. The further testing was performed with $C = 10^3$ and $\gamma = 10^-4$

In this configuration, the SVM algorithm reached 80% of recognition rate on the testing samples.

The same experiment was performed with the usage of the polynomial kernel. The influence of polynomial degree was examined and is presented in figure 8.6.

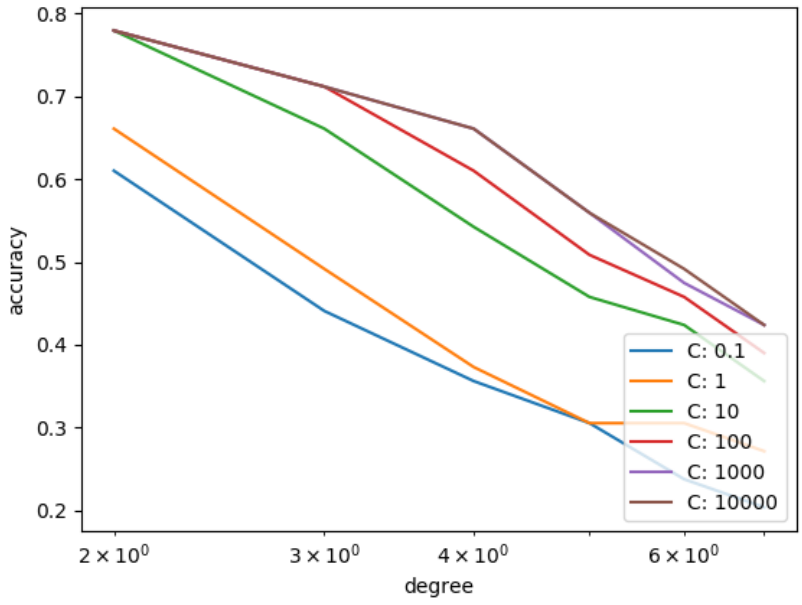


Figure 8.6: Training score with varying polynomial degree and C parameter for 20 individuals

In Figure(8.6) we can see that our data can be nicely separated with a polynomial of the second degree. The recognition rate was tested on SVM trained with $C = 10$ and the degree $= 2$.

The obtained result is 70%, which is 10 percentage points less than the result of SVM trained with a Gaussian kernel.

8.3.2. 40 individuals

The same test scenario was used to examine SVM on the bigger amount of input data - 40 individuals.

The results are presented below.

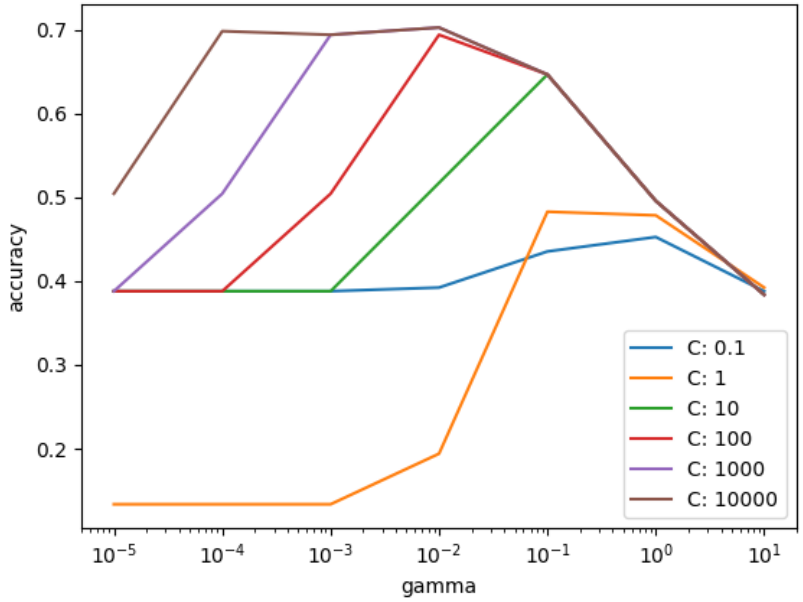


Figure 8.7: Training score with varying gamma and C parameter for 40 individuals

The best score (0.71) was obtained with several configurations of C and γ parameters. The further testing was performed with $C = 10^3$ and $\gamma = 10^-3$

In this configuration, the SVM algorithm reached 60% of recognition rate on the testing samples.

The same experiment was performed with the usage of the polynomial kernel. The influence of polynomial degree was examined and is presented in figure 8.8.

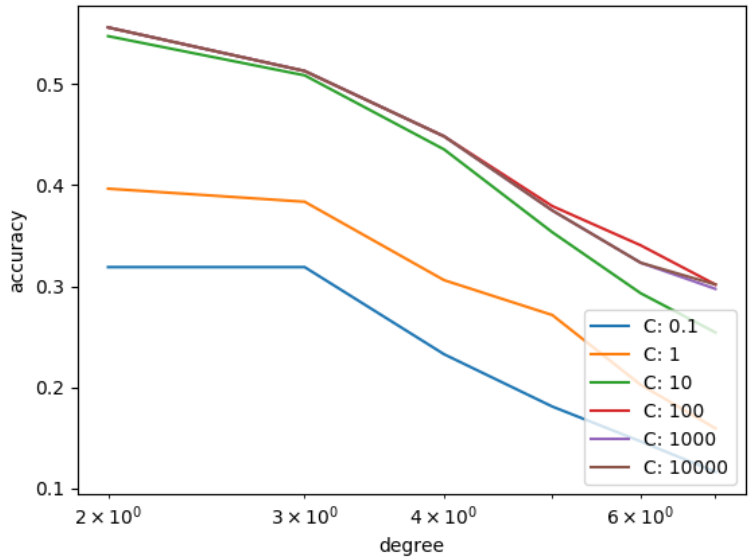


Figure 8.8: Training score with varying polynomial degree and C parameter for 40 individuals

The best score was obtained with degree = 2 and $C = 10^4$. With these parameters, the recognition rate reached 52%, which is still worse than SVM trained with a Gaussian kernel.

8.4. Tests on pictures captured in an uncontrolled environment

To test the algorithm, the Labeled Faces in Wild database was used. The quality of images is much worse than in the previous examples, so the tests result are expected to be worse.

The results are presented below.

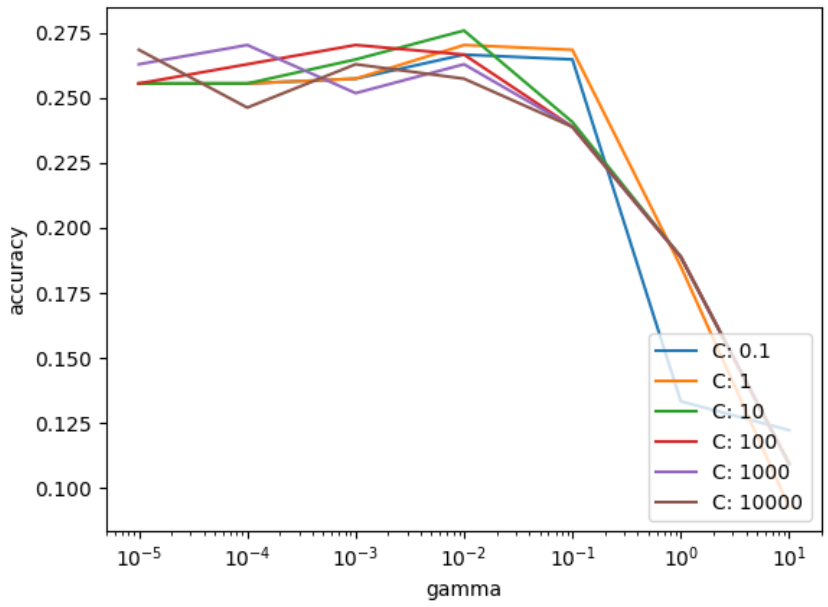


Figure 8.9: Training score with varying gamma and C parameter for 20 individuals

The best training score (0.278) was obtained with $C = 10$ and $\gamma = 0.01$ In this configuration, the SVM algorithm reached 26,6% of recognition rate on the testing samples.

The same experiment was performed with the usage of the polynomial kernel. The influence of polynomial degree was examined and is presented in figure 8.10.

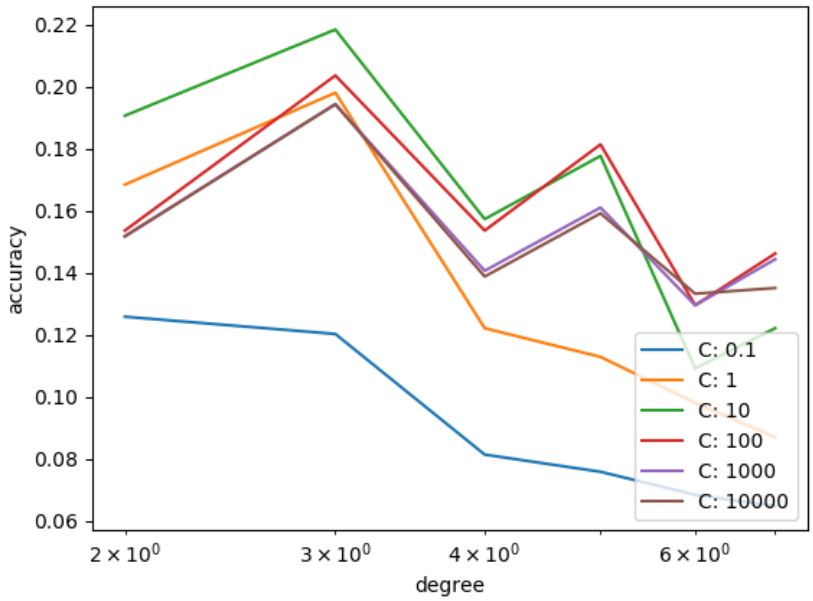


Figure 8.10: Training score with varying polynomial degree and C parameter for 20 individuals

In this case, SVM obtained best results with a third polynomial degree and $C = 10$. The obtained recognition rate was 20%.

To compare obtained results a tabular summary of the recognition rate for SVM algorithm with different kernels on each database is presented (Table 8.1).

Table 8.1: A tabular summary of the recognition rate for SVM algorithm with different kernels on each database

Database	Gaussian kernel	Polynomial kernel
CFD 20	80%	70%
CFD 40	60%	52%
LFW	26.6%	20%

From these tests we can conclude that the Gaussian kernel allows us to obtain better results for every database. As expected, the results are significantly better for pictures captured in a controlled environment and a small amount of data. Even though SVM has an advantage of small computing time, the results do not meet the expectations of a good facial recognition system.

9. Summary

A tabular summary of the recognition rate for each algorithm is presented below:

Table 9.1: Tabular summary of the recognition rate for each algorithm

Algorithm	CFD 20	CFD 40	LFW 20
PCA	75%	65%	8,5%
MLP	90%	70%	73,3%
CNN	-	-	75,4%
SVM	80%	60%	26,6%

For a database composed of pictures captured in a controlled environment (Chicago Face Database) all tested algorithms obtained high recognition rates. Unfortunately, there are not many circumstances in reality, where the face recognition system will be applied to such data. Nevertheless, if we are dealing with good quality pictures with reduced noises and other factors that make the recognition hard to perform, we can successfully use all of the listed algorithms. The best choice among them would be Multilayer Perceptron. The CFD database contains 4 pictures per individual, which was not enough to apply the CNN algorithm. If the given amount of training data was sufficient, the Convolutional Neural Network would probably give us the best results.

The analysis of the algorithm results in a database composed of pictures captured in an uncontrolled environment is much more interesting. Creating a system capable of working with such data is harder, but also more useful. In this case, the best results were given by Artificial Neural Networks, with Convolutional Neural Network being the most successful. Unfortunately, both CNN and MLP suffer from a large amount of computation in the training phase.

Bibliography

[1] Haykin, Simon S., et al. *Neural networks and learning machines.* Vol. 3. Upper Saddle River, NJ, USA:: Pearson, 2009.

[2] Krizhevsky Alex, Ilya Sutskever, and Geoffrey E. Hinton. *Imagenet classification with deep convolutional neural networks.* Advances in neural information processing systems. 2012.

[3] Lawrence, Steve, et al. *Face recognition: A convolutional neural-network approach.* IEEE transactions on neural networks 8.1 (1997): 98-113.

[4] Guo, Guodong, Stan Z. Li, and Kap Luk Chan. *Support vector machines for face recognition* , Image and Vision computing 19.9-10 (2001): 631-638.

[5] Navin Prakash, Dr.Yashpal Singh. *Support Vector Machines for Face Recognition* International Research Journal of Engineering and Technology (IRJET) Vol 2, Issue: 08, November 2015

[6] Simard, Patrice Y., David Steinkraus, and John C. Platt. *Best practices for convolutional neural networks applied to visual document analysis* ICDAR. Vol. 3. 2003.

[7] Turk, Matthew A., and Alex P. Pentland. *Face recognition using eigenfaces.* Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on. IEEE, 1991.

[8] Zhao, Wenyi, et al. *Discriminant analysis of principal components for face recognition.* Face Recognition. Springer, Berlin, Heidelberg, 1998. 73-85.

[9] Jain, Anil K., and Stan Z. Li. *Handbook of face recognition.* New York: Springer, 2011.

[10] M. Kirby and L. Sirovich, *Application of the karhunen-loeve procedure for the characterization of human faces* IEEE Pattern Analysis and Machine Intelligence, vol. 12, no. 1, pp. 103-108, 1990.

[11] Xavier Serra, Polytechnic University of Catalonia, Master thesis: *Face recognition using Deep Learning*, January 2017

[12] Kumar, Dinesh, C. S. Rai, and Shakti Kumar. *Face recognition using self-organizing map and principal component analysis.* Neural Networks and Brain, 2005. ICNN B'05. International Conference on. Vol. 3. IEEE, 2005.

[13] Kim, Kwang In, Keechul Jung, and Hang Joon Kim. *Face recognition using kernel principal component analysis.* IEEE signal processing letters 9.2 (2002): 40-42.

[14] Phillips, P. Jonathon. *Support vector machines applied to face recognition.* Advances in Neural Information Processing Systems. 1999.

[15] Andrej Karpathy, CS231n Convolutional Neural Networks for Visual Recognition Retrieved from: http://cs231n.github.io/convolutional-networks/

[16] Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen, Adam Coates, Andrew Maas, Awni Hannun, Brody Huval, Tao Wang, Sameep Tandon, *Convolutional Neural Network* Retrieved from http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[17] Andrew Gibiansky, *Convolutional Neural Networks*, Retrieved from: http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/, 2014

[18] Jefkine Kafunah, *Backpropagation In Convolutional Neural Network* Retrieved from: http://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/, 2015

[19] Lindsay I Smith, *A tutorial on Principal Components Analysis*, February 26, 2002