AGH

University of Science and Technology in Cracow

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

DEPARTMENT OF AUTOMATICS

MASTER OF SCIENCE THESIS

MAGDA NOWAK-TRZOS

Face recognition system based on deep neural networks

SUPERVISOR:

Adrian Horzyk Ph.D

Cracow 2018

. . .

Abstract

This paper provides an evaluation of various machine learning algorithms concerning the problem of facial recognition. First a survey of available facial recognition machine learning algorithms will be presented. The work will focus on comparing four algorithms: PCA (so-called eignenfaces algorithm), Mulilayered 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 the applications capable of verifying or identifying a person based on a digital image. They have a wide range of usage such as identity authentication or security and access control. Interest in this topic have increased significantly over the past decade, because it has potential to be the less invasive and user-friendly approach in human identification. Facial recognition applications become less expensive, making their use more widespread. Benefits of automatic facial recognition systems can be seen not only in security systems, but also in a 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.

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

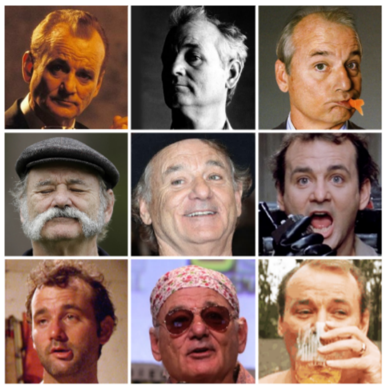


Figure 1.1: Very different pictures of a single individual

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. Gathered information were used by computers in further facial recognition.

In 1988, Sirovich and Kirby 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 accurately code a normalized face image. Few years later Turk and Pentland expanded the approach by discovering how to detect faces within images. This 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, deep neural network approach is one of the most actively investigated method 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 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 data set. 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. Faces have a wide variability in poses, shapes, sizes and texture. The problems or challenges in face detection and recognition are listed as follow:

– Illumination - the amount and type of light present during the image is captured can differ

– Pose - a face varies depends on the position of the camera.

– Presence of structural components such as moustache or beard

– Facial expression

– Face occlusion such as glasses, scarf, mask etc.

– Image quality

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 relationships 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 data set, 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 the 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
Self-organizing map is a type of artificial neural network that belongs to a category of competitive learning networks. 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. 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 suggest, 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

Nevertheless, 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 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.

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

3. 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 reduce number of feature vectors. The main advantages of the PCA are its low sensitivity to noise, the reduction of the requirements of the memory, and the increase in the efficiency due to the operation in a space of smaller dimensions. In face recognition problem PCA is based on extracting the characteristic features of the face and representing the face as a linear combination of so called eigenfaces obtained from the feature extraction process. PCA for face recognition is usually called Eigenfaces method.

3.1. Algorithm Background

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

$$X = (x_1, x_2, ..., x_{N^2}) \tag{3.1}$$

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.

$$ImagesMatrix = \begin{pmatrix} ImageVector_1 \\ ImageVector_2 \\ \vdots \\ Image1vector_M \end{pmatrix} \tag{3.2}$$

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

$$averageFace = \frac{1}{M}\sum_{i=1}^{M} ImageVector_i \tag{3.3}$$

The graphical representation of $avarageFace$ is presented on $Figure$3.1.

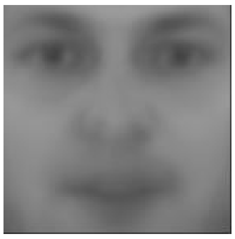


Figure 3.1: Avarage Face graphical representation

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

$$\phi_i = ImageVector_i - averageFace \tag{3.4}$$

$$A = [\phi_1, \phi 2 \cdots \phi_M] \tag{3.5}$$

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

$$C = AA^T \tag{3.6}$$

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

$$C \sim N^2xN^2 \tag{3.7}$$

It turns out that using such a high-dimensional 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

$$C = A^TA \tag{3.8}$$

$$C \sim MxM \tag{3.9}$$

The dimension of matrix C significantly decreased. In result of such an operation we get M eigenvectors (v_i) with M elements each.

$$Cv_i = \lambda_i v_i \tag{3.10}$$

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 corresponds to zero-eigenvalues are discarded.

The set of extracted significant eigenvectors have 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 3.11

$$u_i = Av_i \tag{3.11}$$

$$||u_i|| = 1 \tag{3.12}$$

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

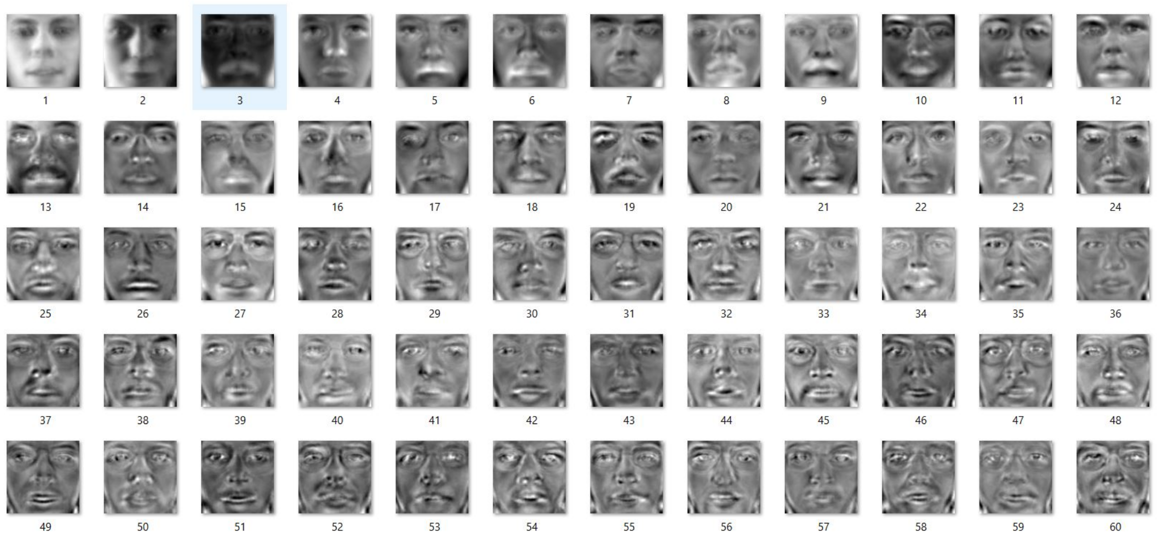


Figure 3.2: Image representation of eigenfaces

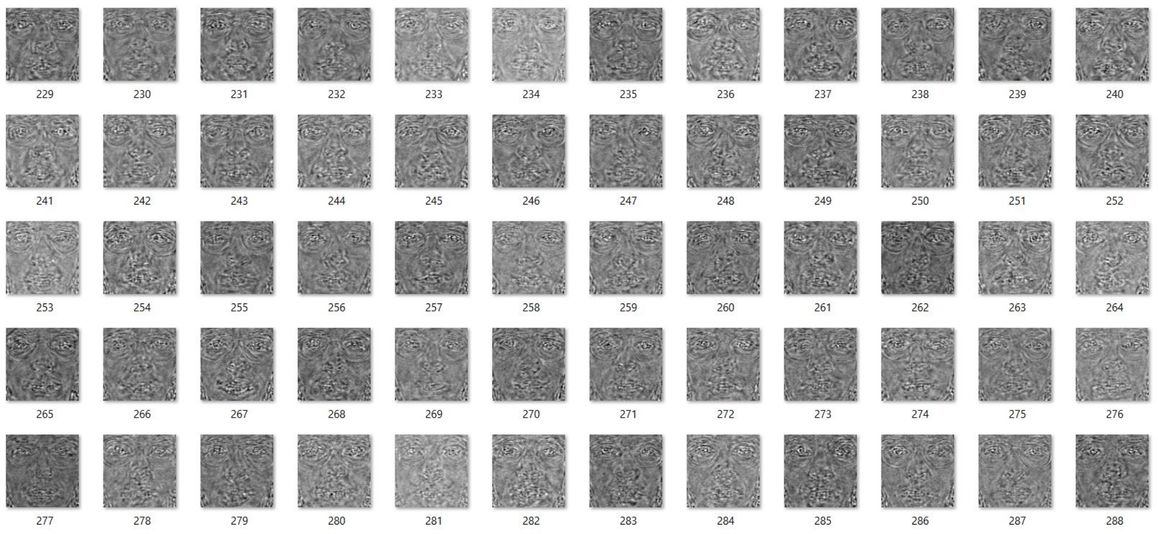


Figure 3.3: Less significant eigenfaces

On figure 3.2, we can observe 50 most significant eigenvectors - the ones with the highest eigenvalues, whereas figure 3.3 presents less significant 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.

$$\Omega_i = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{pmatrix} \tag{3.13}$$

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

$$\Omega = U^T(x - averageFace) \tag{3.14}$$

where U is the set of significant eigenvectors.

One simple way to determine to which face class x belongs is minimizing the Euclidean distance

$$\epsilon_k = ||\Omega - \Omega_k|| \tag{3.15}$$

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 the minimum ϵ_k is smaller than some threshold t. Otherwise face x is classified as unknown.

3.2. Implementation and results

The PCA - Eigenfaces algorithm was implemented with Python 3.6.1 with usage of numpy, scipy and PIL libraries.

The performance of PCA based algorithm was evaluated with two image databases: Chicago Face Database (version 2.0.3) and Labeled Faces in the Wild Database (LFW). The Chicago Face Database consists of good quality images with varying face expression and very similar capturing conditions for each sample. The Labeled Face in the Wild database contains images of peoples captured in various places, with various face expressions, light conditions, pose etc. The quality of images is rather poor. All the images in both the databases were converted to grey scale and normalized.

Samples from each database are presented below:

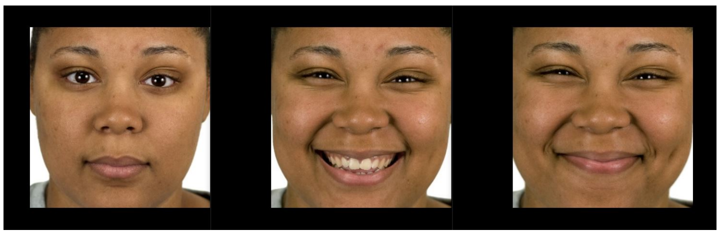


Figure 3.4: Samples from Chicago Face Database

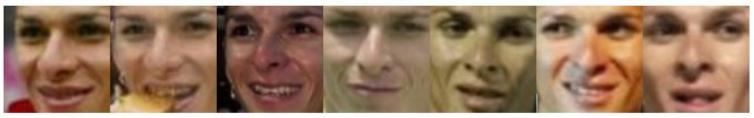


Figure 3.5: Samples from Labeled faces in the Wild

The first test was performed on 158 individuals from Chicago Face Database. Four out of the six images of a person were used for training and the remaining two were used to test the recognition rate. The choice of the training and testing images was made randomly.

Test was ran multiple times for different number of eigenfaces. The Figure 4.6 presents how the recognition rate depends on the amount of eigenfaces on which the tested image is projected.

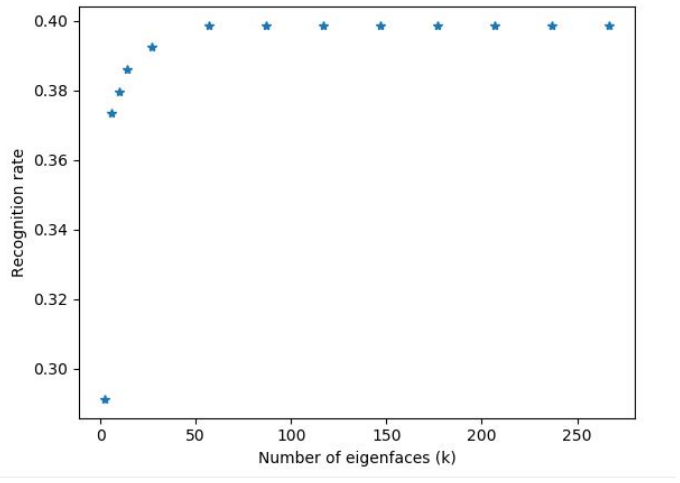


Figure 3.6: Recognition rate with varying number of eigenvectors

The recognition rate stabilizes with $k = 54$ reaching the value of 0.3998, which can be interpreted as the percentage of faces that were classified correctly.

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

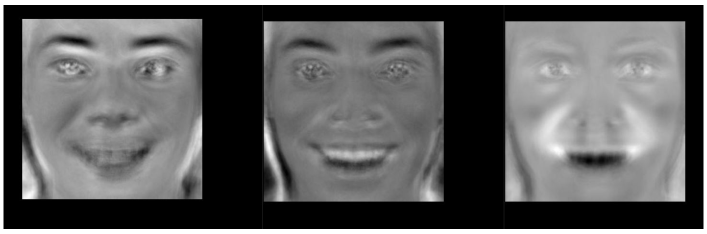


Figure 3.7: CFD eigenfaces

The second test was performed on the set of 40 individuals from Labeled faces in the Wild database. 27 out of 30 images of a person were used for training and the remaining 3 were used to test the recognition rate. The choice of the training and testing images was made randomly. The PCA algorithm achieved $12,22\%$ of accuracy. The result is poor, as expected. The quality of images is by far not good enough, to successfully perform the PCA algorithm.

4. Artificial Neural Networks

One of the most powerful methods to solve the face recognition problem are Artificial Neural Networks. A good definition of ANN, is given by Haykin [TODO] describing ANN as a parallel combination of simple processing unit which can acquire knowledge from 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).

4.1. Biological background

A neuron, in a biological sense, is a cell that carries and processes information - electric signal. The typical neuron is composed of cell body (perikarion) 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 an information from other neurons through the dendrites and transmits the processed information via axons. Axon - dendrites connection is called synapse.

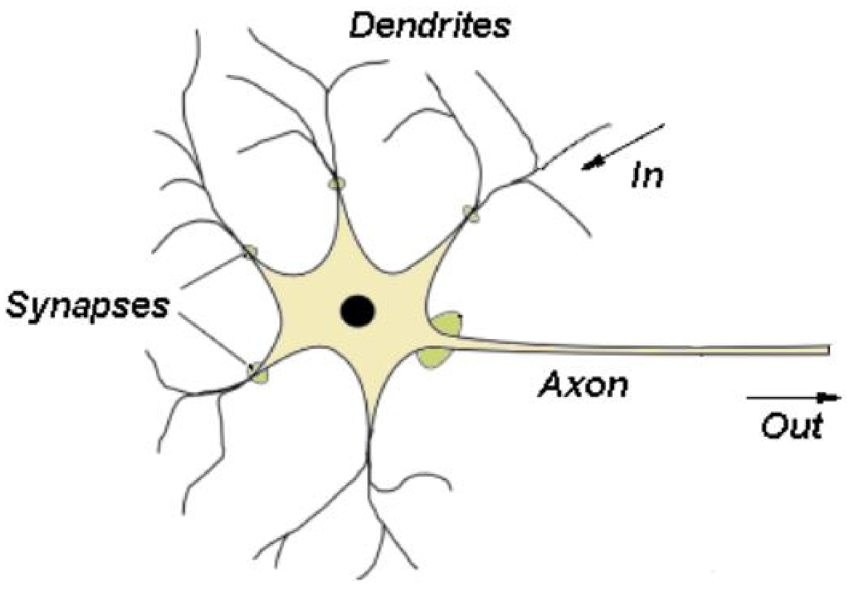


Figure 4.1: Biological Neuron Schema

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

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. This is knows as a hundred step rule: the brain runs parallel programs that are at most 100 steps long for such a tasks.

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 human brain are also hoped to posses some of the most desired brain's characteristics such as:

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

4.2. Artificial Neural Network architecture

4.2.1. 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 so called activation function to the weighted sum of its inputs.

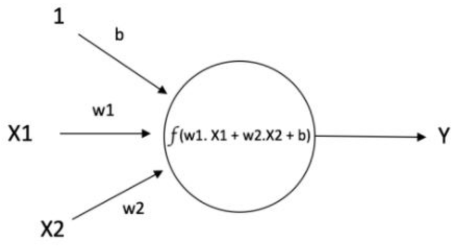


Figure 4.2: Single Neuron

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 most commonly used activation functions are presented below:

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

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{4.1}$$

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

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.2}$$

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

$$f(x) = max(0, x) \tag{4.3}$$

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 (TODO paper by Krizhevsky et al).

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

$$\vec{o} = f(\vec{w} \cdot \vec{x} + b) \tag{4.4}$$

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

4.3. Backpropagation

One of the main requirements for training this kind of algorithms is data. All learning algorithms use data in their training processes, but ANN require more than most.Given the data, there are various learning algorithms, from which gradient descent (backpropagation) is considered the most successful of all of them.

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

The basic idea of gradient descent method is to adjust the parameters of the neural networks in the way that the computed difference 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:

$$E_1(o, o') = \frac{||o - o'||^2}{2} \tag{4.5}$$

The error can be also calculated using Mean Squared Error:

$$E_2(o, o') = \frac{\sum_{k=1}^{K}(o_i - o'_i)^2}{K} \tag{4.6}$$

where K is the size of output vector.

When using a neural network 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:

$$E_3(o, o') = -\sum_{i=1}^{K} o'_i \ln(o_i) \tag{4.7}$$

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} is being calculated.

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

$$\delta_i^{(o)} = \frac{\partial E}{\partial o_i} f'(z_i) \tag{4.8}$$

where z_i is the input to the output layer.

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

$$\delta_i^{(l)} = f'(z_i^{(l)}) \sum_{k=1}^{m^{(l+1)}} w_{i,k}^{(l+1)} \delta_k^{(l+1)} \tag{4.9}$$

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

3. We calculate the derivative:

$$\frac{\partial E}{\partial w_{j,i}^{(l)}} = \delta_j^{(l)} y_i^{(l-1)} \tag{4.10}$$

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

$$\Delta w_{ij} = \lambda z_j^{(l-1)} \delta_j^{(l)} \tag{4.11}$$

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

After choosing the initial weights of the network randomly, 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.

4.3.1. 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 (figure 6.2):

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 result from the previous information processing.

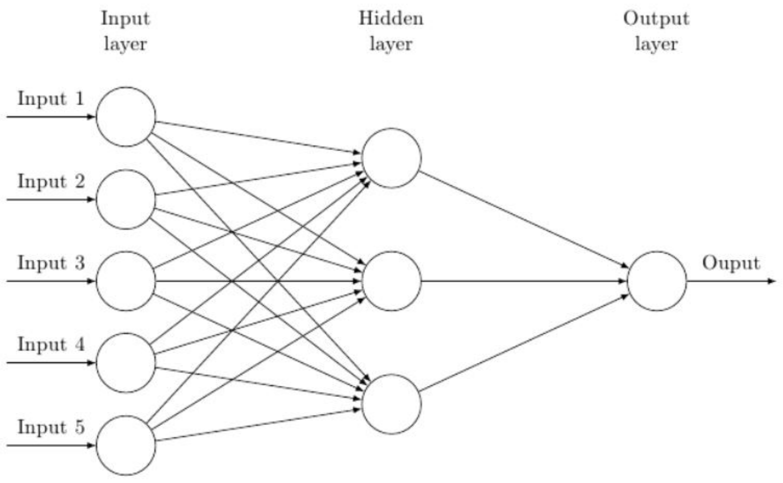


Figure 4.3: Basic Artificial Neural Network Schema

The presented ANN structure performs a feed-forward process. The connections only project forward, that is, 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 ANN, which can differ in the number of neurons in each layer as well as in the amount of layers.

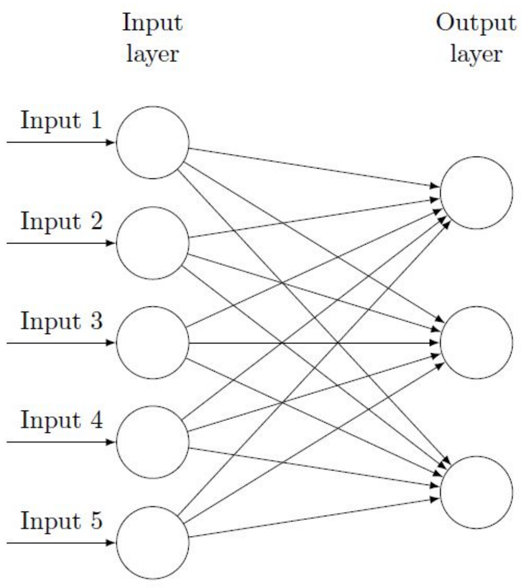


Figure 4.4: Single-layer Artificial Neural Network Schema

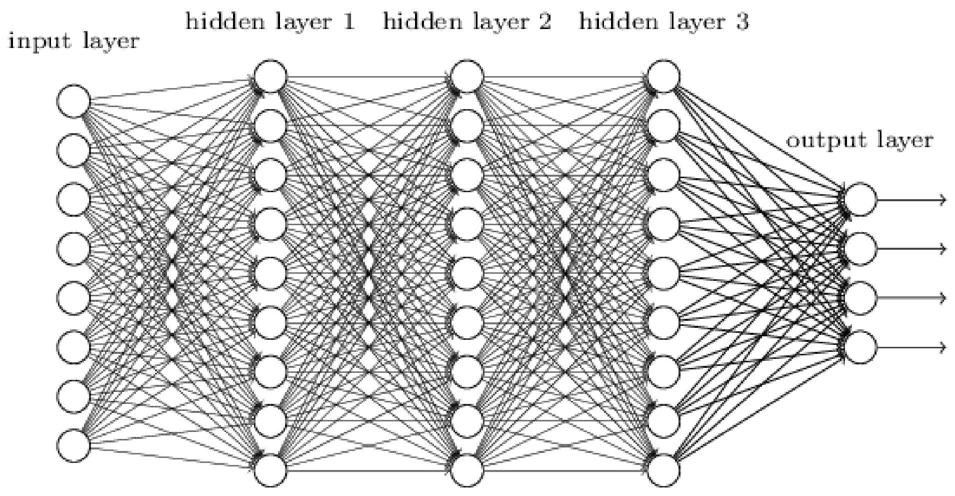


Figure 4.5: Example of Deep Neural Network architecture

The neural network with more than one hidden layer is known as Deep Neural Network. (Figure 6.4). 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.

5. 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 perceptrons 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. Training, so-called backpropagation, involves adjusting the parameters of the model in order to minimize the error. MLP performs well with relatively low dimensional inputs.

5.1. Implementation and test result

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

The first step in this approach is to perform PCA on every image in a training data set. PCA produces a weight vector, (4.13) for an image, hence we can represent each image in a low dimensional space. The dimension depends on the number of eigenvectors chosen for Principal Component Analysis algorithm.

$$X = \begin{bmatrix} w_{11} & w_{21} & \cdots & \cdots & w_{m1} \\ w_{12} & w_{22} & \cdots & \cdots & w_{m2} \\ w_{13} & w_{23} & \cdots & \cdots & w_{m3} \\ w_{14} & w_{24} & \cdots & \cdots & w_{m4} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ w_{1k} & w_{2k} & \cdots & \cdots & w_{mk} \end{bmatrix} \tag{5.1}$$

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

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

$$Y = \begin{bmatrix} 0 & 1 & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & 1 \\ 1 & 0 & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \cdots & 0 \end{bmatrix} \tag{5.2}$$

The goal is to train the model in a way that given the matrix X MLP will produce close approximation of 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
- amount of neurons in hidden layers
- initial weights
- activation function
- learning parameter

The tests were performed on two different databases:

- Chicago Face Database - pictures captured in controlled environment
- Labeled Faces in Wild database - pictures captured in uncontrolled environment, with various lightning conditions, pose, facial expression etc.

5.2. Tests on pictures captured in controlled environment

The database was divided into training and testing samples. The training was performed using 3 pictures of the individual, one pictures was used to test the trained model.

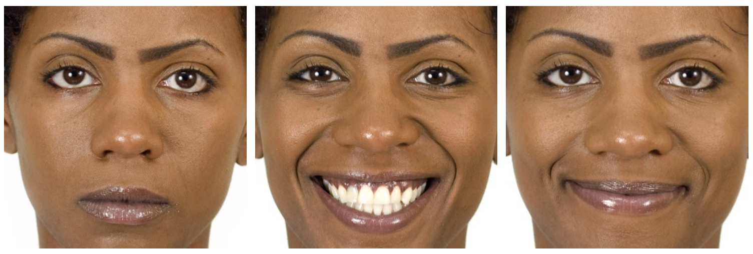


Figure 5.1: Examples of one person pictures in the training dataset

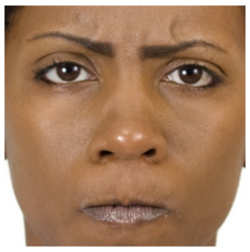


Figure 5.2: Testing image

The tests were performed using MLP topology presented on figure 7.1 (???).

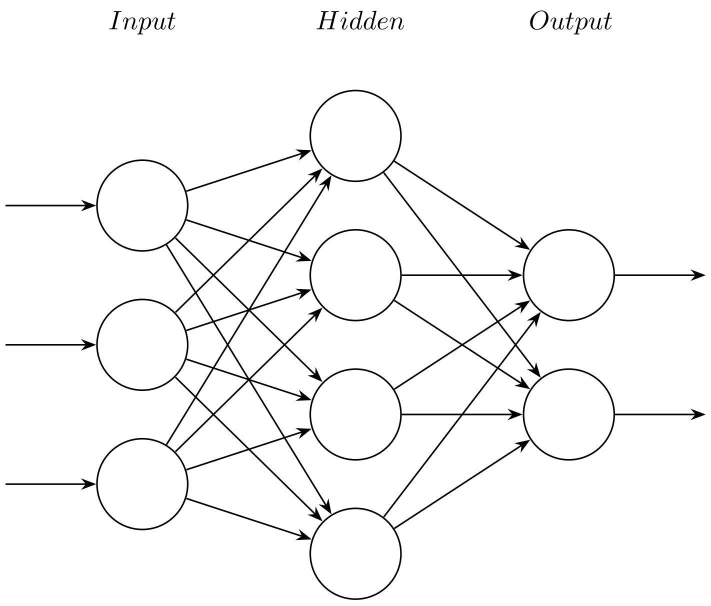


Figure 5.3: Neural Network with one hidden layer

5.2.1. 20 individuals

The first step is to set a number of eigenvectors that the image will be represented with. We do it using previously implemented PCA algorithm.

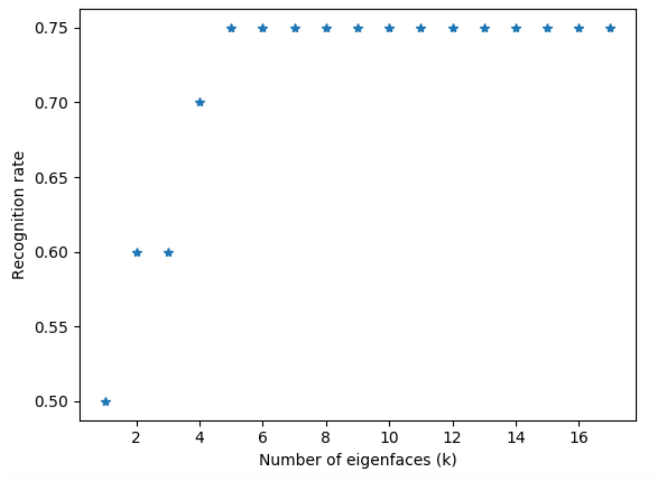


Figure 5.4: Recognition rate with varying number of eigenvectors for 20 individuals

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

Test was performed on a data set with 20 individuals. The PCA-MLP algorithm was ran 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 accuracy of PCA-MLP algorithm we check if the index of obtained outputs' maximum value is equal to the index of maximum value of desired output. The accuracy is a value representing percentage of pictures that fulfills this requirement.

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

Learning rate = 1

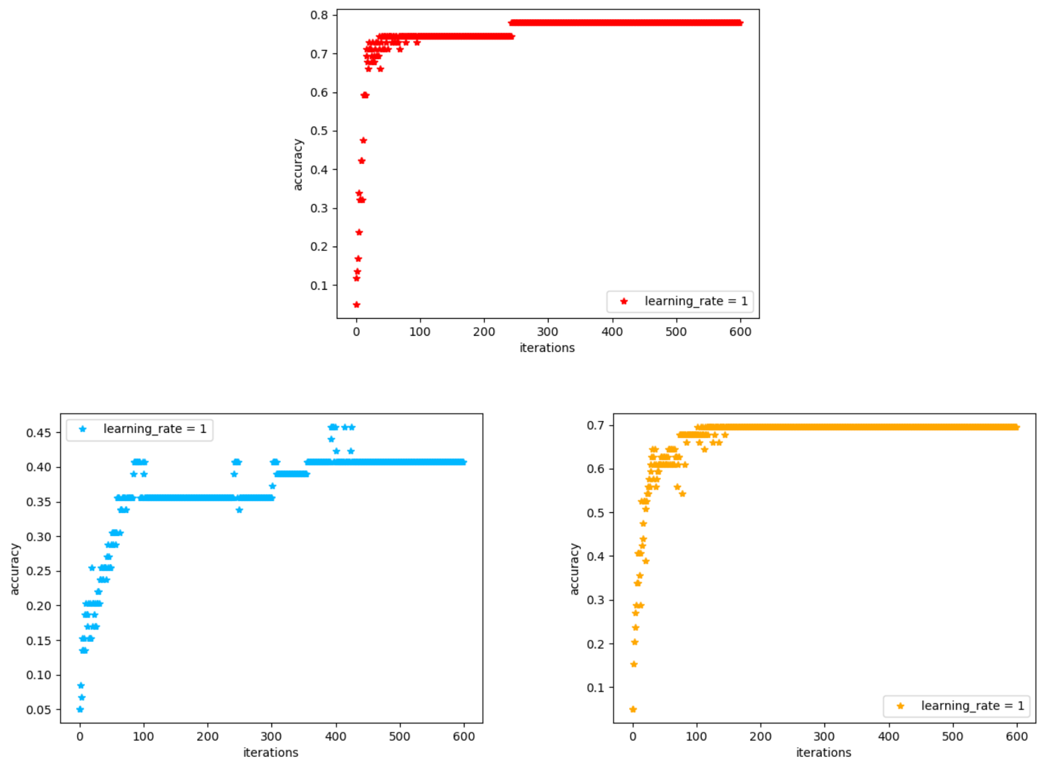


Figure 5.5: Learning rate = 1, initial weights chosen randomly

The neural network with learning rate = 1 is very unstable and strongly depends on initial weights. The tests were performed for different learning rate values:

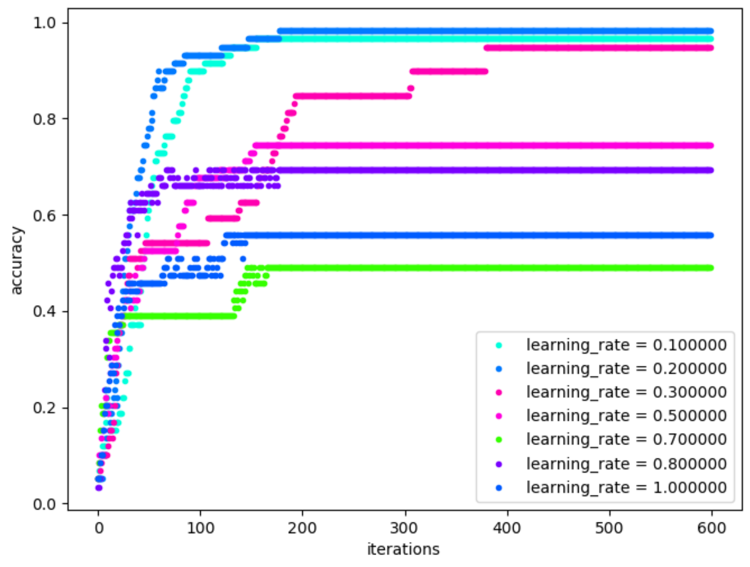


Figure 5.6: Algorithm accuracy with different learning rates

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.

To compare the accuracy of PCA algorithm itself with connection of PCA and MLP algorithms, the PCA results should be examined. The highest obtained recognition rate (figure 7.4) is 0.75.

The testing phase was made using the model that was train 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 result of PCA algorithm.

5.2.2. 40 individuals

The choice of a number of eigenvectors representing the images in a small database is straightforward as presented in section (???).

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

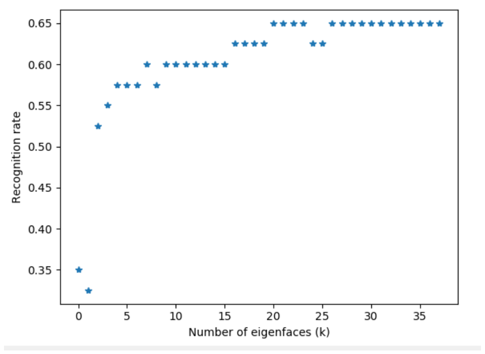


Figure 5.7: Recognition rate with varying number of eigenvectors for 40 individuals

When dealing with a bigger database, we have to find a balance between a sufficient amount of eigenfaces to represent an image and an amount of data that can be applied as the input of multilayer perception network, so that we'll 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 MLP network, hence parameter k was set to 27.

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.

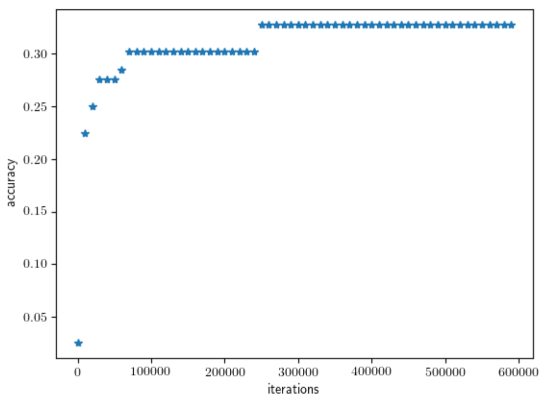


Figure 5.8: number of eigenvectors = 27, number of hidden neurons = 27

The execution of the algorithm took 4 min 54 sek. 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, amount of hidden neurons was equal to the amount 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 rather poor. The next step is to check if increasing or decreasing the amount of neurons in a hidden layer will help to improve the network performance.

Using too few neurons in the hidden layers will result in underfitting. Underfitting occurs when there are too few neurons in the hidden layer to adequately detect the signals in a complicated data set.

On the other hand too many neurons in the hidden layer can result in several problems. First of all overfitting - 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. The proper compromise must be reached between too many and too few neurons in the hidden layer.

There are several rule-of-thumb methods for determining an 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.

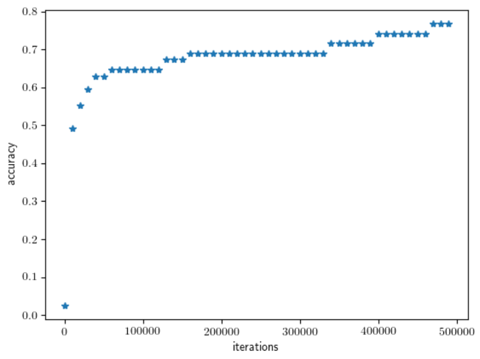
– The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.

– The number of hidden neurons should be less than twice the size of the input layer.

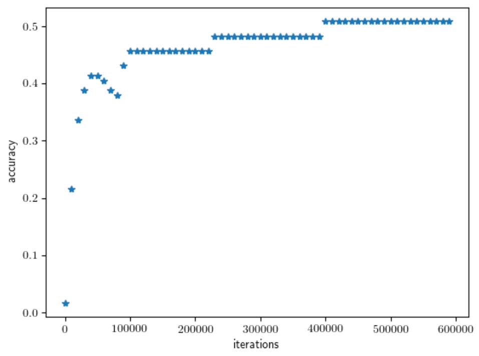
In our case we have 27 input neurons and 40 output neurons. According to the listed rules we have 3 possibilities:

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

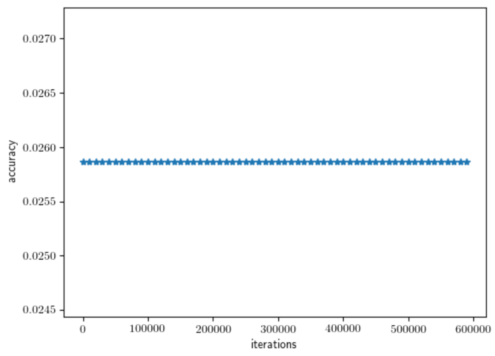
Each of the possibilities was examined. The results not only differ in the accuracy of the algorithm but also in the execution time.



(a) 10 hidden layers



(b) 33 hidden layers



(c) 56 hidden layers

Figure 5.9: Training accuracy for different amount of hidden neurons

Number of hidden neurons	10	33	56
Execution time[sek]	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 PCA algorithm on the same data, it seems that the performance of both is very similar. The reason for that might be not sufficient amount of training data. There is a technique that is facing this problem - so-called data augmentation.

It is a widely used method in many machine learning tasks to enlarge the training data set size and avoid overfitting. The goal of this process is to create new samples from the original training data by, for example, flipping, distorting, adding a small amount of noise to, or cropping a patch from an original image.

The first transformation is shifting the image in four directions - left, right, down, up.

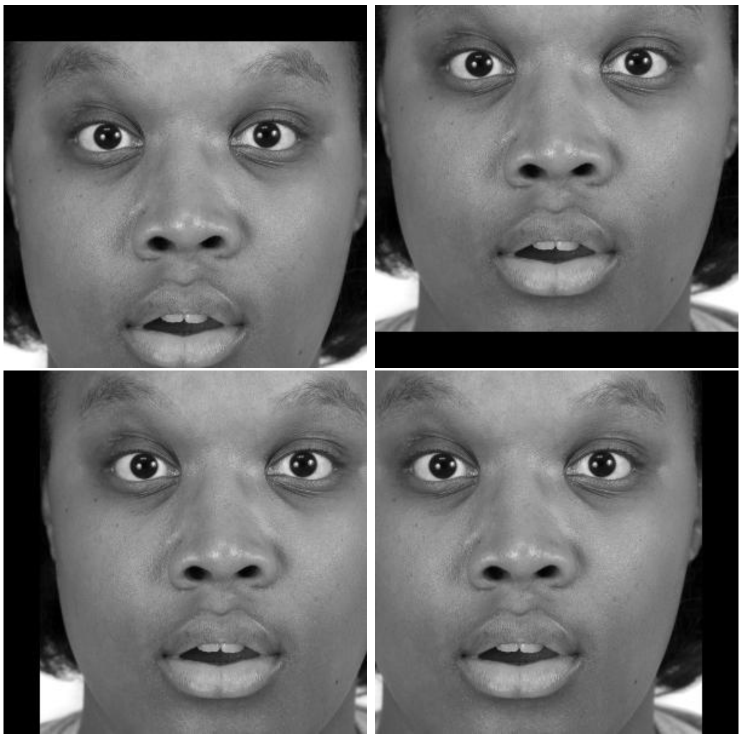


Figure 5.10: Shift transformations

Another applied image transformation is flipping the content of the image from left to right.

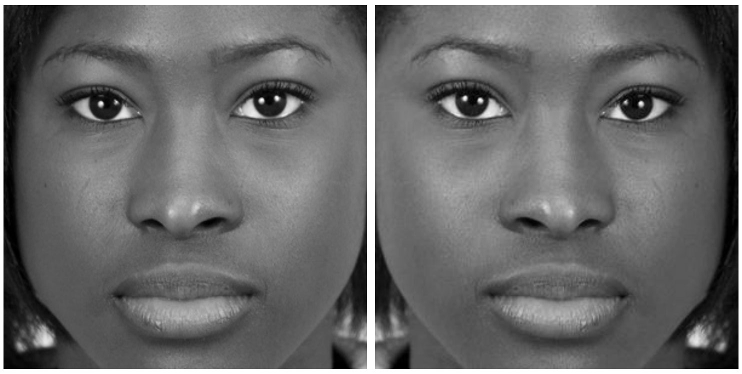


Figure 5.11: Flipping transformation

Increasing amount of training data makes the training process much longer. It took about 15 min to train the network. In our case the results did not improve, but this is mainly because the training data is not varied in terms of shifting. The face is always in the same position in the picture. The recognition rate was 42,5%, which is much worse, then before enlarging the data set.

The algorithm was tested with Chicago Face Database, where all the individuals were captures in controlled environment. The samples of each individual differs only in terms of face expression. It is much more challenging to build a system that is not sensitive for changing the light conditions, the picture quality, face pose and many other factors.

5.3. Tests on pictures captured in uncontrolled environment

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

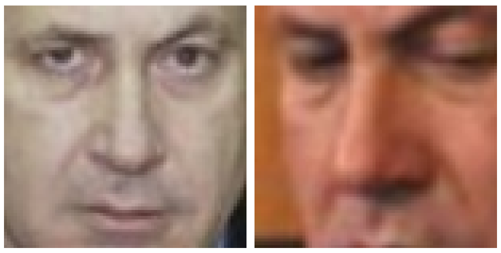


Figure 5.12: Exemplary samples from LFW database

Test was performed on 20 individuals with 30 pictures each. The training set for each person consists of 27 samples, 3 pictures were used to test the model.

5.3.1. One hidden layer

The tests were performed in the same topology as presented in figure(??)

As in the previous examples PCA was performed as a first step. The results (figure 7.13) are very poor as expected.

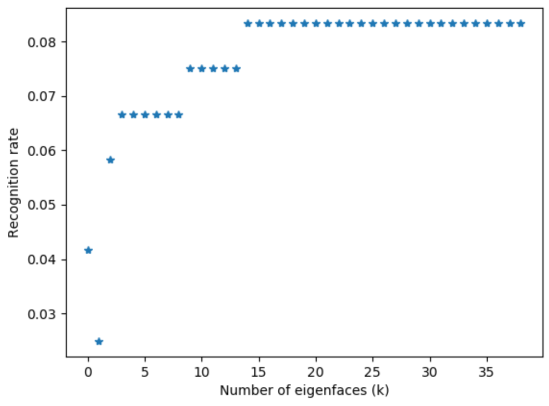


Figure 5.13: Recognition rate with varying number of eigenvectors for 20 individuals

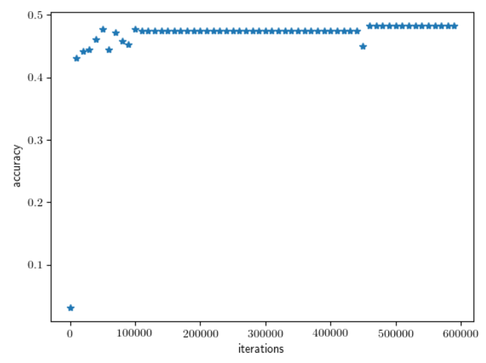
The question is how much can we improve our results with applying MLP network.

Model was trained with with following parameters:

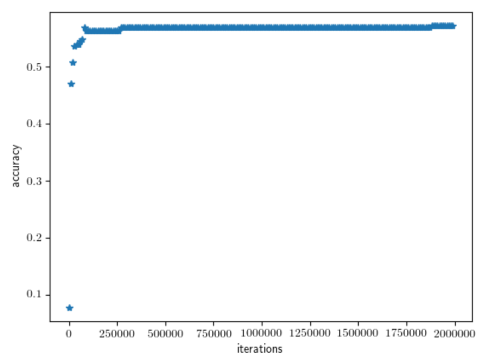
- number of hidden layers = 1
- initial weights chosen randomly in a range [0:1]
- sigmoid activation function
- learning parameter = 0.2

and various dimension of input data (amount of eigenfaces) and different amount of neurons in a hidden layer.

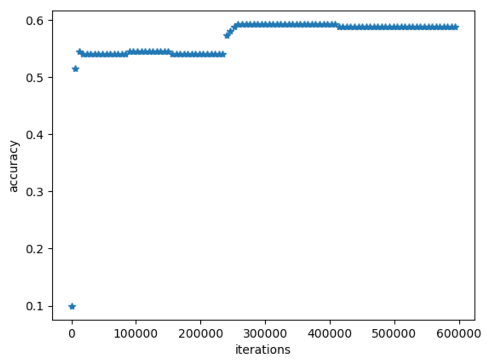
The training results are presented on figure (???):



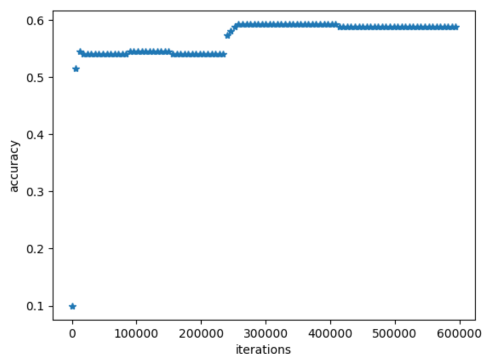
(a) 10 hidden neurons, k = 15



(b) 20 hidden neurons, k = 15



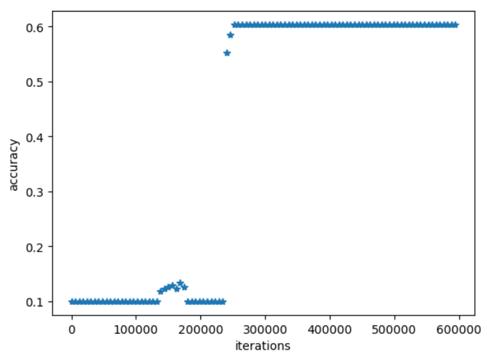
(c) 10 hidden neurons, k = 10



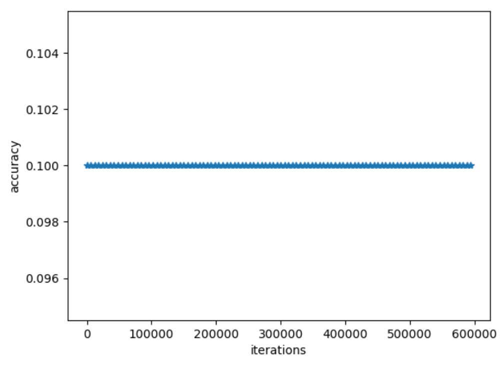
(d) 20 hidden neurons, k = 10

Figure 5.14: Training accuracy for different amount of hidden neurons and eigenfaces

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



(a) 25 hidden neurons, k = 15



(b) 30 hidden neurons, k = 15

Figure 5.15: Training accuracy for different amount of hidden neurons

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

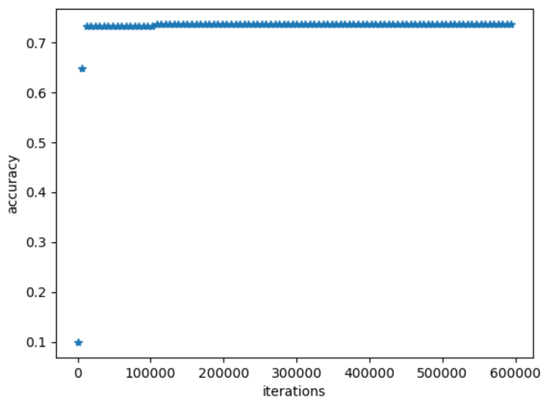


Figure 5.16: 20 hidden neurons, k = 13

The obtained recognition rate was 56.6%, which is much better then PCA result but still not acceptable for facial recognition system. However, taking into consideration the quality and variety of images used for these tests, the result is better than expected.

5.3.2. Two hidden layers

Further experiments were performed on network topology presented on figure(???). The goal is to check if additional hidden layer improves the network performance.

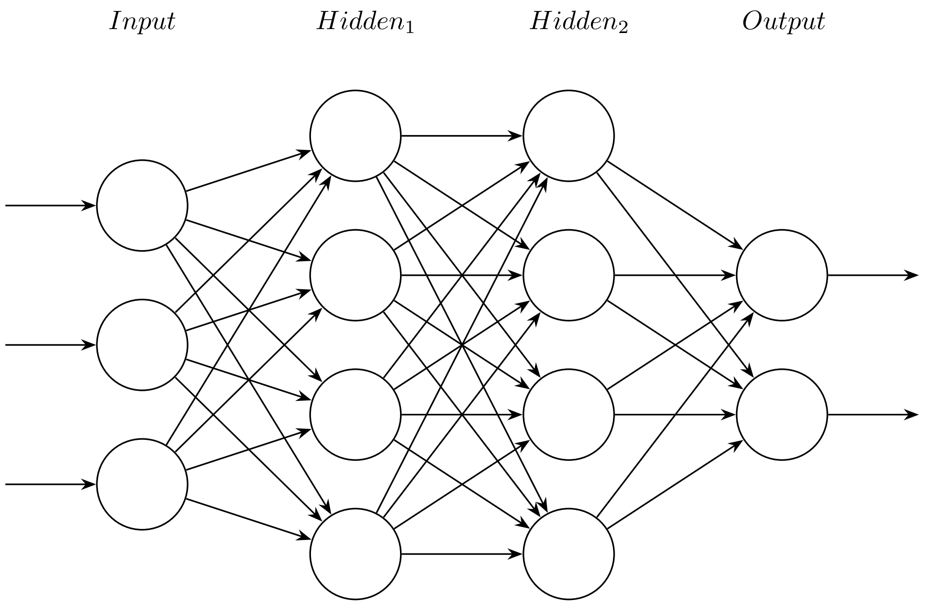


Figure 5.17: Neural Network with two hidden layers

The first tests were ran with following parameters:

– number of hidden layers = 2

– initial weights chosen randomly in a range [0:1]

– sigmoid activation function

- learning parameter = 0.2
- number of eigenfaces = 15

Results are presented on figure(??)

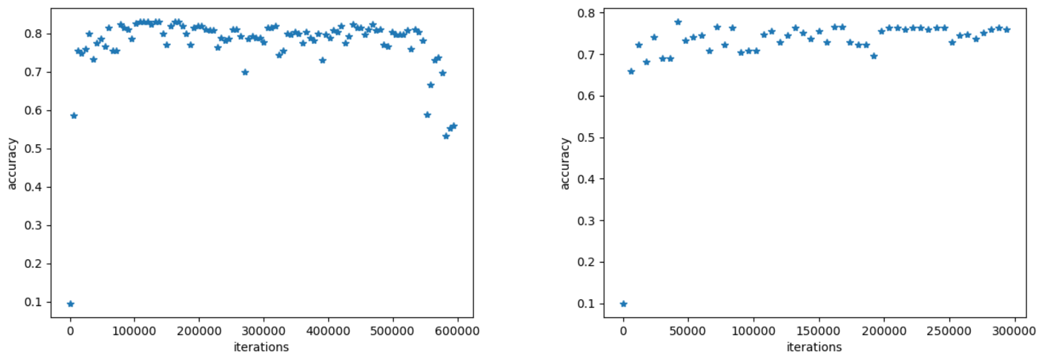


Figure 5.18: 10 hidden neurons in each hidden layers, learning rate = 0.2

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

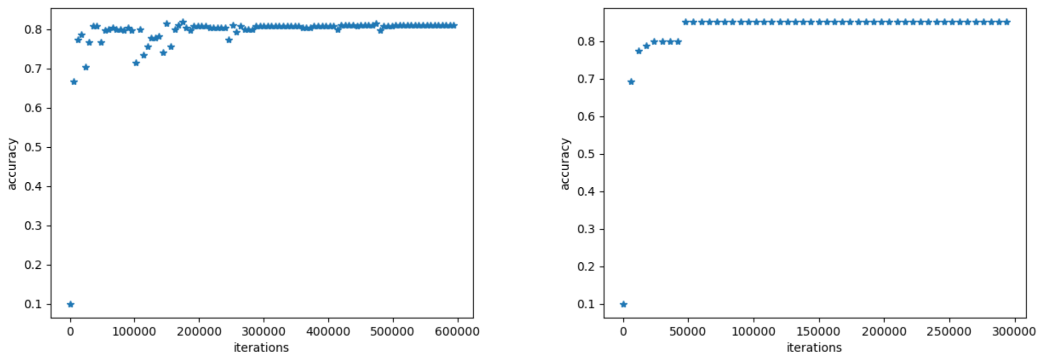


Figure 5.19: 10 hidden neurons in each hidden layers, learning rate = 0.05

Better results were obtained with 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 chosen randomly in a range [0:1]
- sigmoid activation function
- learning parameter = 0.05
- number of eigenfaces = 15

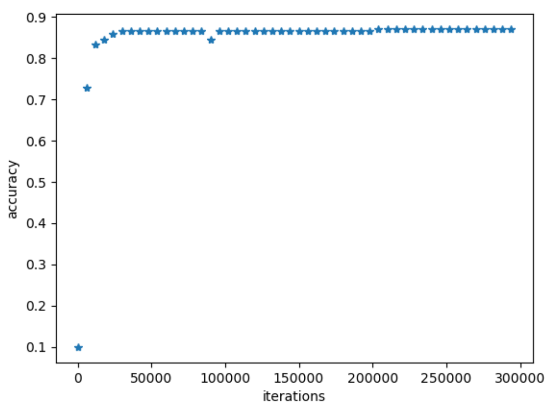


Figure 5.20: 10 neurons in first hidden layer, 15 in the second hidden layer

The obtained recognition rate is 73.3%.

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

6. Convolutional Neural Networks

Convolutional Neural Network is a class of deep, feed-forward neural network, 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 convolutional hidden 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 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, the 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

$$(X * W)_{r,s} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} W_{u,v} X_{r+u,s+v} \tag{6.1}$$

$$W = \begin{bmatrix} w_{-h_1,-h_2} & \cdots & w_{-h_1,h_2} \\ \vdots & w_{0,0} & \vdots \\ w_{h_1,-h_2} & \cdots & w_{h_1,h_2} \end{bmatrix} \tag{6.2}$$

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

The i_th feature map in layer l is computer as

$$z_i^{(l)} = \sum_{j=1}^{m} K_{i,j}^{(l)} * Y_j^{l-1} \tag{6.3}$$

where m is the size of the l layer input.

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

$$Y_i^{(l)} = f\left(z_i^{(l)}\right) \tag{6.4}$$

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 subregion, 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 position, max-pooling is a “smart” way of reducing the dimensionality.

After a sequence of those operation 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.

$$\sigma(o_j) = \frac{e^{o_j}}{\sum_{k=1}^{K} e^{o_k}} \tag{6.5}$$

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 the figure (??)

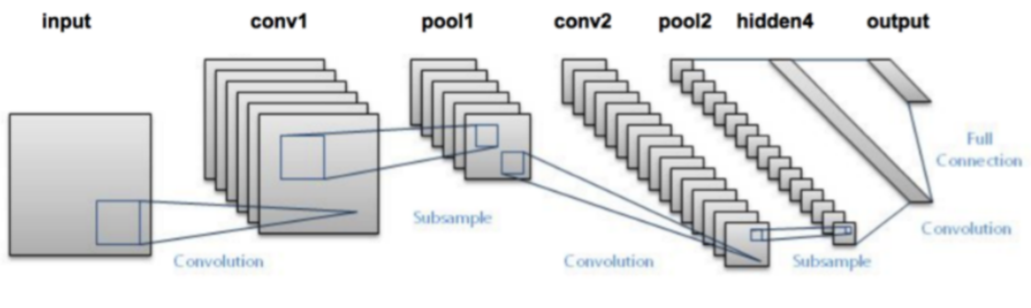


Figure 6.1: A typical convolutional neural network for face recognition

The algorithm is trained with backpropagation. The backpropagation algoritihm 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 flatenning, maxpooling 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 that 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 activations in the output layer.

The softmax activation of the ith output unit is:

$$o_i = \frac{e^{s_i}}{\sum_{k=1}^{K} e^{s_k}} \tag{6.6}$$

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:

$$E(o, o') = -\sum_{i=1}^{K} o'_i \ln(o_i) \tag{6.7}$$

$$E = -\sum_{i=1}^{K} o'_i \ln(o_i) \tag{6.8}$$

Computing the gradient yields

$$\frac{\partial E}{\partial o_i} = -\frac{o'_i}{o_i} \tag{6.9}$$

$$\begin{aligned} \frac{\partial o_i}{\partial s_k} &= \begin{cases} \frac{e^{s_i}}{\sum_{j=1}^{K} e^{s_j}} - \left(\frac{e^{s_i}}{\sum_{j=1}^{K} e^{s_j}}\right)^2 & i \neq k \\ -\frac{e^{s_i} e^{s_k}}{(\sum_{j=1}^{K} e^{s_j})^2} & i = k \end{cases} \\ &= \begin{cases} o_i(1 - o_i) & i \neq k \\ -o_i o_k & i = k \end{cases} \end{aligned} \tag{6.10}$$

$$\begin{aligned} \frac{\partial E}{\partial s_i} &= \sum_{k}^{K} \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial s_i} \\ &= \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial s_i} - \sum_{k \neq i} \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial s_i} \\ &= o'_i(1 - o_i) + \sum_{k \neq i} o'_k o_i \\ &= o' t_i + o_i \sum_{k} o'_k \\ &= o_i - o'_i \end{aligned} \tag{6.11}$$

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

$$\begin{aligned} \frac{\partial E}{\partial w_{ji}} &= \sum_{i} \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial w_{ji}} \\ &= (o_i - o'_i) h_j \end{aligned} \tag{6.12}$$

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 chapter 4.3 (??).

The next step is to perform the reversed flatten layer, so that the error can be backpropagated further to maxpooling layer. Backpropagating through 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 maxpooling computation. All other neurons get zero gradient.

The gradient prepared in such a way is ready to be backpropagated through convolutional layers. The backpropagation for convolutional layer can be applied as described in chapter 4, however equation 4.12 changes to:

$$\Delta w_{ij} = \lambda z_j^{(l-1)} \sum_{k=1}^{m^{(l)}} \delta_k^{(l)} \tag{6.13}$$

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 second one.

6.1. Implementation and test results

The Convolutional Neural Network may differ in 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 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.

$$R = \begin{bmatrix} r_{11} & r_{21} & \cdots & \cdots & r_{k1} \\ r_{12} & r_{22} & \cdots & \cdots & r_{k2} \\ r_{13} & r_{23} & \cdots & \cdots & r_{k3} \\ r_{14} & r_{24} & \cdots & \cdots & r_{k4} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{1k} & r_{2k} & \cdots & \cdots & r_{kk} \end{bmatrix} G = \begin{bmatrix} g_{11} & g_{21} & \cdots & \cdots & g_{k1} \\ g_{12} & g_{22} & \cdots & \cdots & g_{k2} \\ g_{13} & g_{23} & \cdots & \cdots & g_{k3} \\ g_{14} & g_{24} & \cdots & \cdots & g_{k4} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ g_{1k} & g_{2k} & \cdots & \cdots & g_{kk} \end{bmatrix} B = \begin{bmatrix} b_{11} & b_{21} & \cdots & \cdots & b_{k1} \\ b_{12} & b_{22} & \cdots & \cdots & b_{k2} \\ b_{13} & b_{23} & \cdots & \cdots & b_{k3} \\ b_{14} & b_{24} & \cdots & \cdots & b_{k4} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{1k} & b_{2k} & \cdots & \cdots & b_{kk} \end{bmatrix} \quad (6.14)$$

In convolutional layer each of those matrices were 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 first two layers we obtain a matrix with dimensions $= (1, 32, 76, 76)$.

The 3rd and the 5th layer are maxpooling layers, which purpose is 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 (CNN) is a dense layer. It's just a simple MLP layer, as described in chapter(??). It requires the one-dimensional input - the output of convolutional layers has to be converted into a 1D feature vector. This operation is called flattening and is performed with the flatten 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 number of units equals to the number of classes (amount 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 to classify 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 implementation 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 (maxpooling)is applied, the total number of parameters that need to be trained is equal to $1,25 * 10^9$.

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.

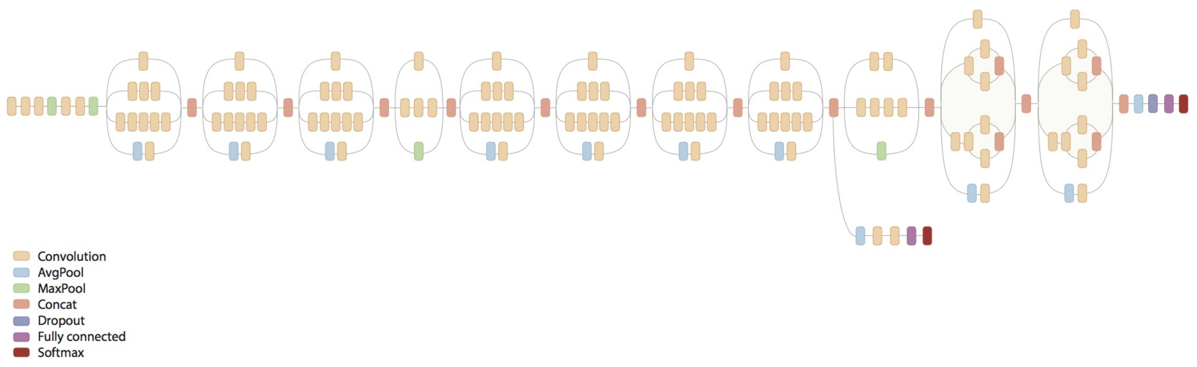


Figure 6.2: Google Inception v3 model architecture

This model structure is much more complicated than the general convolutional network architecture. Instead of applying all convolutional, maxpooling and dense layers one by one, some of the layers consist of the composition of those operation. The model was trained on 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 was no facial images in its training data set.

6.2. Test results

Algorithm was implemented in Python 3.4 with usage of numpy and tensorflow libraries.

The tensofflow library provides us with the function that returns the outputs from a different layers. We take the last layer with 2048 neurons and connect it to the fully connected layer, creating a new output with amount of neurons corresponding to the amount of individuals in the dataset. To train only a single layer, we specify the list of trainable variables in the training operation. The network will train only the parameters that connects 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.

The test was performed on 10 individuals with the biggest amount of images from Labeled Faces in Wild database.

Table 6.1: 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

Data was randomly separated into three sets:

– 75% training set

– 5% validation set

– 20% testing set

The algorithm implements 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).

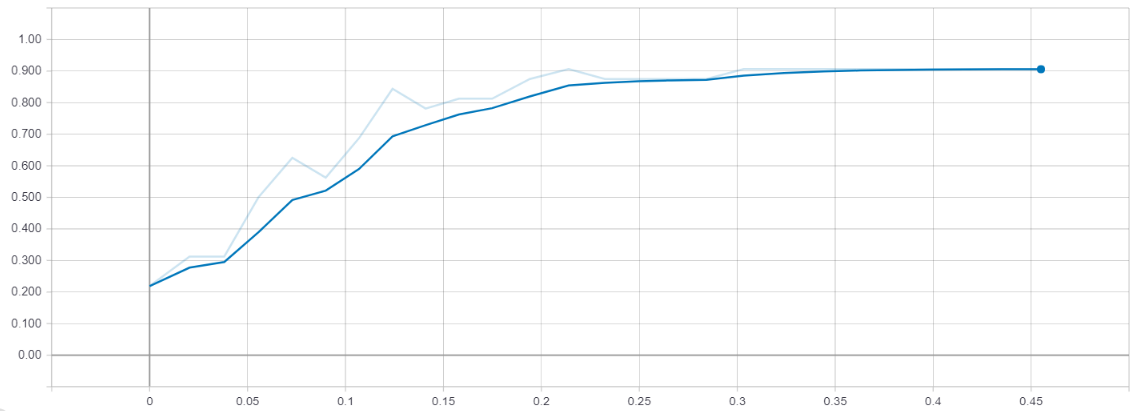


Figure 6.3: Training accuracy

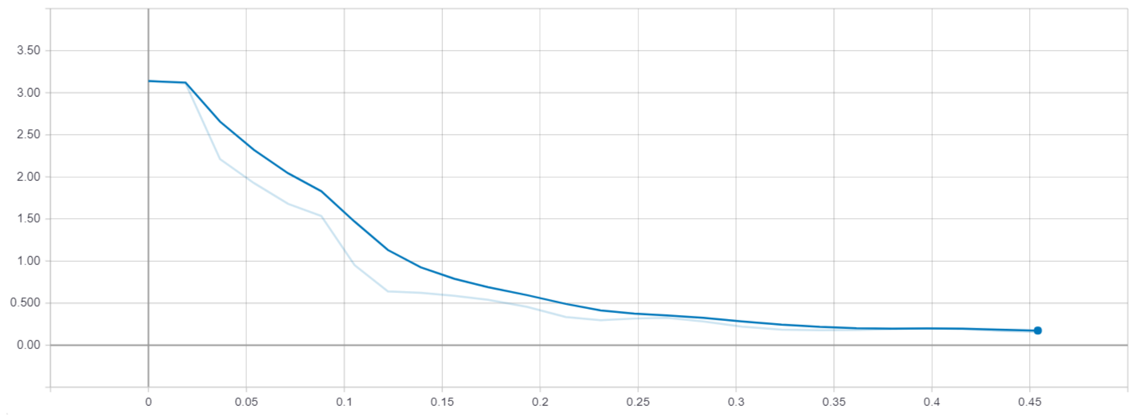


Figure 6.4: Validation loss

Taking into account that these results were obtained via training only last layer of the network, we can consider them better than expected. The training accuracy reached 80%.

The recognition rate obtained from testing data set was 76.5%.

7. Support Vectors Machine

7.1. 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 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.

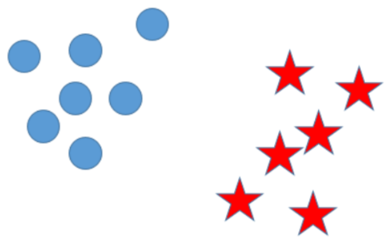


Figure 7.1: Linearly separable dataset and its hyperplane

The hyperplane is a set of points satisfying equation 4.1

$$f(x) = w^T x + b = 0 \tag{7.1}$$

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.

$$w = \sum_{i=1}^{N} \alpha_i x_i \tag{7.2}$$

Then:

$$f(x) = \sum_{i=1}^{N} \alpha_i x_i^T x + b \tag{7.3}$$

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.

7.1.1. The geometric margin

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

$$m(f) = \frac{1}{2}\hat{\mathbf{w}}^T(x_+ - x_-) \tag{7.4}$$

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

$$f(x_+) = w^T x_+ + b = af(x_-) = w^T x_- + b = -a \tag{7.5}$$

where $a > 0$

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

$$m(f) = \frac{1}{||w||} \tag{7.6}$$

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

$$\begin{gathered}\min_{w,b} \frac{1}{2}||w||^2 \\ s.t.\forall i, y_i(w^T x_i + b) \geq 1\end{gathered} \tag{7.7}$$

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

$$\begin{gathered}\min_{w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i \\ s.t.\forall i, y_i(w^T x_i + b) \geq 1 - \xi_i\end{gathered} \tag{7.8}$$

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 ($\xi_i > 1$).

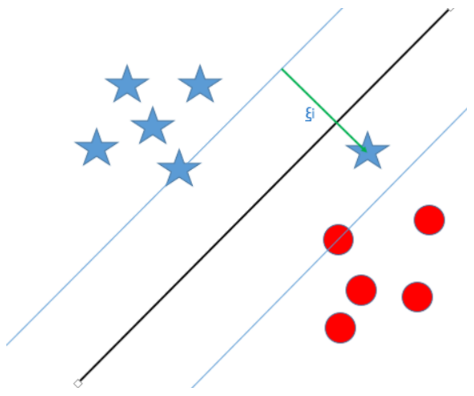


Figure 7.2: Illustration of ξ value

The parameter C is 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 the (?) equation with previously optimized parameters.

$$h_{w,b}(x) = g(w^T x + b) \tag{7.9}$$

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

7.1.2. Non linear separable dataset

The described approach works only for linearly separable data set. Nevertheless, 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?

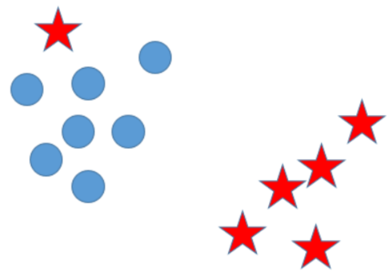


Figure 7.3: Nonlinearly separable dataset and its hyperplane

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 a higher dimensional space, called the feature space.

Rather than applying SVMs using the original input attributes x, we learn 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 data set. The similarity function is also called a kernel.

$$\begin{aligned} f_{i1} &= K(x_i, x_1) \\ f_{i2} &= K(x_i, x_2) \\ &\vdots \\ f_{iN} &= K(x_i, x_N) \end{aligned} \tag{7.10}$$

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

$$\vec{f_i} = [f_{i1}, f_{i2}, \cdots, f_{iN}] \tag{7.11}$$

In the feature space, F expression 4.3 takes the form:

$$f(x) = \sum_{i=1}^{N} \alpha_i \phi(x_i)^T \phi(x) + b \tag{7.12}$$

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

$$K(x, z) = \phi(x)^T \phi(z) \tag{7.13}$$

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

$$f(x) = \sum_{i=1}^{N} \alpha_i K(x, x_i) + b \tag{7.14}$$

Two most widely used kernels are:

1. Gaussian kernel
$$K(x, x_i) = exp(-\frac{||x - x_i||^2}{2\sigma^2}) \tag{7.15}$$
where σ is a parameter that controls the width of Gaussian
2. Polynomial kernel
$$K(x, x_i) = [(x \cdot x_i + 1]^d \tag{7.16}$$
with degree d

7.1.3. 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.

7.1.4. 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 to one person, it is relatively easy to perform recognition. Specifying the threshold, the distance between input and given SVM models are computed as follows:

$$d(x) = \frac{\acute{L} \sum_{i=1}^{l} \alpha_i y_i x_i \cdot x + b}{|| \sum_{i=1}^{l} \alpha_i y_i x_i ||} \tag{7.17}$$

We can notice that numerator is equal to right side of equation (12), hence the sign of d is the classification result for x, and $||d||$ is the distance from x to the hyperplane.

The class label of input picture is computed as follows:

$$y = \begin{cases} n \text{ if } d_n(x) > t \\ \text{not recognized if } d_n(x) \leq t \end{cases}$$

where $d_n(x) = max(d_{i}{}_{i=1}^{q})$

7.2. Implementation and test results

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

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

The tests were performed on two different databases:

– Chicago Face Database - pictures captured in controlled environment

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

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

7.3. Tests on pictures captured in controlled environment

The database was divided into training and testing samples. The training was performed using 3 pictures of the individual, one picture was used to test the trained model.

7.3.1. 20 individuals

The first experiment was performed on 20 individuals from 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 (equation(???), which determines how fast the similarity metric decreases to 0 as the examples are further apart.

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

$$exp(-\gamma * ||x - x_i||^2) \tag{7.18}$$

where:

$$\gamma = \frac{1}{2\sigma^2} \tag{7.19}$$

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

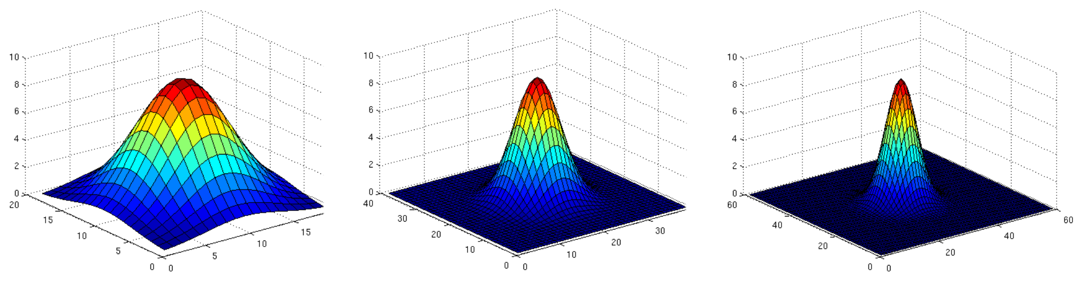


Figure 7.4: Gaussian kernel visualization

The figure(??) presents the SVM training accuracy with varying γ and C parameter:

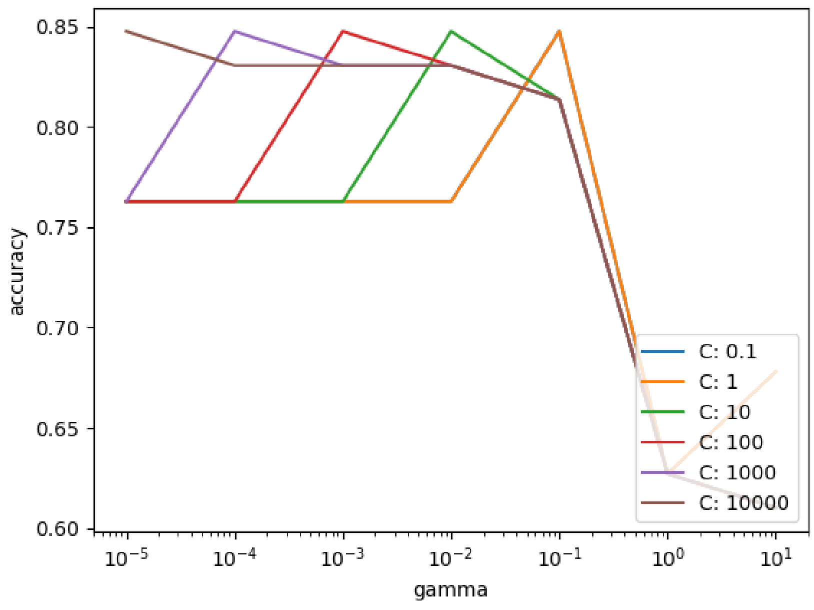


Figure 7.5: Training score with varying gamma and C parameter for 20 individuals

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 SVM algorithm reached 80% of recognition rate on the testing samples.

The same experiment was performed with usage of polynomial kernel. The influence of polynomial degree was examined and is presented on figure (??).

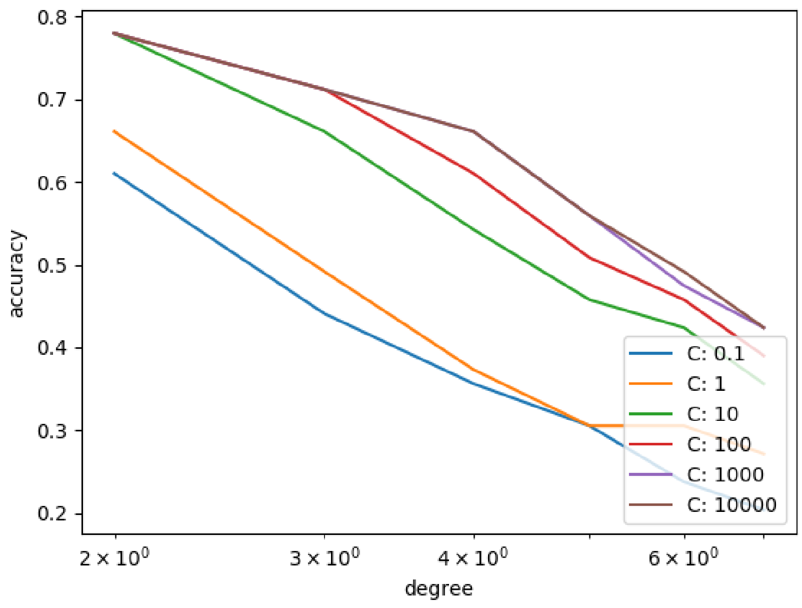


Figure 7.6: Training score with varying polynomial degree and C parameter for 20 individuals

On Figure(7.6 (??)) we can see that our data can be nicely separated with polynomial of second degree. The recognition rate was tested on SVM trained with $C = 10$ and the degree = 2.

Obtained result is 70%, which is 10 percentage points less than result of SVM trained with Gaussian kernel.

7.3.2. 40 individuals

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

The results are presented below.

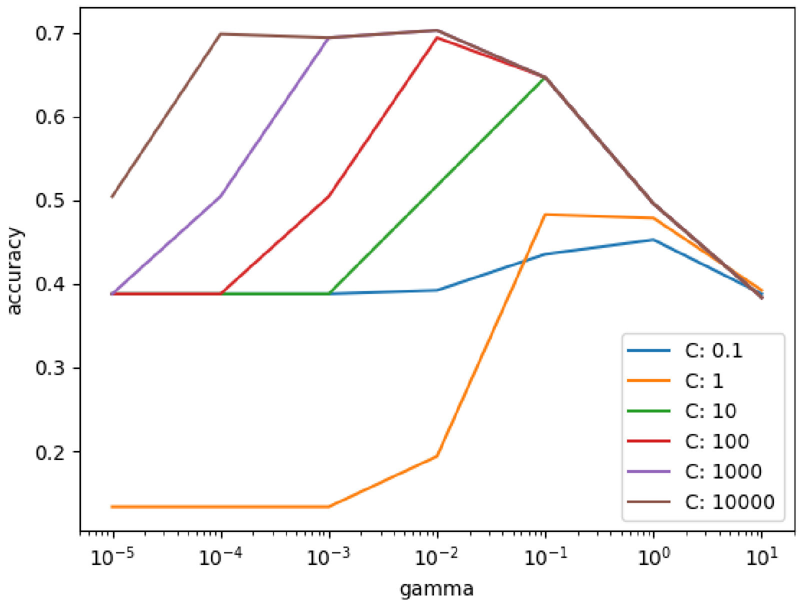


Figure 7.7: Training score with varying gamma and C parameter for 40 individuals

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 SVM algorithm reached 60% of recognition rate on the testing samples.

The same experiment was performed with usage of polynomial kernel. The influence of polynomial degree was examined and is presented on figure (??).

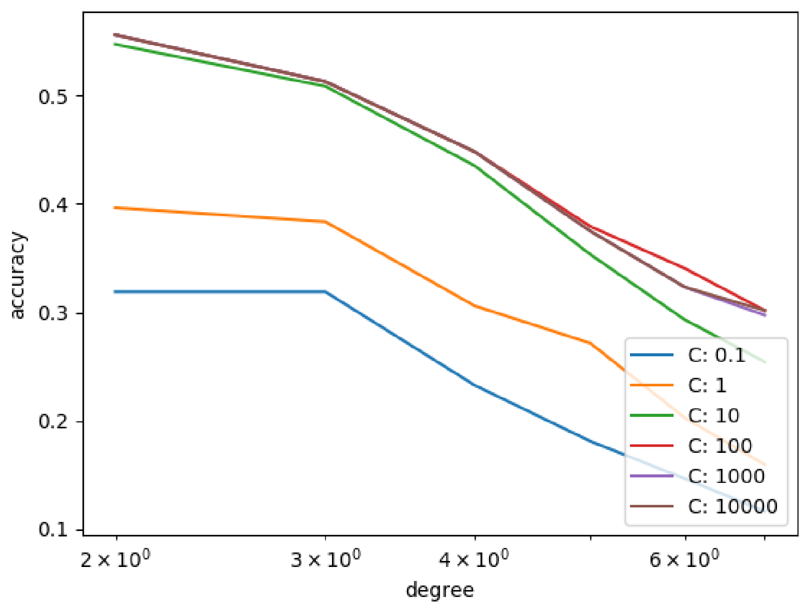


Figure 7.8: Training score with varying polynomial degree and C parameter for 40 individuals

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

7.4. Tests on pictures captured in uncontrolled environment

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

Test was performed on 20 individuals with 30 pictures each. The training set for each person consists of 27 samples, 3 pictures were used to test the model.

The results are presented below.

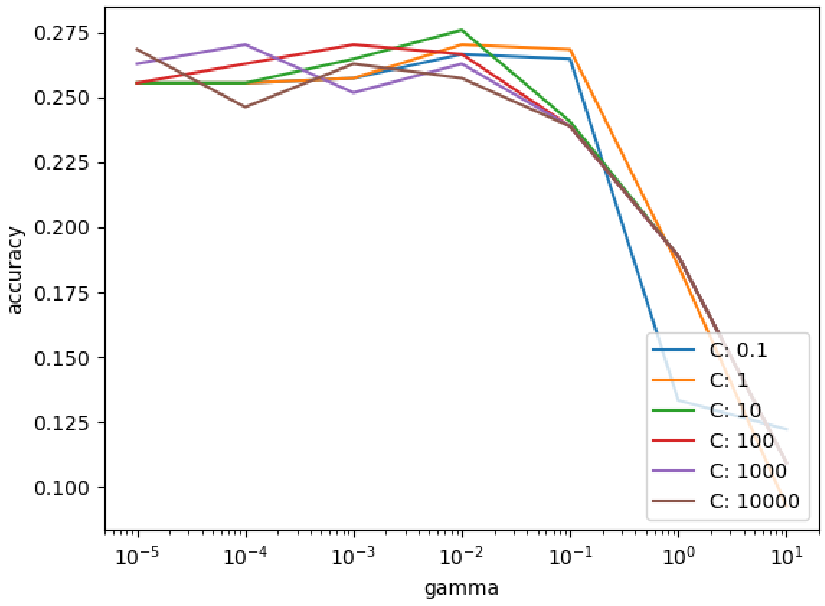


Figure 7.9: Training score with varying gamma and C parameter for 20 individuals

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

The same experiment was performed with usage of polynomial kernel. The influence of polynomial degree was examined and is presented on figure (???).

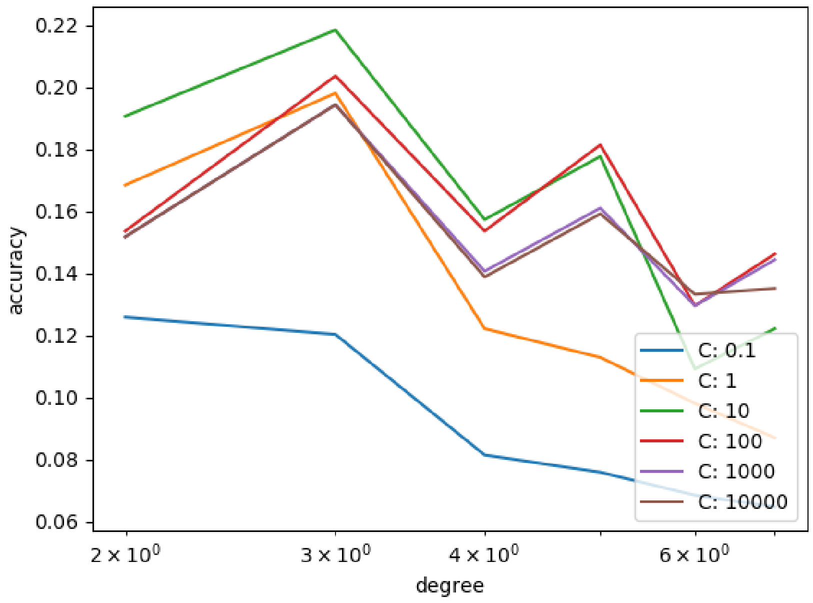


Figure 7.10: Training score with varying polynomial degree and C parameter for 20 individuals

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

7.5. SVM summary

To compare obtained results a tabular summary of the recognition rate for SVM algorithm with different kernels on each database is presented (Table 7.1).

Table 7.1: 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 controlled environment and small amount of data. Even though SVM has an advantage of small computing time, the results do not meet the expectations of good facial recognition system.

8. Summary

Table 8.1: A tabular summary of the recognition rate for each algorithm

Algorithm	CFD 20	CFD 40	LFW 20
PCA	114	173	10
PCA-MLP	70%	43.5%	10
SVM	76.5%	50%	10
CNN	76.5%	50%	10

Table 8.2: A tabular summary of the execution time for each algorithm

Algorithm	CFD 20	CFD 40	LFW
PCA	114	173	10
PCA-MLP	70%	43.5%	10
SVM	76.5%	50%	10
CNN	76.5%	50%	10