UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SUPERVISED MACHINE LEARNING TECHNIQUES IN FACE RECOGNITION

Diploma Thesis

David Georgantas

Supervisor: Panagiota Tsompanopoulou

Volos 2021

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SUPERVISED MACHINE LEARNING TECHNIQUES IN FACE RECOGNITION

Diploma Thesis

David Georgantas

Supervisor: Panagiota Tsompanopoulou

Volos 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΕΧΝΙΚΕΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΜΕ ΕΠΙΒΛΕΨΗ ΣΤΗΝ ΑΝΑΓΝΩΡΙΣΗ ΠΡΟΣΩΠΟΥ

Διπλωματική Εργασία

Δαυίδ Γεωργαντάς

Επιβλέπουσα: Παναγιώτα Τσομπανοπούλου

Βόλος 2021

Approved by the Examination Committee:

Supervisor **Panagiota Tsompanopoulou**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Gerasimos Potamianos**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Michael Vassilakopoulos**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

I would like to thank Prof. Panagiota Tsompanopoulou for her supervision and mentoring. Her guidance and technical support carried me through all stages of accomplishing this project. I would also like to thank my committee members for their helpful and improving comments and suggestions.

Last but not least, special thanks to my family for supporting me all these years and my friends for the great moments in Volos.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

David Georgantas

Abstract

In this Thesis, the fundamentals , the implementations and the results of three different supervised machine learning approaches in face recognition problem are provided. The project is focused on the comparison among Support Vector Machines, Artificial Neural Networks and Convolutional Neural Networks. After the algorithms analysis and implementation, the test results of these three techniques will be presented and compared.

Περίληψη

Σε αυτή τη διπλωματική εργασία, παρουσιάζονται οι βασικές αρχές, οι εφαρμογές και τα αποτελέσματα τριών διαφορετικών μεθόδων εποπτευόμενης μηχανικής μάθησης στο πρόβλημα της αναγνώρισης προσώπου. Η εργασία επικεντρώνεται στη σύγκριση μεταξύ των Μηχανών Διανυσματικής Υποστήριξης, Τεχνητών Νευρωνικών Δικτύων και Συνελικτικών Νευρωνικών Δικτύων. Μετά την ανάλυση και την εφαρμογή των παραπάνω αλγορίθμων, τα αποτελέσματα των δοκιμών αυτών των τριών μεθόδων, θα παρουσιαστούν και θα συγκριθούν.

Table of contents

List of figures

List of tables

Abbreviations

SVM	Support Vector Machines
ANN	Artificial Neural Networks
CNN	Convolutional Neural Networks
LFW	Labeled Faces In the Wild dataset
PCA	Principal Component Analysis
OFD	Olivetti Faces Dataset

Chapter 1

Introduction

Face recognition systems have the potential to verify or identify individuals from images. It is a research area that is combining many other techniques such as image processing and pattern recognition. In this thesis, we will try to examine face recognition as an image classification problem. Three supervised machine learning techniques were used to perform image classification after some important steps of image processing before the classification phase. Support Vector Machines, Artificial Neural Networks and Convolutional Neural Networks are the used techniques. After the machine learning models implementations, there are several loss and accuracy curves presented in order to show the regularization of each model. Two datasets with specific differences that we will discuss later, have been used to train and test the models. The test results are presented analytically, in order to make as capable of comparing these three supervised machine learning models for the face recognition task.

1.1 Thesis Outline

Chapter 2 includes determinations of face recognition types and the face recognition process step by step. In Chapter 3, the fundamentals of these three supervised machine learning techninques are presented, aiming at making us familiar with their operation. Therefore, in Chapter 4, the used datasets for training and testing the algorithms, are described. Specifically, the differences of these two datasets and the way that we split the data for training and testing phase is presented. In chapter 5, the implementations of the models are presented, while there is a comparison of the performances in test sets. In Chapter 6, there is a conclusion of the thesis that takes into account each model's performance, as it was provided in Chapter

5.

Chapter 2

Face Recognition

Face recognition is a process that humans accomplish on a regular basis and with ease. Processing of digital pictures in a variety of applications, such as biometric authentication, surveillance, human-computer interaction, and multimedia management, has become increasingly popular since it is considered not only a fast and low cost desktop but also because embedded computing systems become more widely available. Takeo Kanade developed the first automatic face recognition system in his Ph.D. thesis work in 1973. Until Sirovich and Kirby's work[13] on a low-dimensional face representation obtained using the Karhunen–Loeve transform or Principal Component Analysis, there was an inactivity in automatic face recognition. Turk and Pentland's pioneering work on Eigenface [42] has reactivated face recognition research[14]. The Fisherface technique[15, 16], which used Linear Discriminant Analysis (LDA) after a PCA phase to obtain better accuracy, is another important achievment in face recognition. Since the Eigenface technique was introduced[14], face recognition technology has improved substantially, in restricted conditions, such as when lighting, position, stand-off, facial wear, and facial expression can be controlled, automated face recognition can outperform human recognition, especially when the dataset includes a high number of images. However, when face pictures are collected in uncontrolled situations, automated face identification confronts several obstacles. In some occasions, pictures of the same individual of the dataset, may can not be recognized even by humans, because of their no matching. As an example there may be some face pictures of the same person that are captured in a long time range.

The process of identifying a face in a picture is divided into two stages, face detection and face recognition :

- Face detection is about detecting which pixels represent the face in a picture.

- Face recognition is about recognizing in which person this face belongs.

Face Detection is a similar problem to face recognition but we will not discuss it further in this thesis, because our datasets already include face images that do not need to be cropped.

Face recognition may be used in two separate applications, face verification and face identification :

- Face verification is a face recognition task that its goal is to recognize if two images belong to the same person.

- Face identification is a face recognition task that its goal is to recognize in which person of the dataset, this image belongs.

We could say that face verification applications reply ”Yes” or ”No”, while face identification applications answer with the identity of an individual of the dataset. In verification problem we have to determine whether there is a pairs match between two images, otherwise in face identification we have to determine the person that this face belongs to.

Face recognition has been examined with different kind of machine learning algorithms. These algorithms are separated in supervised and unsupervised learning methods. There are some differences between these two categories, such as the way that these algorithms learn and the way they make predictions. Supervised learning algorithms need labeled data to be trained, for example, in face recognition, labels are names or class target numbers. Unsupervised machine learning algorithms are trained with data with no previous classification and they are mostly used for clustering, dimensional reduction and feature extraction.

Before training and testing some supervised learning algorithms, some preprocessing is necessary. So the face recognition process is not only about classifying a testing image. After the face detection phase, face normalization and feature extraction must be applied in order to produce trainable data.

- Face normalization is a classic process in image processing. It changes the input’s values in order to be in a specific range of real numbers. The values of an image, are pixel intensity values.

- Feature extraction is like a dimensionality reduction technique. Its goal is to represent the input with less data but in a way that the significant information is highly represented.

Chapter 3

Background

In this chapter, we will try to analyze the algorithms and the used techniques in this Thesis. We start with the Principal Components Analysis (PCA) and discuss the reason why this method can be used when we examine a large dimensional dataset and how dimensional reduction through PCA can be achieved. Then we analyze a widely known machine learning model called Support Vector Machines. Classification examples with linear and non-linear separable datasets will give the evidence of how actually this algorithm works. Artificial Neural Networks come at the end of this chapter. We try to examine how the whole mechanism works for small architectures with a few neurons and generalize for bigger architectures with more weights. We also consider about Convolutional Neural Networks and how Convolutional and Pooling layers work and influence feature extraction.

3.1 Principal Component Analysis

Principal Component Analysis is an unsupervised machine learning technique. This means that this method, is not examining the labels of the data. In several fields, large datasets are becoming more common. To interpret large datasets, approaches must dramatically reduce their dimensionality in an interpretable manner, preserving the majority of the data's content. For this goal, a variety of approaches have been developed, but principal component analysis (PCA) is one of the oldest and most commonly utilized. Its goal is to minimize a dataset's dimensionality while keeping as much statistical information as possible. This means that keeping as much variability as possible entails identifying new variables that are linear functions of the original dataset's variables, that maximize variance in a sequential manner and

are unrelated to one another[17]. Naturally, performing dimensional reduction in a data collection means losing some accuracy, but instead simplicity is acquired which is considered quite significant. The steps of PCA are :

1. First step of PCA is to standardize the input variables. If some variables have large differences in range with some another, this may be an issue. For example, a variable with a range of 0 to 100 will outperform a variable with a range of 0 to 1. So, transformating or rescaling the data in scales that are comparable will overcome this issue[18].

2. Second step is Covariance Matrix Computation. This way, we can examine if our data are correlated. The Covariance matrix is a n x n symmetric one , which includes the covariances of all the possible pairs between the variables in the dataset[18].

3. Third step is the computation of the eigenvectors and the eigenvalues from the covariance matrix, in order to identify the principal components. Principal components are new features that are created by combining the basic variables in a linear way. The new variables are uncorrelated as a result of these combinations, and the majority of the useful information in the basic variables is squeezed into the first components. Then we have to decide how many principal components we will keep in order to transform our data in their direction. This transformation is a result of the product between the original data matrix and the matrix of eigenvectors corresponding to top k largest eigenvalues. In Figure 3.1, we see how the squeeze of the information is happening in a 10-dimensional data problem when PCA is applied.

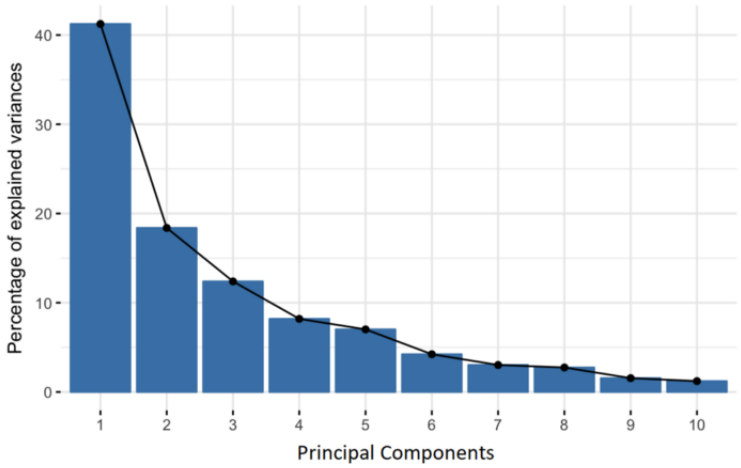


Figure 3.1: Percentage of explained variances in 10 Principal Components[1]

In disciplines such as facial recognition, computer vision, and image compression, PCA is primarily utilized as a dimensionality reduction technique. It's also used in finance, data mining, biology, psychology, and other fields to uncover patterns in high-dimensional data.

3.2 Support Vector Machines

SVMs are categorized as decision boundary classifiers and they form a widely known and used supervised machine learning model. They execute pattern recognition between two classes by determining a decision surface with the greatest distance between the training set's closest points, known as support vectors. Suppose We begin with a set of training samples, where each of this set's points, belongs to one of the two label-identified classes. SVM's purpose is to find a way to separate these data correctly.

A hyperplane is a collection of points that satisfy an equation. A hyperplane or commonly an equation like this can be presented as :

$$f(x) = w^T x + b = 0 \tag{3.1}$$

where the weight vector is w and bias is represented by b. Assume that the weight vector is a linear combination of training samples.

$$w = \sum_{i=1}^{N} a_i x_i \tag{3.2}$$

Then :

$$f(x) = \sum_{i=1}^{N} a_i x_i^T x + b \tag{3.3}$$

The hyperplane divides space into two parts. Each part of the space represents a class. However, if we evaluate our training samples to function f, the sign of the result f(x) for a random point x, will demonstrate in which class this point is classificated.

The simple problem we examined before, may have many solutions. Many solutions means many hyperplanes that classify the training data the right way. But, the goal is to find the optimal solution. The optimal solution is a hyperplane that has the maximum margin between the data points of the two classes. In order to find this hyperplane, we take advantage of support vectors which are some data points close to the hyperplane. Support vectors influence the position and orientation of the hyperplane and using them we maximize the margin[19].

3.2.1 Non-linear separable dataset

In a real-world scenario, getting a precise separate line splitting the data within the space is unlikely. And we may have a non-linear decision boundary. We might have a hyperplane that perfectly separates the data, but this isn't always ideal if the data contains noise. It is preferable for the smooth boundary to overlook a few data samples rather than being curved or looped around outliers. A technique called Soft Margin is used to face the above issue. With this technique, some data points are allowed to be on the wrong side of the hyperplane for a small distance called s_k.The points lying on the wrong side of the hyperplane are called slack variables, and they might be very large values. In these cases, we introduce the Lagrangian variable in order to penalize the large slacks [20]. As the Lagrangian parameter is getting reduced, we allow more data samples to lie on wrong side of hyperplane in order to get a smoother decision boundary[21]. In 3.2 Figure we see a linear separable dataset on the left side, and a non-linear separable dataset on the right side

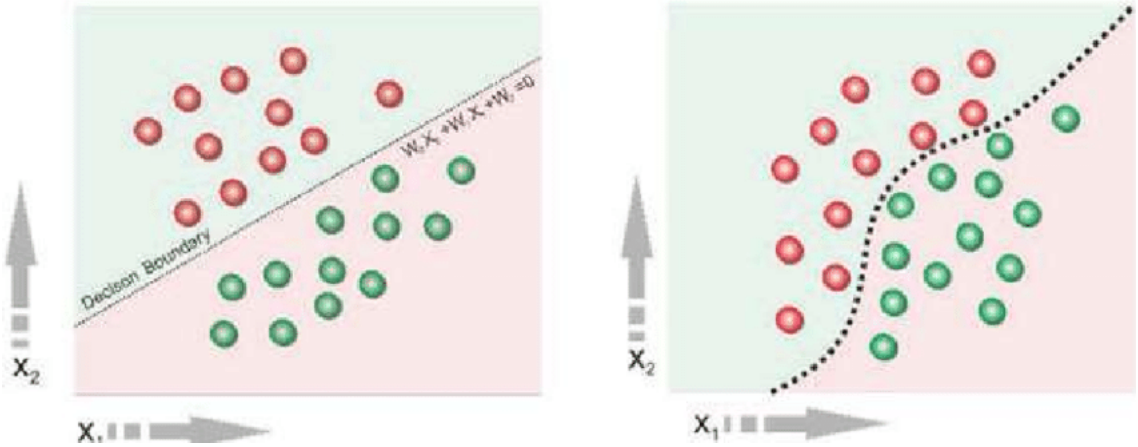


Figure 3.2: Differences between linear and non-linear separable datasets[2]

Another technique to deal with non-linear separable data is the Kernel trick. The Kernel function is used in order to map our data with some other data in a feature space that our classes are linear separable. Mapping our data in another feature space might make our linear classifiers capable to classificate with a better accuracy and provide us a better model. The Kernel function takes as inputs the original dimensional space and returns the dot product of the higher dimensional transformed vectors[22]. Some of the most used kernel functions are:

1. Gaussian Radial Basis Function

$$K(x, x_i) = exp(-\frac{|x - x_i|}{2\sigma^2}) \tag{3.4}$$

where σ is a parameter that affects the gaussian distribution's width. Due to its resemblance to the Gaussian distribution, RBF kernels are the most generic form of kernelization and one of the most extensively used kernels.

2. Polynomial Kernel

$$K(x, x_i) = (x \cdot x_i + 1)^d \tag{3.5}$$

where d is the degree of the polynomial kernel.

In Figure 3.3 below, we can check how gaussian pdf is behaving for varying σ values.

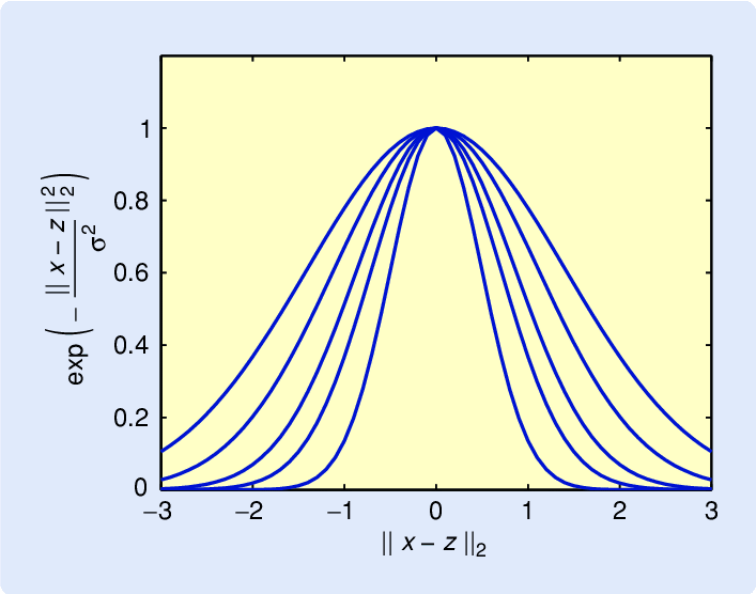


Figure 3.3: Gaussian RBF kernel for different σ values[3]

3.3 Artificial Neural Networks

Artificial Neural Networks are one of the most powerful ways for solving the face recognition problem. Haykin [23] provides a fair explanation of ANN, defining it as a parallel mixture of basic processing units that can learn from their environment and acquire knowledge from it. Keep track of what you've learned by looking at the connections you've made. The Artificial Neural Network concept was influenced by biological systems. The brain, in particular, has neural networks. Machine learning is accomplished using neural networks, in which a computer learns to do a task by examining training instances. Typically, the examples have been labeled by hand ahead of time. For example, an object recognition system could

be fed thousands of tagged photographs of vehicles, houses, coffee cups, and other objects, and it would look for visual patterns in the images that correspond to specific labels[24].

3.3.1 Artificial Neural Networks architecture

Neural Networks are multi-input, multi-output systems made up of artificial neurons. A Neural Network's principal work is to convert inputs into useful outputs. Typically, a neural network has an input and output layer, as well as one or more hidden layers. All of the neurons in a Neural Network influence each other, thus they are all connected. The network can recognize and observe every facet of the dataset in question, as well as how the various pieces of data may or may not be related to one another. This way, neural networks can detect incredibly complicated patterns in massive amounts of data.

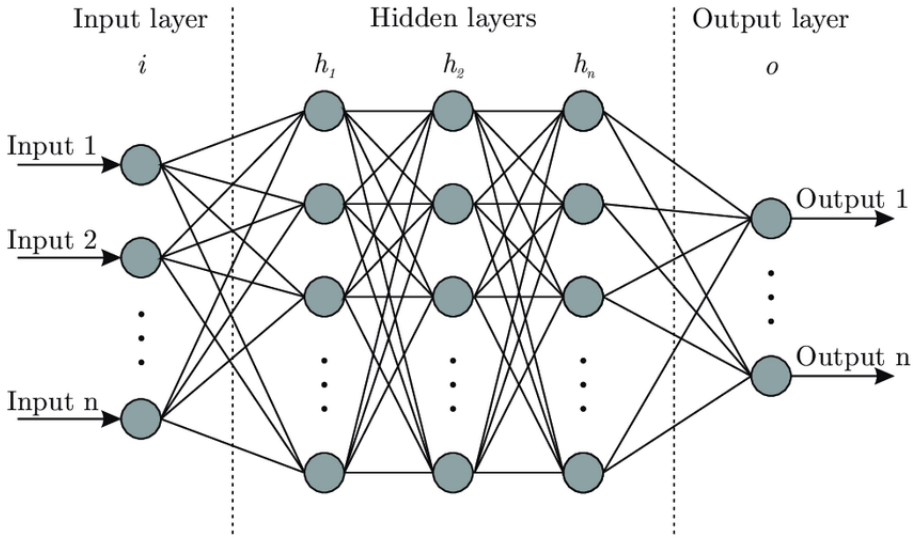


Figure 3.4: Neural network architecture[4]

In the above 3.4 Figure, there is a visualized neural network architecture, so now we have an idea about how it looks optically. They collect data from a remote source or from other nodes. Each node is connected to a node in the next layer, and each connection has a different weight. A neuron is given a weight based on how important it is in comparison to other inputs. The hidden layers of a Neural Network are where the majority of the computation happens. Thus, the hidden layer conducts the calculations necessary to get a result, using data that were propagated from the input layer[25].

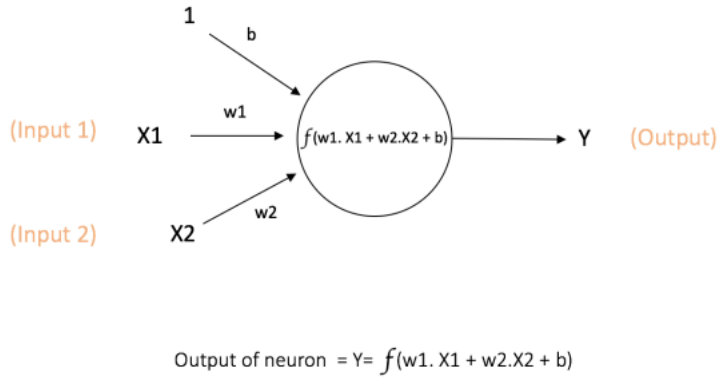


Figure 3.5: A single neuron with two inputs, bias and an activation function

In figure 2.5 above [26], an activation function influences the output. What actually this function is responsible for, is non-linearity and whether or not this neuron will be activated or how active it shall be. The most widely used activation functions are below:

- Sigmoid : shrinks the input to a value between 0 and 1,

$$s(x) = \frac{1}{1 + e^{-x}} \tag{3.6}$$

- Hyperbolic tangent : Shrinks the input to a value between -1 and 1,

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.7}$$

- Rectified Linear Unit (ReLU) : Thresholds the value that takes as input

$$R(x) = max(0, x) \tag{3.8}$$

The ReLU function is thought to perform better than the other activation functions, according to recent papers. Its simplicity minimizes computer cost because it avoids complex operations like exponentials. Another significant advantage of ReLU is that its gradient is not saturated, which substantially accelerates stochastic gradient descent convergence as compared to sigmoid / tanh functions[27].

3.3.2 Feed-forward process

The basic formula for the forward pass of a neural network is:

$$\vec{o} = f(\vec{w} \cdot \vec{x} + b) \tag{3.9}$$

where f is the activation function, w is the vector including the weights, b is bias and is the input features and o is the output.

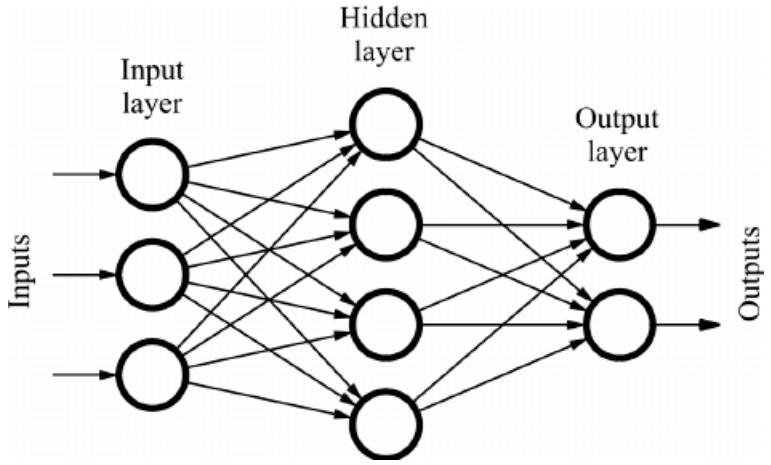


Figure 3.6: Feed-forward process[5]

Figure 3.6, is presenting an artificial neural network topology, which is working through the feed-forward process. The projection of each single neuron , is only forward and never backward. Additionally, no connections exist between neurons in the same layer or between layers that are not nearby.

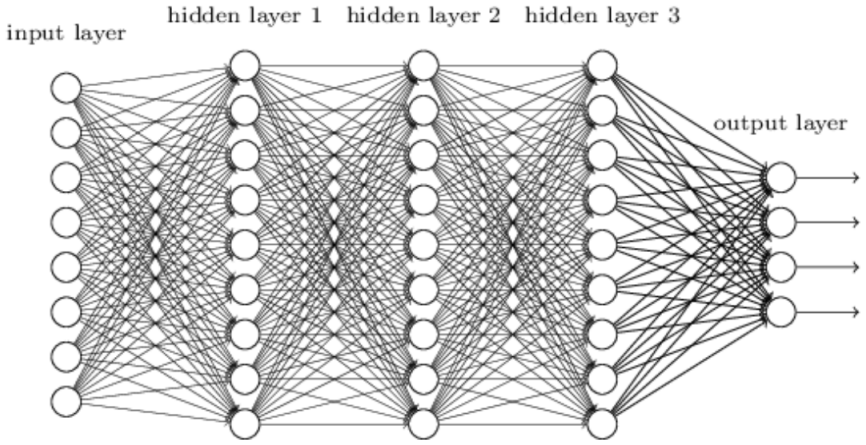


Figure 3.7: Deep neural network architecture example[6]

A Deep Neural Network is a neural network with more than one hidden layer. This sort of ANN appears to be an excellent choice for the creation of an object recognition system because it is capable of processing very high-dimensional data.

3.3.3 Backpropagation

Backpropagation algorithm employs supervised learning. Which requires providing the algorithm with labeled samples that the network must be capable to compute, then calculating the error. The purpose of the backpropagation algorithm is to change the weights in order to minimise this error until the Artificial Neural Network learns the training data. The network weights are changed along the negative of the performance function's gradient in standard backpropagation, which is a gradient descent technique. A solution to the learning problem is a weighted combination that minimizes the error function. There are a lot of cost functions to use. A simple one , is the calculation of Euclidean distance between predicted and expected outputs.

$$E_2(o, o') = \frac{||o - o'||^2}{2} \tag{3.10}$$

where o is predicted values and o' the expected ones.

Mean Squared Error can also be used to calculate the error:

$$E_2(o, o') = \frac{\sum_{k=1}^{K}(o_i - o_i')^2}{K} \tag{3.11}$$

where K is the vector's length.

Due to the higher convergence of the gradient, cross-entropy error is frequently preferred over classification error or mean squared error when employing a neural network for classification and prediction. The cross-entropy error function is :

$$E_3(o, o') = -\sum_{k=1}^{K}(o_i' ln(o_i)) \tag{3.12}$$

We must deal with each weight in the network one at a time since the purpose of backpropagation is to minimize the error E, which is dependent on the network weights. The output of each unit from the forward pass must be stored in order to do backpropagation. With respect to each parameter w_{ij}, the gradients are in the process of being calculated.

1. First step is the calculation of δ_i errors from the outputs.

2. Second step is the determination of δ_i for every hidden layer.

$$\delta_i^{(l)} = \sum_{k=1}^{m^{(l+1)}} (w_{i,k}^{(l+1)})\delta_k^{(l+1)} \tag{3.13}$$

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

3. Calculation of derivatives.

$$\frac{\partial E}{\partial w_{i,j}^{(l)}} = \delta_j^{(l)} y_i^{(l-1)} \tag{3.14}$$

4. Fourth step, the gradient descent is achieved by subtracting the increment from each weight after all partial derivatives have been computed.

$$\Delta w_{ij} = \lambda z_j^{(l-1)} \delta_j^{(l)} \tag{3.15}$$

The learning rate, λ, influences how quickly the network coefficients change.

The backpropagation algorithm is used to determine the necessary corrections after randomly selecting the network's starting weights. The training of neural networks can be broken down into the following steps:

1. Feed-forward process,
2. Backpropagation for every network's layer
3. Weight updates

3.3.4 Multi-layer Perceptron

The multi layer perceptron neural network is the most well-known and often utilized type of neural network. There are at least three layers to it. The distinction between Multi Layer Perceptron and Single Layer Perceptron is that the former can learn non-linear functions, whereas the latter is only good for solving linear issues. Multilayer perceptrons are frequently used in supervised learning situations, where they are trained on a collection of input-output pairs and learn to represent the relationships between the inputs and outputs. MLP works effectively with inputs that are somewhat low in dimension.

3.4 Convolutional Neural Networks

In 1980, a Japanese researcher in the field of Computer Science, Kunihiko Fukushima, published a research project called ”Necognitron: A self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position” [28]. This publication introduced the architecture of Convolutional Neural Networks for the first time. The structure of the network that Fukushima suggested, was a self-organized multilayer neural

network which had the capability to recognize patterns. Fukushima's model laid the foundations because of the presentation of two new types of layers in Convolutional Neural Networks. These two new types were Convolutional layers and downsampling layers. In 1987, Alex Weibel presented Time Delay Neural Networks (TDNN). TDNN were capable to process speech signals,regardless of their duration. In 1990 , a variaton of TDNN was publiced, this network had the ability to compute two dimensional convolutions. This new TDNN presents the idea of applying Convolutional Neural Networks in image processing. In 1989, Yann LeCun applies CNN to handwritten digits recognition, in order to develop a model for mail classification [29]. This method was using backpropagation for training, in order to be able to recognize handwritten digits from U.S mail. Today, the increased computing performance made Convolutional Neural Networks widely used in many applications. Someone could say that CNN are very similar to traditional ANN, which form a group of neurons that is self-optimized during training. Here too, its neuron gets an input and makes a specific process. From the input, unprocessed image matrices, to the output, the prediction of the class, the whole network presents a result function. The last layer includes the cost function connected with the classes, while every process which is applied in ANN, is applied in CNN too.[30][7]

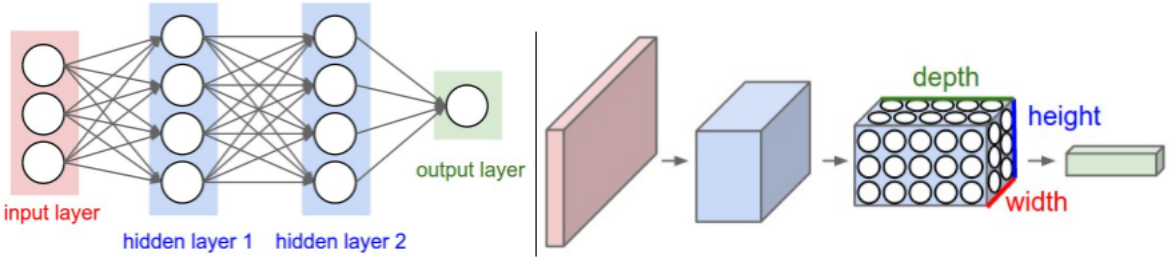


Figure 3.8: ANN and a 3 dimensional CNN[7]

3.4.1 Convolution

CNN have been a successful approach in several real-world applications. Their operation is described in their name, which means that CNN is carrying out a mathematical process, called Convolution. In general form, convolution is the product of two real valued functions. Let's call the two discriminant function f and g. The convolution between f and g is presented from the equation below.

$$(f * g)(x) = \sum_{k=-\infty}^{+\infty} f(k)g(x-k) \tag{3.16}$$

In CNN the first function f is called input,while the second one g, in this equation is called kernel. Some people call the output of this convolution operation is a feature map. Convolution is a widely general idea. We can use it in more than one distances. Specifically, in CNN , most of the times the input is multi dimensional matrices, while kernel is a multi dimensional matrix too, which includes affected values from the training algorithm. Usually, we call this matrices tensors. As we said before, we are using convolution in more than one dimensions. If I is an two dimensional input image , and K is the same dimensional kernel :

$$(I * K)(x) = \sum_{m} \sum_{n} I(m,n)K(i-m, j-n) \tag{3.17}$$

Let input to be a two dimensional matrix. This matrix, is an image that we have to recompute its weights by the sum of the around pixel weights. As we said before,The matrix of weigths is called kernel. Kernel, will stride over the input matrix and compute the multiplication between the kernel weights and the values that the striding window is over them. The previous process, is happening again and again. Kernel is like a convolutional sliding window over the input image[31].

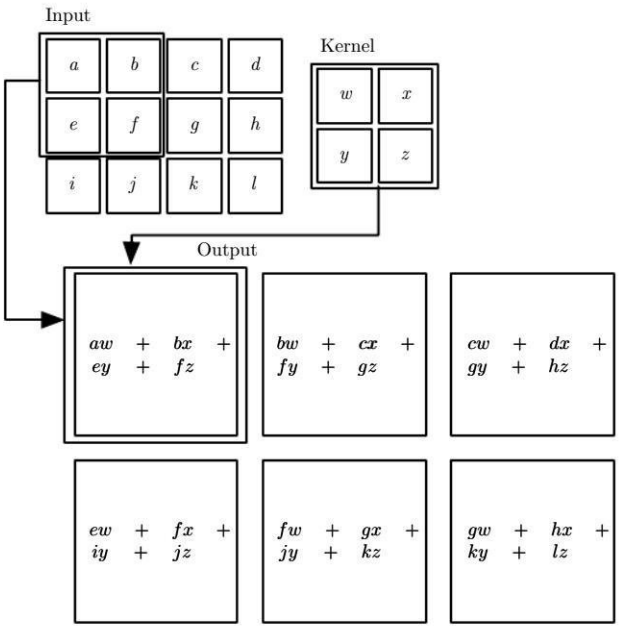


Figure 3.9: Convolution process example[8]

In Figure 3.9, we can see a convolution example of a 3x4 matrix with a 2x2 kernel. The convolution results seem to be according to the general equation we discussed about before. The highlihts of the convolution process present that we have a 2x3 matrix in the output. It is clear now, that the size of the outpout is strongly dependent on the size of the kernel. In the previous example we had 12 (3x4) input values and 6 (2x3) output values. If we had a fully connected layer instead of a convolutional layer, the weight's matrix, would be a 12x4 matrix, which means 48 components. Every output value is the weight's sum of every input value.

3.4.2 Padding

When we have to process images, we must know that there is hidden information in every pixel. When we apply convolution between the input image and kernel, the size of the output matrix is smaller than the input. The convolution process is aims more the central pixels of the image, than pixels which are close to the edges. So, it seems that this way we are loosing some information of the input image. Padding is an answer for these kind of issues. Padding in CNN, is about adding pixels in the input image while the convolution process with the kernel is running. When padding is set to zero, every additional pixel in the perimeter of the image will have the zero value. With the addition of the ”fake” pixels, when kernel is sliding, it will process the edges of our intput, in the center of the image for many times. This way, the information of the output image is not alterated comparatively to the input one. Moreover, padding's use is about extending the image, so we have a bigger image as input[32]

3.4.3 Stride

In CNN there are many hyperparameters that we have to consider, in order to achieve better accuracy. Stride parameter is the one that determines the steps of sampling over the the network's input. So the output image of the convolution process will have less values than the input image if stride is getting bigger. Stride is always an integer and it may be interesting to examine what changes it affects in matrix sizes and in network performances.

3.4.4 Pooling

Pooling layers are responsible for downsampling the information from the feature maps by summing the features in groups. So, when downsampling is performed, the result is dimensional reducted feature maps, while our networks has less data to deal with after that. Our networks has less work to do, because it needs to learn less parameters. Another benefit of pooling is that computational cost may be significantly decreased. There are two widely used types of Pooling in CNN, Max Pooling and Average Pooling.

Max Pooling

In Pooling layers the process is somehow common with the process in Convolutional layers. Actually in Pooling's layer occasion, there is a predefined window, which is sliding over every part of the input, always with respect to the stride parameter. But it is important to notice, that Pooling layers have no parameters, so there is no a filter. In figure 3.10, there is a pooling example with predefined kernel and stride.

Figure 3.10: Max Pooling with 2x2 kernel and stride[9]

Average Pooling

Average Pooling has a lot of commons with Max Pooling. The process is very similar. While kernel is sliding over the input image, average pooling is computing the average of values included "below" the kernel window. Average Pooling is holding more information from less significant units of the window, while Max Pooling is just choosing the higher values. In figure 3.11, there is an Average Pooling example with predefined kernel and stride size.

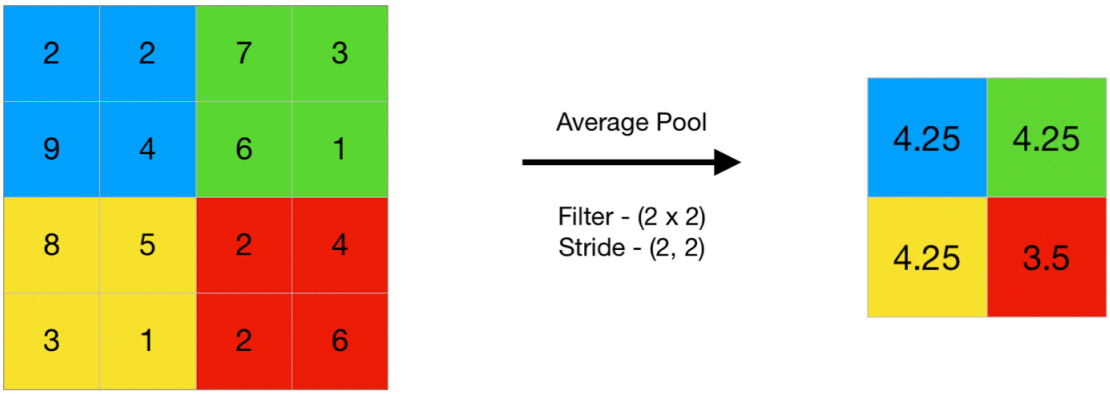


Figure 3.11: Average Pooling with 2x2 kernel and stride[9]

Chapter 4

Test scenario

In this thesis, the focus is more on the verification problem, as we will have to make a comparison between classes and test images. A closer look, has been taken to Support Vector Machines and Artificial Neural Networks. For training and testing our models,we used Labeled Faces in the Wild Dataset (LFW)[33]. This dataset contains low quality pictures and many ethnic groups are underrepresented in LFW. For example there are few children, no babies, very few adults over the age of 80, and a small proportion of women. Furthermore, several ethnic groups have very little or no representation. LFW images are captured in uncontrolled environment. This means that we have image conditions such as, poor lighting, and extreme pose. The faces of each image included in the dataset, is detected with Viola Jones algorithm[34].

Figure 4.1: Samples from LFW dataset[10]

The dataset has 5476 persons. The number of images for each individual is not the same. 21 individuals with the greatest number of samples, where chosen to train and test the models.

Names	Samples
George Bush	109
Colin Powell	52
Tony Blair	32
Donald Rumsfeld	27
Gerhard Schroeder	25
Junichiro Koizumi	18
Luiz Inacio Lula da Silva	17
Hugo Chavez	17
Jean Chretien	16
Lleyton Hewitt	14
John Ashcroft	13
Jacques Chirac	13
Hans Blix	13
Ariel Sharon	12
Serena Williams	12
Laura Bush	11
Arnold Schwarzenegger	10
Jeniffer Capriati	10
Alejandro Toledo	9
Gloria Macapagal Arroyo	9
Vladimir Putin	7

Table 4.1: List of individuals taken to test models

Dataset was splitted into two sets :

- 80 % training set
- 20 % test set

The above separation of the data is only for the Support Vector Machines model. For the Artificial Neural Network models, included the Convolutional one, the separation is :

- 75 % training set

- 20 % test set
- 5 % validation set

The second dataset that we used is Olivetti Faces Dataset. This dataset contains a set of face images taken between April 1992 and April 1994 at AT&T Laboratories Cambridge. It is a dataset with 40 individuals and 400 images. Each person included in the dataset has 10 images assigned with its label. This dataset has very few images so we cant export significant thoughts about our models, but it has the same images per person. This is an issue that we will discuss in the next chapter, how distribution of classes affects our results. Images included in Olivetti Faces Dataset are captured in different light conditions and in different poses. Therefore, all images are captured with a black background[35]. Dataset splitting in Olivetti Faces Database is the same as in LFW.

Figure 4.2: Samples from Olivetti Faces Dataset[11]

4.1 Tests performance calculation

The tests that we perform in chapter 5, are calculated with f1 score. This type of score is function of precision and recall. The math formula is :

$$f1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{4.1}$$

- Precision - For an imbalanced multi-class classification problem, precision is determined as the sum of true positives for each class, divided by the sum of true positives plus false positives for each class.[36]

- Recall - For an imbalanced multi-class classification problem, recall is determined as the sum of true positives for each class, divided by the sum of true positives plus false negatives for each class.[36]

Chapter 5

Implementation and results

In this chapter, several machine learning algorithms and their implementations will be presented and compared. The process of training and testing these algorithms has been integrated through Google Colaboratory [37], which is a Google Research online product. It allows users to write Python code, in a widely used platform for machine learning and data analysis tasks. This happens because Google Colaboratory allows access in some GPUs in order to optimize training with parallel computations. Therefore, Python tools such as NumPy [38], Matplotlib [39], Scikit-learn and Keras Tensorflow [40, 41], were used to implement the algorithms in Google Colaboratory.

Face recognition problem will be examined with a multi-class classification approach, called face identification. Someone could say that this is just an image classification task, but in reality the approach of this thesis is a subset of image classification. The goal of each algorithm's implementation, is to classify face images, which is a complex image classification task. As we will not examine the face detection problem, the workflow of face recognition will start from image normalization, then feature extraction and classification.

5.1 Support Vector Machines model

In support vector machines model, a Support Vector Classifier has been trained and tested with two datasets presented in Chapter 4. The classification type is one-vs-rest, widely known as one-vs-all (ova) classification. The pixel values of the images, are normalized in the range between zero and one, while feature extraction was executed with PCA algorithm. Feature extraction is an important step for training machine learning models, because these extracted

features will be propagated in the model in order to make it learn about the dataset. The data for this model, are one dimensional. Each image is a row of the matrix that represents the dataset. These data are called ravelled, that means that they are don't have a remarkable quality, while there are some noise data within them. Every image of the dataset is represented by a one dimensional array with 2914 features. In order to extract images features, we applied PCA algorithm for 100 eigenvectors. This way, we keep significant data and avoid less significant data(Figure 3.1), that can cause overfit or bad performance. In Figure 5.1 there are some eigenfaces from LFW dataset.

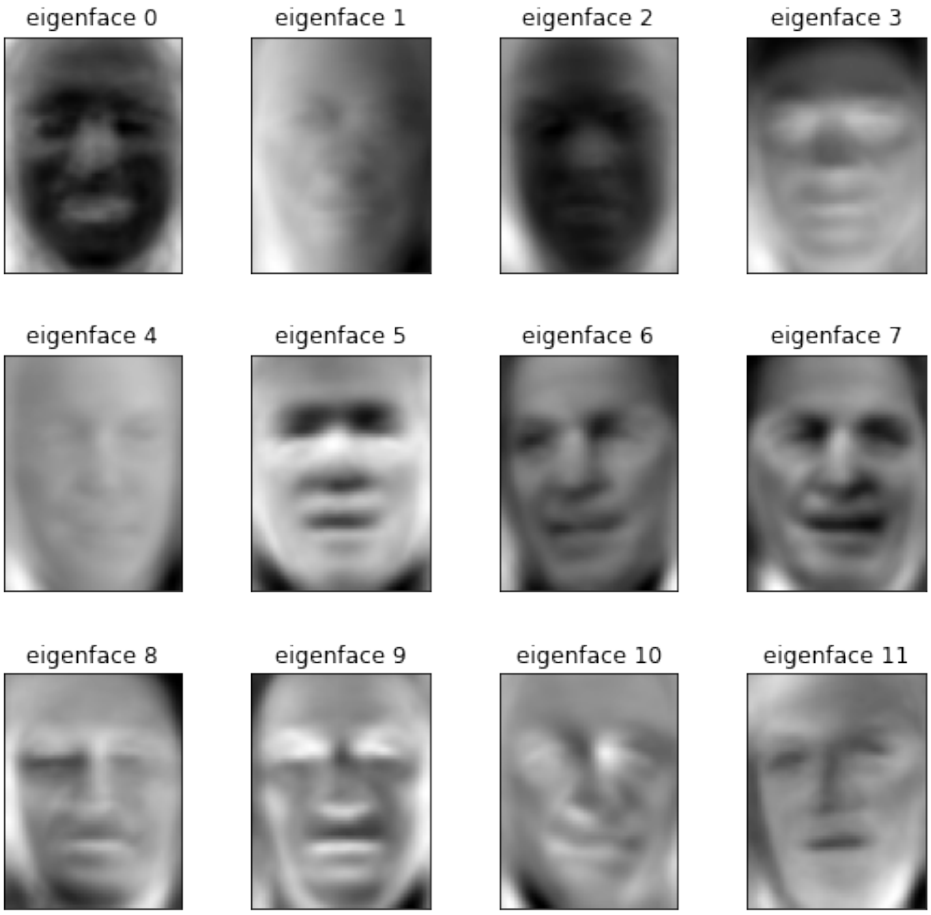


Figure 5.1: Plot of Eigenfaces

A Support Vector Classifier has a lot of hyperparameters to tune. The most significant of them are :

- kernel - kernel type
- C - regularization parameter
- gamma - kernel coefficient

- class weight - multiplier of C. It depends on classes distribution

The hyperparameters for the Support Vector Machines model have been chosen as below :

```
clf=SVC(kernel='rbf', C=1, gamma=0.005, class_weight='balanced')
```

The learning curves of this model, representing the validation accuracy and training accuracy curves are presented below:

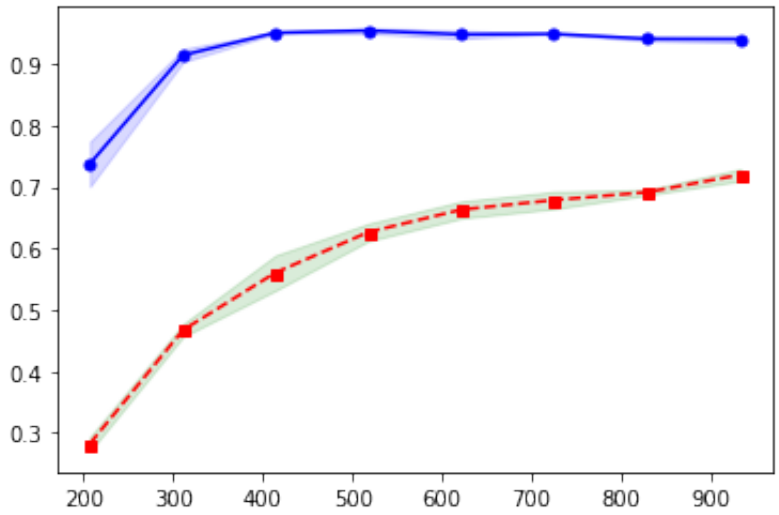


Figure 5.2: Training accuracy (blue) vs validation accuracy (red)

It seems that our model has a kind of overfit. But, we can also see that the training accuracy is starting to decrease during training, while validation accuracy is increasing constantly, which means that the model is starting to generalize. Our model is gaining during training despite the overfit situation. Hyperparameter tuning was performed in order to reduce overfit but the results for unseen data were lower. The recognition rate for this model is 76%. The recognition rate was calculated from f1 score with balanced average. This type of score was used because it seems optimal for multi-class classification problems, especially when an imbalanced dataset is used for training and testing.

After the imbalanced dataset (LFW), we are going to use Olivetti Faces dataset. As it described in Chapter 4, this database has the same number of images per individual, but not so much images per person. There are 400 images, 10 images per person and 40 classes. The learning curves of the model for Olivetti Faces dataset is presented below.

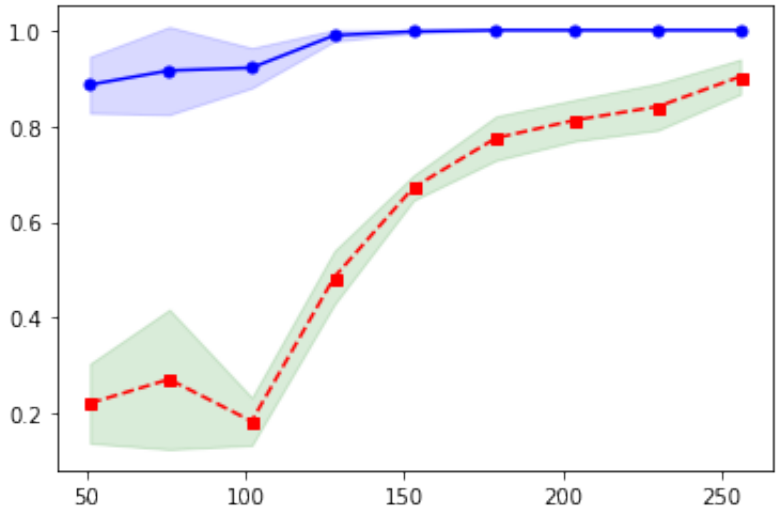


Figure 5.3: Training accuracy (blue) vs validation accuracy (red)

We can notice that the gap between training and validation accuracy curves is much smaller than before. In Figure 5.4, the model seems to avoid the overfit issue. The recognition rate of the model is 95%. A good performance as we expected is reasonable because this dataset includes balanced classes distributions and images may have better quality than in LFW.

5.2 Artificial Neural Networks model

The Artificial Neural Network model is a multi layer perceptron implemented in Keras. The processing flow is the same as in SVM model. Normalization, feature extraction, classification. We choose the images pixels to be valued in the range of zero and one and we extracted the features with PCA. Multi layer perceptrons need low dimensional data. With the PCA algorithm we represent an image with a number of weighted vectors that we choose. Then, we fit these images to the network in order to train it with the backpropagation algorithm. The neural network performance depends on several parameters, the most important of them are presented below :

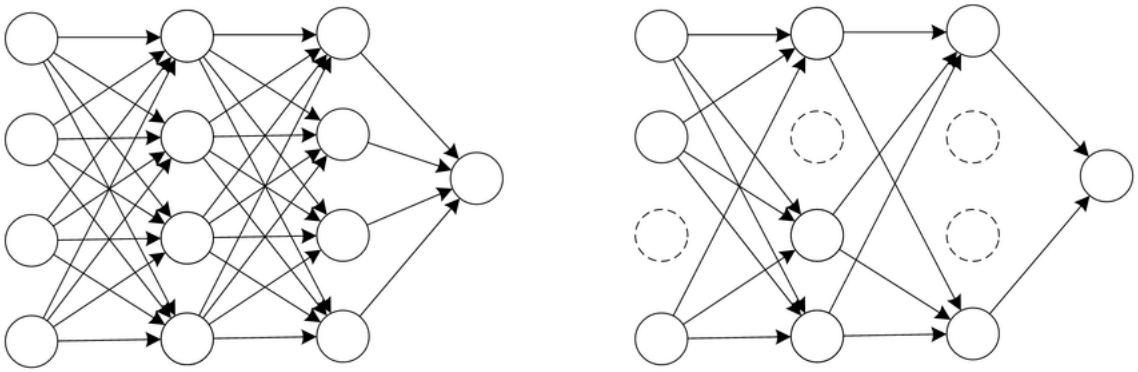
- amount of data
- number of neurons and hidden layers
- activation function

- learning rate
- initial weights

Tests were performed in LFW dataset, as described in Chapter 3. The parameters of the neural network are :

- One hidden layer
- 60 neurons in hidden layer
- Sigmoid activation function
- Softmax activation function in the output layer
- 100 eigenvectors
- Learning rate= 0.005
- Dropout layer with rate 0.6

Dropout is a regularization technique for overfit preventing. The concept is that during training some neurons of the network will be deactivated and the neural network will be forced to learn the same task with different neurons, which is something that improves generalization.



(a) Standard Neural Network (b) Network after Dropout

Figure 5.4: Standard neural network and neural network after dropout[12]

The neural network is trained for 150 epochs with the LFW dataset. The training loss and validation loss curves are presented below. Categorical cross entropy loss is used.

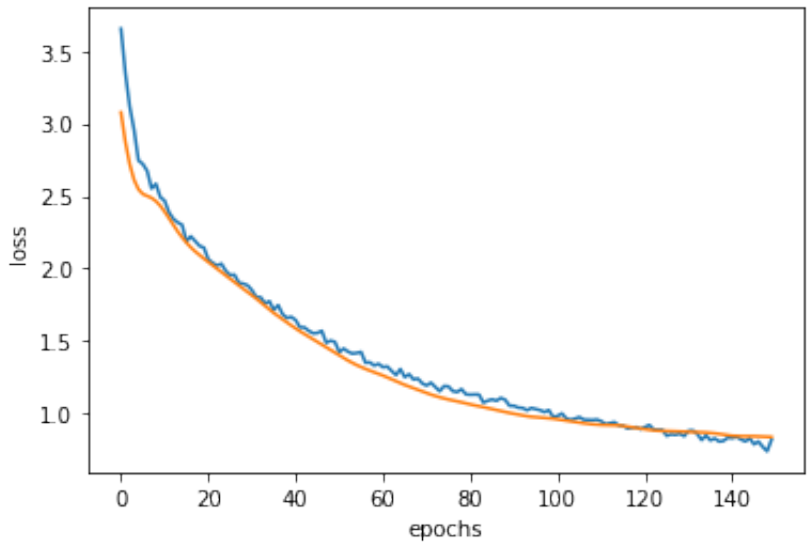


Figure 5.5: Validation loss(orange) vs training loss(blue)

It seems that the neural network is well generalized and there is no overfit but is also important to see validation and training accuracy curves.

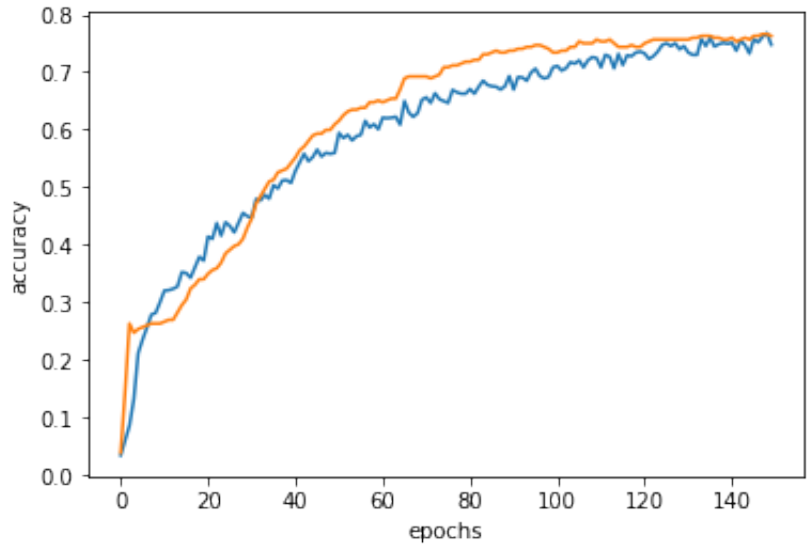


Figure 5.6: Validation accuracy (orange) vs training accuracy (blue)

The recognition rate of the neural network in the test set is 77% and its calculated with f1 score for balanced average such as in SVM model.

Now it's time to train and test this neural network with Olivetti Faces dataset. This dataset has less samples than LFW, so we will train the network for 250 epochs. The validation and training curves are presented below.

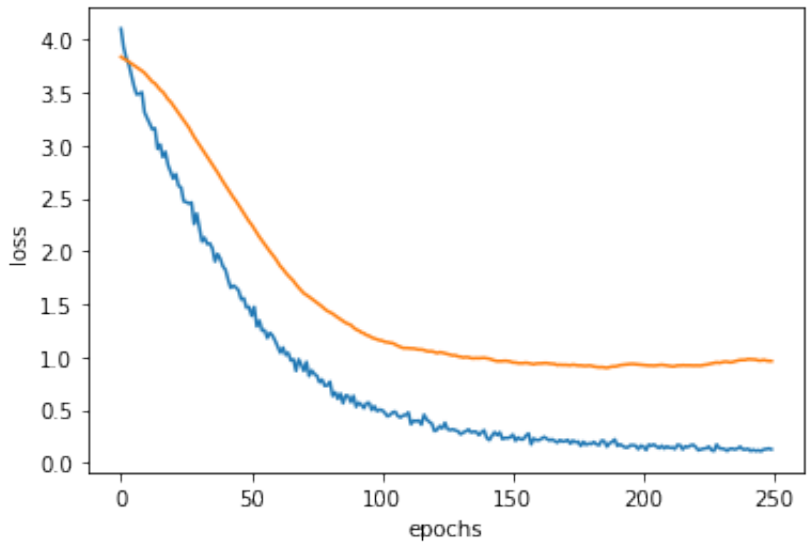


Figure 5.7: Validation loss (orange) vs loss (blue)

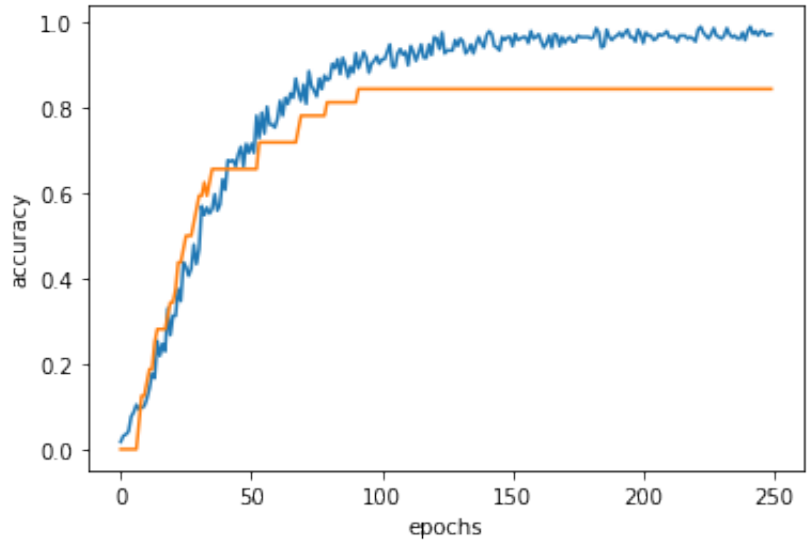


Figure 5.8: Validation accuracy (orange) vs training accuracy (blue)

From Figures 5.8 and 5.9, it is clear that our network is overfitting. In order to reduce overfit the number of eigenvectors and learning rate are changed. The images are represented with 50 eigenvectors, while the learning rate is set to 0.0001. The validation and training loss curves of the network are presented below.

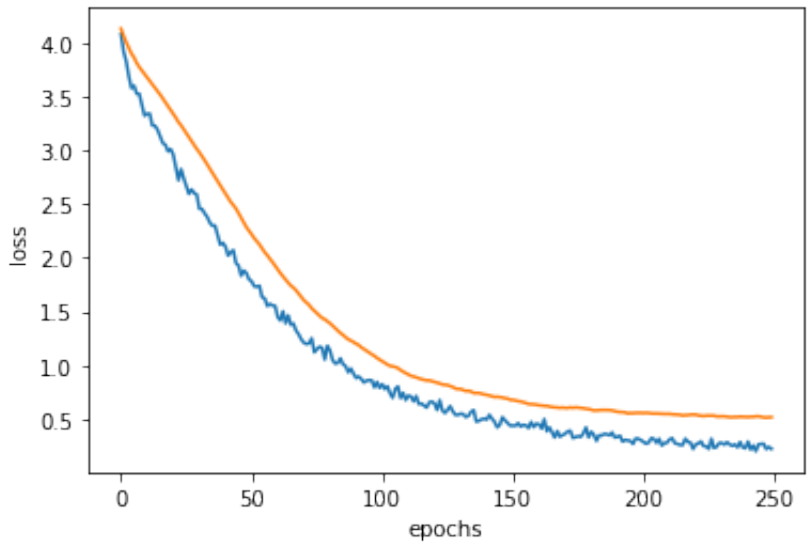


Figure 5.9: Validation loss(orange) vs training loss(blue)

In Figure 5.9 , it is shown that overfit is reduced. The network is getting better during training as the gap between training and validation loss curves is not getting bigger. Let's have a look and at training and validation accuracy curves.

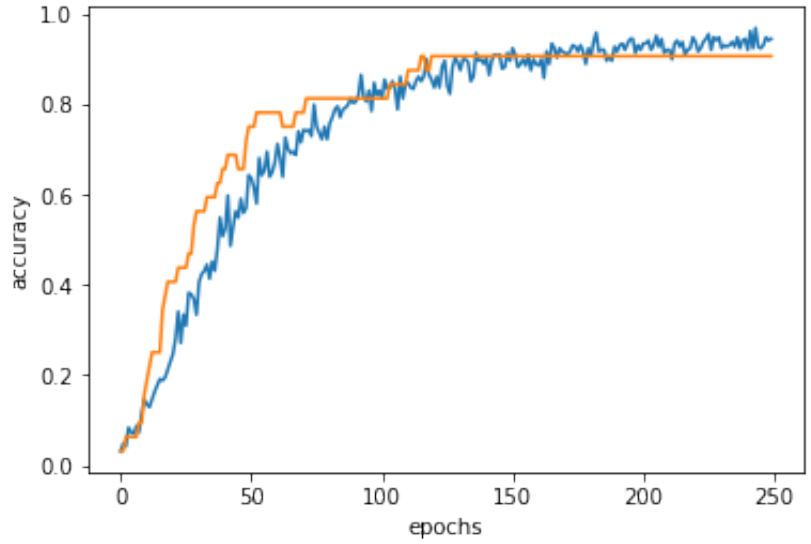


Figure 5.10: Validation accuracy(orange) vs training accuracy(blue)

The recognition rate in test set is 97.5%. As we expected, the performance for Olivetti database is much better than in LFW. So, we notice that is happens the same as in SVM model, which means that the dataset and its classes distributions affect the performance of the network in a significant way.

5.3 Convolutional Neural Networks model

In Convolutional Neural Netwrok model, the process of face recognition is the same as before, but the feature extraction is not a PCA result. We used 2D convolutional layers, so the input is 2D matrices with 3 RGB channels. The input shape of the images is 62x47x3. 3 matrices, that each one is presenting the values for red, green and blue colours. Feature extraction is performed convolutional and pooling layers. The concept in this approach is to add some convolutional and pooling layers at the top of the the neural network we used before in order to let them perform feature extraction and then provide the extracted information to the network for classification. Convolutional layers have a good performance in feature extraction, because convolution operation lets them to find more separable patterns between classes. Someone could say that they work like a magnifying glass. Images are convolved with kernels, widely known as feature extractors and we can choose how many of them to use in each convolutional layer.

The architecture of the CNN is :

- Convolutional 2D layer with 64 8x8 kernels and ReLu activation function
- Batch normalization layer.
- Convolutional 2D layer with 64 8x8 kernels and ReLu activation function.
- Convolutional 2D layer with 64 8x8 kernels and ReLu activation function.
- Batch normalization layer.
- Convolutional 2D layer with 64 8x8 kernels and ReLu activation function
- Convolutional 2D layer with 64 8x8 kernels and ReLu activation function
- Batch normalization layer.
- Average pooling layer with pooling size 8x8 and strides=4
- Convolutional 2D layer with 128 8x8 kernels and ReLu activation function
- Batch normalization layer.
- The fully connected neural network with the same hyperparameters, that we used in the previous approach.(5.2 section)

The loss and accuracy curves of training set and validation set are presented below.

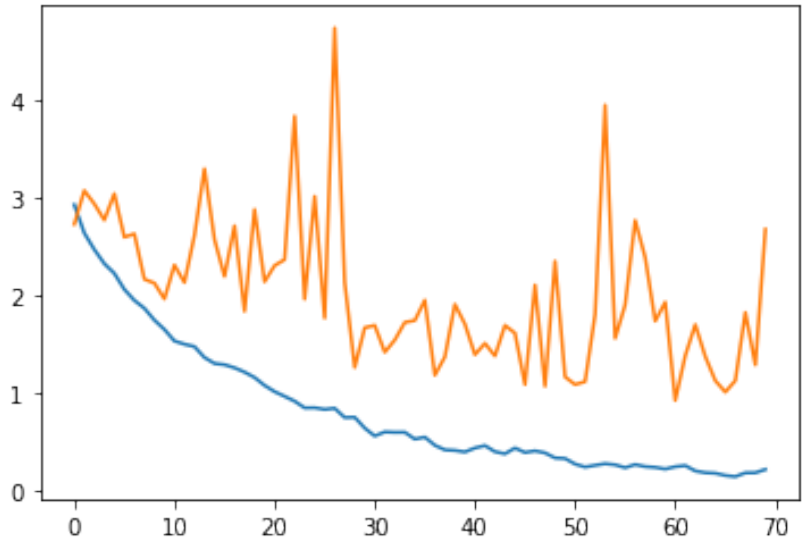


Figure 5.11: Validation loss(orange) vs training loss(blue)

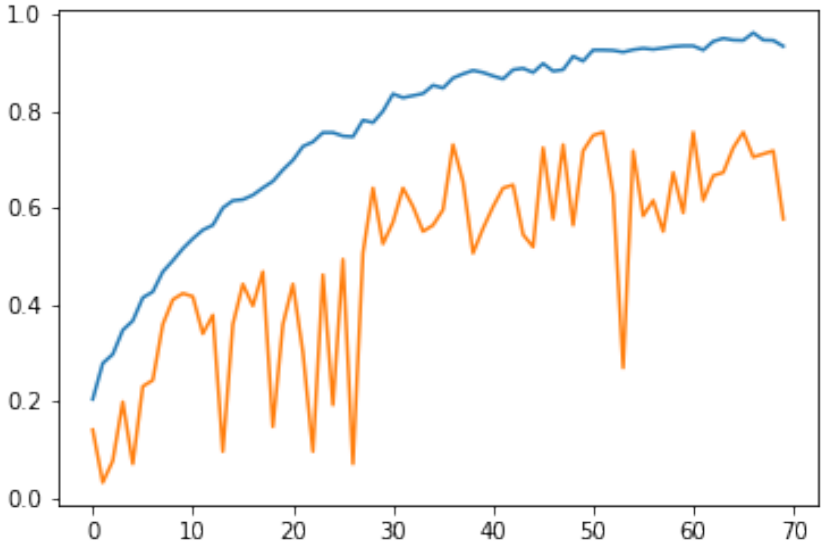


Figure 5.12: Validation accuracy (orange) vs training accuracy (blue)

From Figures 5.12 and 5.13 it is clear that the model is overfitting. Also, it is very important to notice the fluctuations of validation curves in both of the figures. The idea is to add some more regularization with dropout techinque in the fully connected network in order to reduce overfit. Therefore, a good idea to avoid these fluctuations is to reduce the learning rate. After applying these changes progressively to see what happens with loss and accuracy curves, it was decided to set the learning rate at 0.0001 and to increase the dropout rate in the fully connected network from 0.6 to 0.7.

The loss and accuracy curves of the optimized model are presented below.

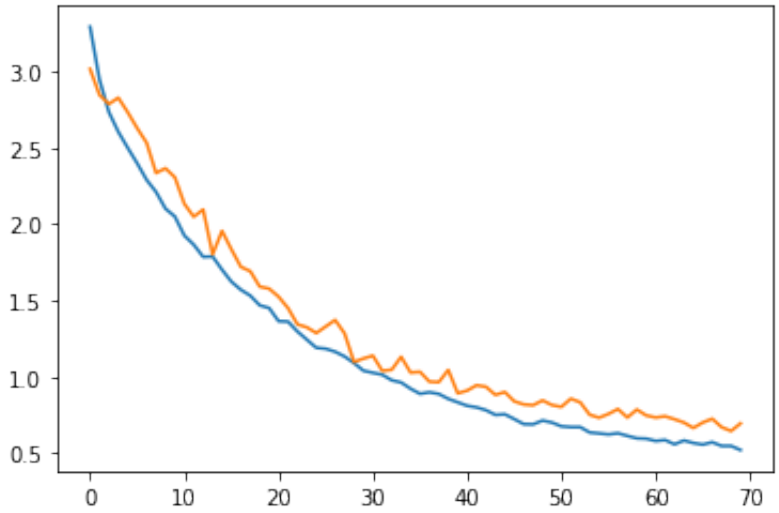


Figure 5.13: Validation loss (orange) vs training loss (blue)

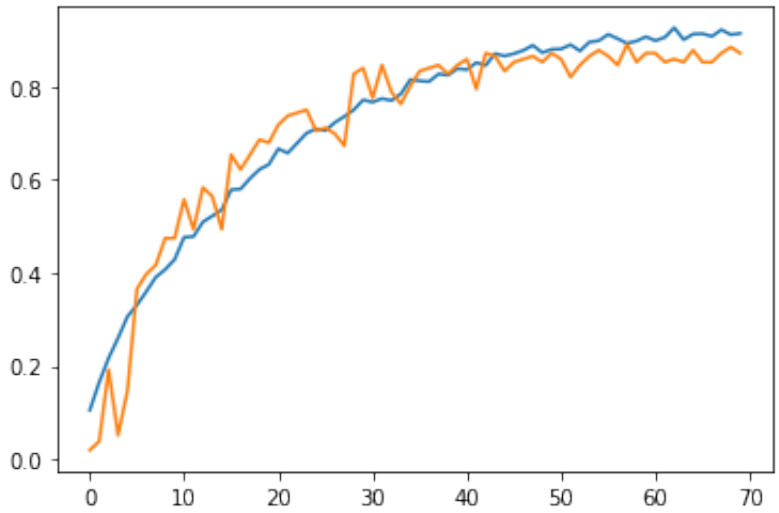


Figure 5.14: Validation accuracy (orange) vs training accuracy (blue)

In Figures 5.14 and 5.15 we notice that the hyperaprameter tuning we performed was successful. We may not have a goodfit, but the model is well generalized and is having progress during training, while the gap between the validation and training loss curves is small and not increasing. The recognition rate in the test for this model, was 89%. This is a good performance in comparison with the previous approaches. The test was calculated from f1 score as in previous approaches.

We will follow the same process as in previous sections, next step is to train and test the optimized CNN model with Olivetti Faces dataset. Training duration will be 70 epochs. The

only difference is in data splits, the validation set took a higher percent of the training data in order to provide us more reliable calculations for the validation set curves. The previous issue, is a consequence of Olivetti Faces dataset size, it has only 400 greyscale images.

The loss and accuracy curves are presented below.

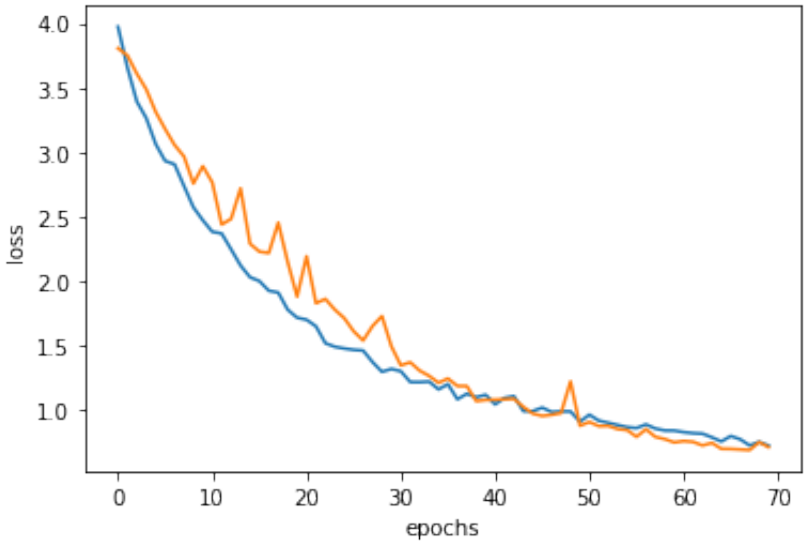


Figure 5.15: Validation loss(orange) vs training loss(blue)

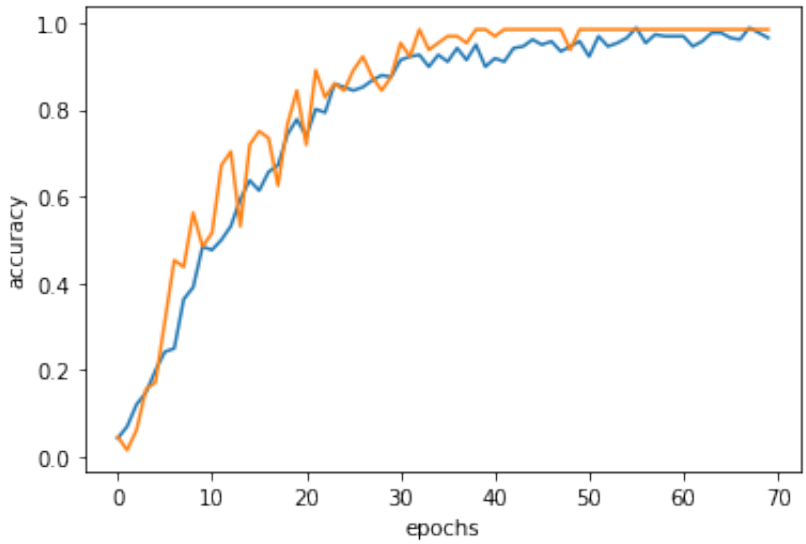


Figure 5.16: Validation accuracy (orange) vs training accuracy (blue)

As it seems from the above figures, our model is well generalized despite the fluctuations in the first 30 epochs, which may are caused from the small size of the validation set. The recognition rate is 100% , calculated as alwyas with f1 score. But, we have to notice that the testing set is also small, actually is quite bigger than the validation set, so in a hypothetical

scenario that we have more samples for testing, the recognition may not be the absolute 100% and be a a bit smaller, something like 99%.

5.4 Results

The table below, is presenting a results summary for these three unsupervised machine learning approaches we examined before in two different datasets.

Models	LFW	Olivetti Faces dataset
SVM	75%	95%
ANN	77%	97.5%
CNN	89%	100%

Table 5.1: A test results summary

The ANN model seems to have a little better perfofmance than the SVM one. We also have to notice that the ANN models generalize better than the SVM models, avoiding the overfit more effectively. CNN models seem to be far better than the two previous approaches. Therefore, it is important to emphasize that the only change we made in the face recognition process in CNN, is that we performed feature extraction with convolutional and pooling layers and not with PCA. Convolutional layers are very effective in feature extraction , especially when they are combined with pooling ones for downsampling. Because CNN results are far better, it is easy to realize that feature extraction is a very important step in image processing and classification.

Chapter 6

Conclusion

In Table 5.1, it is easy to notice that all the implemented algorithms have a better performance in Olivetti Faces dataset than in LFW. As it is described in chapter 4, the differences between these two datasets are their classes distributions and the amount of data that they include. Generally, the development of a system that performs extremely good in datasets like LFW is a difficult challenge, but a more useful one. Unfortunately, we can't say that the experiments we provided meet real world face recognition tasks because real world problems have to deal with much more data. We could say that the face recognition problem is examined in a smaller scale in this Thesis, that helps us to extract significant conclusions about the algorithms performances and the affect that different kind of datasets may have in training and testing phases. CNN models have a far better performance in both of the datasets in comparison with the SVM and ANN models, because of the effectiveness of convolutional and pooling layers in feature extraction process. A remarkable notice is that ANN and CNN may perform better, but both of these techniques suffer from a large computational cost in training phase.

Bibliography

[1] Articles - principal component methods in r: Practical guide principal component analysis in r: prcomp vs princomp. http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/118-principal-component-analysis-in-r-prcomp-vs-princomp/#pca-results-for-variables. Ημερομηνία πρόσβασης: 1-9-2021.

[2] Gabriele Novembri, Francesco Rossini, and Antonio Fioravanti. *Construction time and cost optimization using A.I. and statistical methods, through Bayes-Point Machines*, page 40. 09 2017.

[3] Marcelo Espinoza, Johan Suykens, Ronnie Belmans, and Bart De Moor. Electric load forecasting. *Control Systems, IEEE*, 27:43 – 57, 11 2007.

[4] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158, 11 2017.

[5] Ramon Quiza and J. Davim. *Computational Methods and Optimization*, pages 177–208. 01 2011.

[6] A quick introduction to neural networks. https://towardsai.net/p/machine-learning/machine-learning-vs-deep-learning. Ημερομηνία πρόσβασης:5-9-2021.

[7] Convolutional neural networks (cnns / convnets). https://cs231n.github.io/convolutional-networks/. Ημερομηνία πρόσβασης: 6-9-2021.

[8] Ulugbek Abdullaev, Bedilbek Khamidov, Sherzod Niyazov, Shakhobiddin Urmanov, Khurshidbek Bakhromov, Tulkinjon Isakov, Saida Yusupova, Rustamjon Khalmatov, and Navruz Jandullaev. Convolutional neural networks for image classification, 12 2017.

[9] Cnn | introduction to pooling layer. `https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/`. Ημερομηνία πρόσβασης: 6-9-2021.

[10] Emrah Basaran, Muhittin Gökmen, and Mustafa Kamasak. An efficient multiscale scheme using local zernike moments for face recognition. *Applied Sciences*, 8:827, 05 2018.

[11] Xiaoqin Hu, Ling Jing, and Qian Li. A discriminate multidimensional mapping for small sample database. 09 2021.

[12] Amine ben khalifa and Hichem Frigui. Multiple instance fuzzy inference neural networks. 10 2016.

[13] Lawrence Sirovich and M Kirby. Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America. A, Optics and image science*, 4:519–24, 04 1987.

[14] Matthew Turk and Alex Pentland. Eigenfaces for recognition. 3(1):71–86, 1991.

[15] P.N. Belhumeur, J.P. Hespanha, and D.J. Kriegman. Eigenfaces vs. fisherfaces: recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720, 1997.

[16] K. Etemad and R. Chellappa. Face recognition using discriminant eigenvectors. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, volume 4, pages 2148–2151 vol. 4, 1996.

[17] Principal component analysis: a review and recent developments. `http://www.latex-project.org`. Ημερομηνία πρόσβασης: 2-9-2021.

[18] A step-by-step explanation of principal component analysis (pca). `https://builtin.com/data-science/step-step-explanation-principal-component-analysis`. Ημερομηνία πρόσβασης: 2-9-2021.

[19] Support vector machine — introduction to machine learning algorithms. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. Ημερομηνία πρόσβασης: 9-1-2021.

[20] V. Jakkula. Tutorial on support vector machine (svm). 2011.

[21] J. P. Lewis. A short svm (support vector machine) tutorial.

[22] The kernel trick in support vector classification. http://www.latex-project.org. Ημερομηνία πρόσβασης: 1-9-2021.

[23] Simon S. Haykin. *Neural Networks and learning machines*. Pearson Education, Upper Saddle River, NJ, 3 edition, 2009.

[24] Explained: Neural networks. https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414. Ημερομηνία πρόσβασης: 3-9-2021.

[25] Neural network: Architecture, components i& top algorithms. https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/. Ημερομηνία πρόσβασης: 3-9-2021.

[26] A quick introduction to neural networks. https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/. Ημερομηνία πρόσβασης: 3-9-2021.

[27] A. Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90, 2012.

[28] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[29] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1990.

[30] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *ArXiv e-prints*, 11 2015.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning , chapter9: Convolutional Neural Networks, Convolutional process*. The MIT Press, 2016.

[32] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.

[33] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[34] Labeled faces in the wild home. `http://vis-www.cs.umass.edu/lfw/`. Ημερομηνία πρόσβασης: 2-9-2021.

[35] The olivetti faces dataset. `https://scikit-learn.org/0.19/datasets/olivetti_faces.html`. Ημερομηνία πρόσβασης: 2-9-2021.

[36] Explained: Neural networks. `https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/`.

[37] Ekaba Bisong. *Google Colaboratory*, pages 59–64. Apress, Berkeley, CA, 2019.

[38] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[39] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[40] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent

Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[41] François Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

APPENDICES

Appendix A

Python code

A.1 SVM combined with PCA

The presented code below remains the same independetly of the dataset that we use. The only difference is that we have to fetch the of OFD images instead of the LFW images.

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.metrics import classification_report, f1_score
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import learning_curve
from sklearn.metrics import accuracy_score

lfw_people = fetch_lfw_people(min_faces_per_person=39,resize=0.5)
n_samples, h, w = lfw_people.images.shape
X = lfw_people.data
n_features = X.shape[1]
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]
```

```
print(”Total dataset size:”)
print(”n_samples:” , n_samples)
print(”n_features:”, n_features)
print(”n_classes:”, n_classes)
print(X.shape)

X_train, X_test, y_train, y_test = train_test_split(X, y,

test_size=0.2,random_state=0)

n_components = 100;
pca = PCA(n_components=n_components, whiten=True).fit(X_train)
eigenfaces = pca.components_.reshape((n_components, h, w))

X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

print(”Fitting the classifier to the training set”)

clf=SVC(kernel='rbf', C=1, gamma=0.005, class_weight='balanced')
clf = clf.fit(X_train_pca, y_train)

train_sizes, train_scores, test_scores=learning_curve(clf,X_train_pca,
y_train,scoring='accuracy',
train_sizes=[0.2, 0.3, 0.4,0.5, 0.6,0.7,0.8,0.9, 1.0], cv=5)

print(”Predicting the people names on the testing set”)
y_pred = clf.predict(X_test_pca)
y_pred2=clf.predict(X_train_pca)
print(classification_report(y_test, y_pred, target_names=target_names))
print(confusion_matrix(y_test, y_pred, labels=range(n_classes)))
```

```
score = f1_score(y_test, y_pred,average='weighted')
train_score=f1_score(y_train, y_pred2,average='weighted')
print(train_score)
print(score)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.plot(train_sizes, train_mean,
         color = ”blue”, marker = 'o',
         markersize = 5,label  ='Training accuracy')

plt.fill_between(train_sizes,
                 train_mean + train_std,
                 train_mean - train_std,
                 alpha=0.15, color = 'blue')
plt.plot(train_sizes, test_mean,
         color='red', linestyle = '--',
         marker = 's',markersize = 5,
         label = 'Validation accuracy')
plt.fill_between(train_sizes,
                 test_mean + test_std,
                 test_mean - test_std,
                 alpha=0.15, color = 'green')

plt.show()
```

A.2 ANN combined with PCA

A.2.1 Training and testing with LFW dataset

```
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import learning_curve
import numpy as np
import matplotlib.pyplot as plt
from numpy import mean
import tensorflow as tf
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.datasets import fetch_olivetti_faces
from sklearn.datasets import fetch_lfw_people
from sklearn.decomposition import PCA



lfw_dataset = fetch_lfw_people(min_faces_per_person=39, resize=0.5,

color=True)
print(lfw_dataset.keys())
print(lfw_dataset.images)

#printing the set side by side
print("Size of data set: \n")
samples, h, w ,c = lfw_dataset.images.shape
print(h)
print(w)
print(h*w)
X = (lfw_dataset.data)/255
print(X)
print(samples)
```

```
print(X.shape)
features = X.shape[1]
print("features: \t %d \n" % features)

y = lfw_dataset.target
target_names = lfw_dataset.target_names
classes = target_names.shape[0]
print("classes: \t %d \n" % classes)

from sklearn.model_selection import train_test_split
# split into a training and testing set
X_train, X_test, y_train, y_test = train_test_split(X,
y,test_size=0.2, random_state=0, shuffle=True)

print(X.shape)

n_components =100;
print("Computing PCA.....")
pca = PCA(n_components=n_components, whiten=True)
pca.fit(X_train)



X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

print(X_train_pca.shape)
print(X_test_pca.shape)
print(y_train)
print(X_train.shape)

model = tf.keras.models.Sequential([
tf.keras.layers.Dense(60,activation='sigmoid', input_dim=100),
```

```
tf.keras.layers.Dropout(0.6),
tf.keras.layers.Dense(21,activation='softmax'),
])
opt = tf.keras.optimizers.Adam(learning_rate=0.005)

model.compile( optimizer=opt,loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

model.fit(X_train_pca, y_train, batch_size=1024, epochs=150,validation_

model.summary()

print(model.history.history)
print(model.history.history['loss'])
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
plt.xlabel(”epochs”)
plt.ylabel(”loss”)
plt.show()
plt.plot(model.history.history['accuracy'])
plt.plot(model.history.history['val_accuracy'])
plt.xlabel(”epochs”)
plt.ylabel(”accuracy”)
plt.show()

y_pred=model.predict(X_test_pca)
y_classes=y_pred.argmax(axis=-1)

train_pred=model.predict(X_train_pca)
train_classes=train_pred.argmax(axis=-1)
test_score=f1_score(y_classes, y_test,average='weighted')
train_score=f1_score(train_classes,y_train, average='weighted')
```

```
print(classification_report(y_test,y_classes,target_names=target_names))
print(confusion_matrix(y_test, y_classes,labels=range(classes)))

print(test_score)
print(train_score)
```

A.2.2 Training and testing with OFD

```
from sklearn.model_selection import learning_curve
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score
import tensorflow as tf
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.datasets import fetch_olivetti_faces

olivetti = fetch_olivetti_faces()

#printing the set side by side
print("Size of data set: \n")
samples, h, w  = olivetti.images.shape
print(h)
print(w)
print(h*w)
X = (olivetti.data)/255
print(X)
print(samples)
print(X.shape)
features = X.shape[1]
print("features: \t %d \n" % features)
y = olivetti.target
```

```
classes = 40
print("classes: \t %d \n" % classes)
from sklearn.model_selection import train_test_split
# split into a training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2,random_state=0, shuffle=True)

print(X.shape)
from sklearn.decomposition import PCA
n_components =100;
print("Computing PCA.....")
pca = PCA(n_components=n_components, whiten=True)
pca.fit(X_train)

X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

print(X_train_pca.shape)
print(X_test_pca.shape)
print(y_train)
print(X_train.shape)

model = tf.keras.models.Sequential([
tf.keras.layers.Dense(60,activation='sigmoid', input_dim=100),
tf.keras.layers.Dropout(0.6),
tf.keras.layers.Dense(40,activation='softmax'),
])
opt=tf.keras.optimizers.Adam(learning_rate=0.005)
model.compile( loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

model.fit(X_train_pca, y_train, batch_size=16 , epochs=250,
```

```
validation_split=0.1)
model.summary()
```

A.3 CNN

The code below presents the CNN model that is trained and tested with LFW dataset. When we train and test the CNN model with OFD, the only change is that we give some more samples in the validation set in order to have descriptive validation curve results.

```
from sklearn.datasets import fetch_lfw_people
import numpy as np
import matplotlib.pyplot as plt
from numpy import mean,log
from sklearn.metrics import accuracy_score, f1_score
import tensorflow as tf
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
lfw_dataset = fetch_lfw_people(min_faces_per_person=39,
resize=0.5, color=True)
print(lfw_dataset.keys())
print(lfw_dataset.images)

print("Size of data set: \n")
samples, h, w ,c = lfw_dataset.images.shape
print(h)
print(w)
print(h*w)
X = (lfw_dataset.images)
print(X)
print(samples)
print(X.shape)
features = X.shape[1]
print("features: \t %d \n" % features)
```

```
y = lfw_dataset.target
target_names = lfw_dataset.target_names
classes = target_names.shape[0]
print("classes: \t %d \n" % classes)

from sklearn.model_selection import train_test_split
# split into a training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

X_train_new , X_validation, y_train_new, y_validation =
train_test_split(X_train, y_train, test_size=0.1, random_state=0)
print(X.shape)
print(X_validation.shape)

cnn = tf.keras.models.Sequential()
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=8,
activation='relu', input_shape=[62, 47, 3]))
cnn.add(tf.keras.layers.BatchNormalization())
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=8,
activation='relu' ))
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=8,
activation='relu' ))
cnn.add(tf.keras.layers.BatchNormalization())
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=8,
activation='relu' ))
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=8,
activation='relu' ))
cnn.add(tf.keras.layers.BatchNormalization())
cnn.add(tf.keras.layers.AveragePooling2D(pool_size=8,strides=4))
cnn.add(tf.keras.layers.Conv2D(filters=128, kernel_size=2,
```

```
activation='relu' ))
cnn.add(tf.keras.layers.Dropout(0.2))
cnn.add(tf.keras.layers.BatchNormalization())
cnn.add(tf.keras.layers.Flatten())

opt = tf.keras.optimizers.Adam(learning_rate=0.0001)

cnn.add(tf.keras.layers.Dense(units=60, activation='sigmoid'))
cnn.add(tf.keras.layers.Dropout(0.7))
cnn.add(tf.keras.layers.Dense(units=21, activation='softmax'))
cnn.summary()
cnn.compile(optimizer = opt, loss = 'sparse_categorical_crossentropy',
metrics = ['accuracy'])
cnn.fit(X_train_new,y_train_new,validation_data=
(X_validation,y_validation),shuffle=True,epochs=70,batch_size=64)

print(cnn.history.history)
print(cnn.history.history['loss'])
plt.plot(cnn.history.history['loss'])
plt.plot(cnn.history.history['val_loss'])
plt.show()
plt.plot(cnn.history.history['accuracy'])
plt.plot(cnn.history.history['val_accuracy'])
plt.show()

y_pred=cnn.predict(X_test)
test_predictions=y_pred.argmax(axis=-1)
train_pred=cnn.predict(X_train)
train_predictions=train_pred.argmax(axis=-1)

test_score=f1_score(test_predictions, y_test,average='weighted')
train_score=f1_score(train_predictions,y_train, average='weighted')
```

```
print(classification_report(y_test,
test_predictions, target_names=target_names))
print(confusion_matrix(y_test, test_predictions,labels=range(classes)))
print(test_score)
print(train_score)
```