‘Learning convolutional Neural Network for Face Anti-Spoofing”

Beatriz Gómez Ayllón

May 9, 2017

Abstract

The purpose of this thesis is developing an algorithm to detect spoofing attacks for a face biometric system. The face anti-spoofing algorithm consists in the implementation of a Convolutional Neural Network and the best classify that solves the problem.

Some face databases are used for this purpose: CASIA, FRAV and MFSD-MSU databases. These databases are images databases of real users and different attacks types as printed photos, masks or photos displayed in a smarphone or tablet that are used to train the CNN and the classifiers.

The output of the CNN is the input of the classifiers. The classifiers that are goiing to be trained and tested are KNN, SVM, Decision Tree and Logistic regression. Also, reduction algorithm methos as LDA and PCA are used with each classifier.

The Programming language use in this project is Python and the Theano has been used as main library to develop the convolutional neural network.

Contents

Part I

MEMORY

Chapter 1

Introduction

1.1 Motivation

Artificial intelligence is a field that I, personally, find very interesting. Convolutional neural networks and its bio-inspired basis is the most attractive subject that I have considered: how from images, features are learned and the big accuracy of pattern recognition task is gotten makes me learn deeply in this subject. This is the main reason on focusing in this thesis in convolutional neural networks.

The object with which neural network would be used in this thesis is face spoofing detection, this is because I would like to work with real problems that could be used in the real world like is this subject, that is being part of the *ABC4EU European Project*. Because of this and the importance of detecting cases of identity theft in nowadays security has driven me to work in spoofing detection attacks using convolutional neural networks.

1.2 Objectives

The principal objective of this thesis is being able to classify face images from people. Those images are people images (real users) and images from people trying to impersonate (attacks).

The samples used are from different databases: CASIA, FRAV and MFSU database. Those databases are formed by real users and attacks samples (images of people with mask, images from smart phones or tablets, ...) and have been used in recognition and spoofing task before.

To differentiate between real users images and the attacks images a Convolutional Neural Netork (CNN) has been developed using Python language and Theano (a Python library). The secondary objective is learning this artificial intelligence research field that

nowadays has a significance importance.

Images from each database are feed to the CNN and the output features obtained are the input of the classifiers. Various classifiers has been trained (KNN, logistic regression, Support Vector Machine, ...) for this purpose.

Get the best combination between CNN and classifier to classify correctly the most number of samples is another objective. Knowing what database is better for this thesis objectives and why also would be part of the objective.

Finally, trying to differentiate characteristics of the well-classified and bad-classified samples would be interesting. The last objective is identify a pattern (if exists) of bad-classified samples.

1.3 Thesis structure

This thesis is formed by a brief introduction, presented in chapter 1, where the motivation to carry out this project has been exposed and this thesis main objectives are described.

Before go ahead to the development of the main part, in chapter 2, a short introduction to neural network is presented as well as anti-spoofing projects developed in others research articles. The classifiers and metric used along the project is shown in this chapter too.

In chapter 3, the main part is developed, here the steps that have led to the final architecture of the neural networks are explained.

In the following chapter 4 the results obtained in the previous chapter 3 are presented and discussed.

The last chapter 5, the conclusions obtained along the thesis are discussed and the future work is proposed.

Chapter 2

Theoretical Basics

2.1 Introduction

In this chapter, the theoretical basis of anti-spoofing and convolutional neural networks are exposed in order to understand the whole document. In addition, the classifiers used and the metrics to compare them are defined in this chapter as well as the databases that are going to be used to train and test the CNN and classifiers.

2.2 Anti-spoofing and biometrics systems

At the same time as technology advance, capture systems and processing algorithms have proceed too. This open huge of quantity the doors and options and one of them is the capability of developed more sophisticated identification systems, giving way to biometrics systems.

2.2.1 Spoofing and biometrics introduction

The need to get characteristics from people to identify them appeared in the 1870s when Alphonse Bertillon measures doby parts as skull diameter and arm and foot length to identify prisoners in USA until the 1920s. There were in the 1880s when fingerprint and facial identification were proposed. But with the appearance of digital signals processing systems in 1960, the voice and the fingerprint biometric systems were started to be investigated and researches started to think to use this system to identification in access control security. Ten years later, the geometry of the hand was started to be a field of interest for automated technologies of identification. The retina and signature verification appeared in the 80s and after a short time the face systems. The last biometrics systems appeared in the 1990s with the iris recognition [1].

Some biometrics systems have been mentioned above. Each one uses a specific part of the human body to identify or verify an identity. But there are more biometrics than the described above as the veins or the ear.

As the same time as biometrics authentication systems appeared, the attacks to trick systems emerge and they are called *spoofing attacks*. For example making plaster molds for geometric hand biometrics or fingerprints would be an attack as the presented in figure **??** (image has been obtained from [2]).

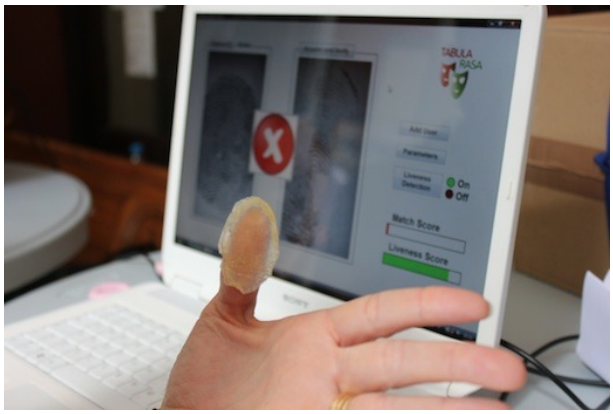


Figure 2.1: Spoofing fingerprint. Image obtained from [2]

Spoofing is referenced to create a new and genuine identity or impersonate a system. The attack would depend on the biometry and the capturing system [3].

In order to get the tricks and do not allow spoofing attacks, *anti-spoofing* tries to minimize the attacks or prevent them.

There are two ways of evaluating anti-spoofing scenarios [3]:

algorithm-based or technology evaluation: to evaluate algorithms or liveness detection.

system-based or scenario evaluation: to evaluate the entire acquisition systems.

When an identification or biometric system is used two general task could perform:

Recognition: when the system, from the input, has to decide the identity of the user (name, ID,...)

Verification: when the system is given a determined identity and has to decide if it is the genuine or is a person trying to supplant the identity.

In this project it is going to be developed the algorithm to detect anti-spoofing attacks when the face is using as biometrics. Given an image, the system has to decide if it is an

attack or a genuine user. It would be a verification task.

2.2.2 Face anti-spoofing

Face is a very used biometric system and also it is one of the most attacked biometrics because of the fact that the capturing system for face biometrics could be as simple as a RGB camera, so a printed photo or an image in a smartphone could be easy to acquire as spoofing attacks. Otherwise, a spoofing attack for an vein biometrics is more sophisticated than a printed photo.

The principal or main used acquire system are RGB cameras although other cameras as infra-red or depth camera are used too. Using different types of camera could help to detect anti-spoofing attacks. The main used attacks for this biometric is a printed image or a mask and displaying a photo or video in a smartphone or tablet and its cost is low [4].

Two big advantages of this biometric are the non-intrusive method and the positive acceptance of this biometric in society. On the other hand, compare frontal face with faces with a determined angle is not as well managed as compared frontal with frontal faces [5].

2.2.3 Related Works

There are a lot of research work in face anti-spoofing and different method has been tested in researches.

Deep learning has been used to detect spoofing attacks. Convolutional neural networks are used to extract features and be allowed to differenciate between genuine and fake users. In [6] A LSTM-CNN has been developed, where the input in stead of being images are video and the network learn temporal features. In [7] the purpose is the verification task, where the input are two images and the convolutional neural network is a Siamese. Another Anti-spoofing research with deep learning is the developed in [8] where a simple Convolutional neural network has been developed to difference between genuine and no-real users.

Not only deep learning has been used with this purpose. The extraction of texture-based features (LBP, HOG, DoG,..) for face anti-spoofing have been researched [4]. In [9] authors study Local Binary Patterns (LBP) and variants of this method for face anti-spoofing.

Also the motion has been used. The lip movement, the head rotation or the blink eyes [4]. In [10] is the blink movement of the eyes what authors use for face anti-spoofing.

2.3 Neural Network background theory

In this section, the basis and the theoretical background of the convolutional neural networks theory is exposed.

2.3.1 Historical introduction

The historical introduction to neural networks is going to be summarized [11]:

Neural Networks are introduced by first time in the years 40 by Warren McCulloch and Walter Pitts, more specifically, simple models of neural networks (switches based on neurons) were able to calculate almost every arithmetic problems and logic operations. Few years layer, the recognition of spacial patterns field was defined as a interesting scope for neural networks.

In 1949 the 'Hebbian rule' was postulate by Donald O. Hebb. The rules describes the generalized learning basis of neural networks. This rule explain that two connected neurons have a bigger strength if they are active at the same time, and the strength changes proportionally to the product of the two activities.

From 1951, the golden age of neural networks start although it stopped with a big pause. The first neurocomputer which was capable to adjust the weights by itself was developed in 1951 and was called *Snark*. The fist neurocomputer able to recognize simple numbers was called *'Mark I perceptron'* as was developed between 1957 and 1958.

It was in 1959 when Franck Rosenblatt defined the *perceptron* and the *perceptron convergence theorem* was verified. One year later, the *ADALINE (ADAptive LInear NEuron)* was developed, this was the first commercially used neural network which was a fast and precise system. One important characteristic was the *delta rule* rule used for the training procedure. Before the golden age finished, researched figured out that the XOR function was not able to be solve with just one perceptron.

From these years, neural networks does not have much importance, until computational resources were enough. But some important advances were developed ad the *linear associator* model (an associative memory model). The backpropagation of error, a very used learning procedure was defined in 1974 by Paul Werbos. *The self-organizing feature*

maps were described in 1982, there are also known as Kohonen maps. In 1983, a neural model able to recognize handwritten characters was developed as an extension of the Cognitron, the new model was called Neocognitron.

From this time on, neural networks are having a significant importance and are researched widely, it has to notice, that nowadays the computational resources are bigger than some years and that makes that the researchers could be more extensive.

2.3.2 Introduction to ANN

Humans, along the history, have tried to reproduce nature. Evolution, has turned out to be a big coordination in nature. If object recognition, associate concepts, memorize or extracting the semantic of images are focused, humans are provided of those processing information qualities. Trying to get those qualities with technology is a current (and not surprising) challenge.

One of the matters that is increasing its importance are neural networks which are inspired in biological neural networks. Neural networks are a important subject of the Artificial Intelligence (AI) and it is a new way (and more sophisticated) of processing information [12].

2.3.3 Biological Neural Networks

Biological neural networks are part of the nervous system and are formed by neurons units or nervous cells. Each neuron is a capable of process information in different ways by itself [12].

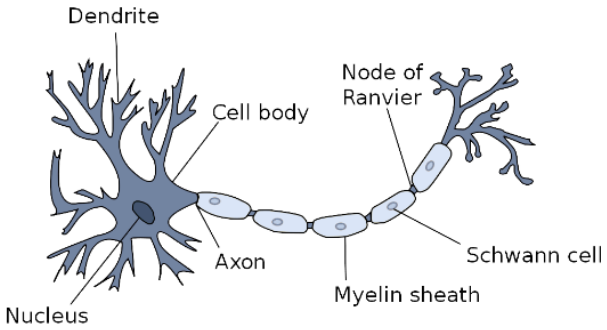


Figure 2.2: Biological Neural Network. Image obtained from [11]

The principal components of a neuron are the nucleus, dendrite, cell body, axon, schwann cell, the node of Ranvier... In figure 2.2 a biologica neural is shown in which its components are signalized [11].

The soma is the spherical central part of the nerve cell in whose there is a salt and potassium concentration which is covered by the neuronal membrane. Inside the soma is the neuronal nucleus and from the soma branches are extend (dendrites). Nerve cells are connected among them by dendrites. Neurons are transmitting and receiving nervous signals continuously, communicating among them. The information transference is produced, more specifically, in the synaptic clef (space between two neurons connection). This exchange of information is denominated synapses and it is made though electrochemical activities. The axon is a soma extension whose responsibility is transmitting the information electrochemical out of the neuron, by the nervous system thanks to its terminal branches [11, 13]

2.3.4 Artificial Neural Networks (ANN)

As the same way as the nervous systems is formed by neurons, artificial neural networks are composed by artificial neurons. Each artificial neuron is a processing unit whose input is processed and shared to another neuron or to the output. Neurons are connected among them [11]

Basically, there are three types of artificial neurons:

Input neurons, if belong to the input layer. This neurons receive the input data of the network.

Hidden Units, if belong to the processing layers. There are many types of layers: convolutional, pooling, dropout... There could be as many hidden layers and hidden units as user desire. The connectivity and the topology of the layers define the topology of the network.

Output neurons, if belong to the output layer. This neurons gives the user the processed information.

In figure 2.3 the schematic of a general neural network it is possible to visualize the schematic of a general neural network with an undefined number of neurons in each layer and an undefined number of hidden layers.

A single neuron is formed by an input, a weight W and a bias b associated (Independent term). The value of the bias is always 1, but it has an associated weight that make that the value of the bias change. Weight and bias values could be modified. The output of the neuron is associated to an activation function.

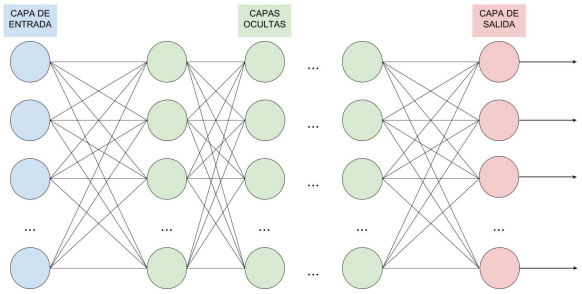


Figure 2.3: Schematic of a general neural network.

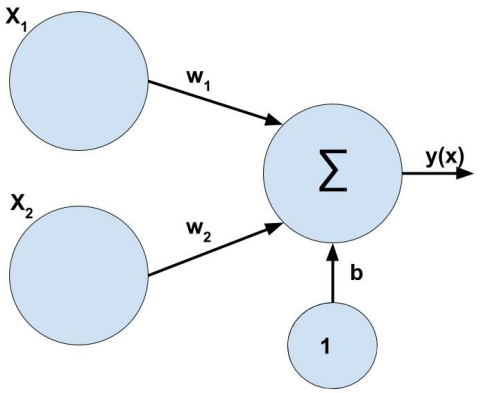


Figure 2.4: Simplest neural network architecture.

The simplest example of a neural network is formed by two inputs neurons and a output neuron as it is shown in figure **??** in which $W = [w_1, w_2]$ are refered to the weights assigned to each neuron. The b value is refered to the bias term. $X = [x_1, x_2]$ are the input of the network architecture and *y(x)* the output. The output is predetermined by the equation 2.1. So, the output would depend on the input, weights and bias sand the activation function (or transfer function) F [14].

$$y(x) = F(\sum_{i=1}^{2}(w_i * x_i + b)) \tag{2.1}$$

Due to activation function, the output of the neuron would change if the input is bigger than a some threshold. This threshold is defined by the own activation function. There are various functions that are used: sigmoid function, Heaviside function, Fermi function or hyperbolic tangent among others. In figure 2.5 (which has been obtained from [11])

the mentioned function are shown and the output would be 0 or 1 values if the function is sigmoid or Fermi and the output would be -1 or 1 if the function is an hyperbolic tangent.

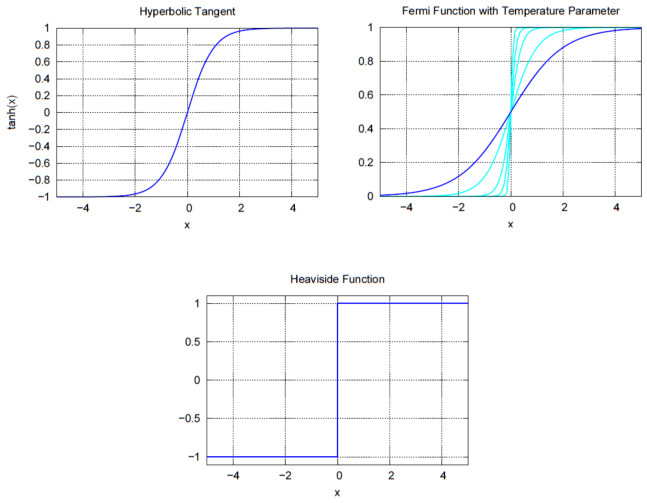


Figure 2.5: Different activation functions. Image obtained from [11]

Analogies between ANN and Biological Neural Networks

Analogies between Artificial and Biological Neural networks could be found (if both basic theory is known) because the first one (artificial NN) are based in the second one. It is easy to see that biological neurons correspond to artificial neurons. But the cell body correspond with transference function. The output to others neurons would correspond to the axon and synapses with weights and bias. The soma would correspond with the transfer function. Those analogies are summarized in table 2.1. Artificial neural networks are just inspired in Biological neural networks, that both work in a different (although inspired) way must be made clear.

Artificial Neural Component	Analogy
Neuron Artificial Neuron	Biological Neuron
Transfer function	Soma
Connexion between Artificial neurons	Axn
Weight	Synapses

Table 2.1: Analogies between artificial neural networks and biological neural networks

Learning

Neuronal networks are able to learn features from the input to classify or just extract features of the input data. In order to know how to get the desired output, a learning process is needed. The learning process what really do is modifying weights and bias values for each layer with the purpose of getting a better output [15].

There are three main different learning procedures that defined following [15, 16]:

Supervised Learning: the input training data is given with its target (correct result of its corresponding sample), that means that the train samples has associated the desired output. The learning consist in adjust the weights and bias until the output of the network is the same or the closest value as the target.

Reinforcement Learning: in the training, the correct target is not provided, but just a grade is given to the network.

Unsupervised Learning: there is not help in the training, no target and no grade, just the input data. The networks learn to cluster the input data and the weights and bias changes in function of the bias.

All the learning procedures in neural networks have in common the modification of weights ans bias to learn and obtain the desired output, so the error at the output is minimized. The most useful learning procedure is the supervised learning.

The gradient descend procedure is an algorithm whose goal is minimize the error of a function *J.J(a)* would be the minimum if a is the solution. To get the solution the gradient of J ($\nabla J(a)$) is calculated for each value of a: $\nabla J(a(1))$ is obtained and *a(2)* is obtained moving some distance from *a(2)*, in steps called *learning rate* η, in direction of the negative gradient. The *a(k+1)* value would be: [16].

$$a(k+1) = a(k) - \eta(k)\nabla J(a(k)) \tag{2.2}$$

One of the most used supervised techniques of neural network learning is the back-propagation, this learning rule is based in the gradient descend. In general, this rule, from a randomize initialization of weights, the error would be minimized changing the weights in the direction of the gradient descend:

$$\delta w = -\eta \frac{\partial J}{\partial w} \tag{2.3}$$

Being $\frac{\partial J}{\partial w}$ The gradient of J in function of weights w.
The error is calculated as defined in equation 2.3.4

$$E = \sum_p E^p = \frac{1}{2}\sum_p (\delta^p - y^p)^2 \tag{2.4}$$

The error is backpropagated to the last layers until the first layer, modifying the weights to balance the error proportionally to the gradient of the error function. This is called the chain rule or the delta rule [11,14,16]. The backpropagation equation is defined below

$$\Delta_p W_{jk} = eta\delta_k^p y_j^p \tag{2.5}$$

Being k the unit which receives the input and j the output.

There are three useful training protocols depending on how training set is used:

Stochastic training: Samples are choosen randomly and for each sample, weights are updated.

Batch training: All training samples are used to the learning process.

on-line training: There is no memory, as the same time as samples are received, they are used once to train.

Type of Layers

The layers that conforms the networks could be different depending on the mathematical operation. The most common used layers are described below:

Convolutional layer: In this type of layers, the convolution operator is processed at the input. Weights are the applying filters and the output are the features or characteristics useful to extract the information of the image. In one convolutional layer, there could be used as many filters are user defines.

Pooling Layer: In this layer, the dimensionality is reduced. The most important information is preserved. It is usually used at the output of the convolutional to resume its out [17]. The most used pooling layer is the max-pooling layer, the maximum values are saved.

Normalization layer: This layer normalize the activities of the neurons, because of that the processing time could be reducing during training or testing. There are two types of normalization layer: Local Response Normalization (LRN) and Batch Normalization.

Dropout layer: The ouput of a network could rely on the output of an spefici neurons being this a overfitting cause. To prevent it, during training, some neurons are 'turn off setting its value to 0. In this way, neurons are more adaptative. Dropout is a spefic layer that manage this. In stead of *eliminate* neurons, if it is possible that weights are put to 0. In this case, the layer would be **"DropConnect"** [17].

Fully-connected layer: Fully connected layer is vector in which the input is connected to each nenuron that conforms this layer. This layer is the high-level reasoning layer so, it is used before the classification.

One neural network is defined by its type of layers and the number of neurons, also, there are parameters that also affect to the behaviour of the network (learning rate,...) [18].

2.4 Convolutional Neural Network (CNN)

Convolutional neural network are a determinate typology of neural network, CNN are inspired in how the visual cortex of a cat works [17]. This type of neural networks are (usually) used with images at the input of the network.

Convolutional neural network has been used successfully in recognition and classification task includying document and object recognition, face detection, robotics navegation... [18, 19].

2.5 Programming language and frameworks

2.5.1 Python

Python is a object-oriented programming language and it is used to develop the experiments necessaries is Python, more specifically, it has been used the 2.7 Python version.

2.5.2 Theano and others frameworks

Theano is the main framework used to develop the deep learning code. But others libraries has been used to, NumPy,Scikit-learn and matplotlib are the most relevant. In addition to this, other Python packages has been used to like Pickle.

Theano

Theano [20] is a Python library that allows users to work with mathematical expressions and work with simbolic variables, moreover, Theano handles multidimensional arrays efficiently. This framework has been used in order to build convolutional neural networks architecture and its training procedure.

There are numerous open-source deep-libraries that have been built on top of Theano, for example Keras, Lasagne and Blocks. Although the usability of using those libraries

in stead of Theano is bigger, Theano is more flexible when user wants to develop its own layer, for example.

Theano is not the unique language oriented to deep learning, for example Google, has developed its deep learning language called TensorFlow; Caffe and Torch are others examples.

Theano main page is available in http://deeplearning.net/software/theano/ where the documentation, examples,.. could be found.

NumPy, Scikit-learn and Matplotlib

NumPy is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. The documentation of this library is available in https://docs.scipy.org/doc/numpy/.

Scikit-learn is an open source Python library and commercially usable that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms which has been build in top of Numpy and matplotlib. Its documentation is available in the following url http://scikit-learn.org/dev/index.html.

Matplotlib is a python library for 2D plotting.

2.6 Databases

Some databases has been used in order to learn, compare results and carry out the project. All the databases are formed by three subsets whose samples are not repeated among subsets: train, validation and test.

- The training subset is used to train the network during epochs. To know how the training behavior, a cost is calculated.

- The validation subset is used to check the behavior of the network while is training, also, the validation subset is usually used to calculate the hyper-parameters of the network, although the hyper-parameters are not calculated until is pointed. The validation error is calculated for each training epoch. The metric use to the validation is $error(\%) = cost * 100$.

– The test subset, that is used just at the end of the training. The best model is chosen with regard to the best validation error. Different metrics are going to be used for after testing the network (error rate(%), TP, FP ...).

2.6.1 MNIST digit database

MNIST digit database is a image database of human written digits. This database is commonly used to learn machine learning techniques. Because of that, this database has been used, in this thesis, for learning Theano and convolutional neural networks. In addition, this database has been used in a implemented convolutional neural network (LeNet).

Some examples of the digit image MNIST database could be seen in 2.6, image obtained

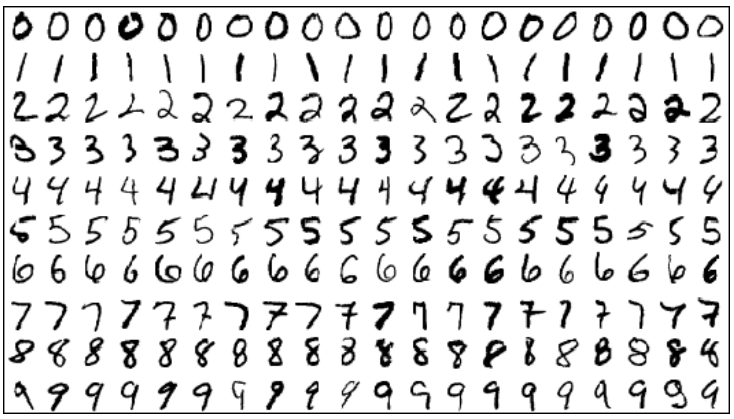


Figure 2.6: MNIST digit images database. Image obtained from [21]

from [21] and the characteristics of this database are the following ones:

– There are 70.000 number of unique samples.

– Altough original size of the database is 32x32 pixels, the samples of this downloaded database are 28x28 pixels in gray scale, that is 784 features per image.

– 10 classes could be differentiated, one per digit.

– The samples are directly separated into train, test and validate subset.

2.6.2 Labeled faces in the wild

The Labeled Faces in the Wild (LFW) is face database of well-known people that are collected from the net. The database can be found in its official web page http://vis-www.cs.umass.edu/lfw/ [22].

The characteristics of this database are the following ones:

- There are 13233 unique samples.
- The size of each image is 250x250 pixels. The number of features per image is 187500.
- There are images from 5748 different people, so there are 5748 different classes (if is used as one class per people).
- The number of images per person is not the same for each one. There are 1680 people with two or more images.
- The images are in RGB space.
- The faces are centred in the image.

Figure 2.7: Samples of LFW database

Four examples of the images of this database could be seen in figure 2.7 in which faces of well-known people are visualized.

This database has been used to learn; to learn how to read a database, how to feed the network with those images. It has been used assigning each class to a different people.

2.6.3 FRAV dataset

FRAV database is an anti-spoofing face database built in the FRAV research group of tu URJC University and which is part of the Automated Border Control Gates for Europe project [23].

One exaple of RGB images of FRAV database are shown in figure 2.8 and another example could be seen in 2.9. In both images, the four attacks described previously and the real user could be visualized.

(a) printed image attack (b) mask attack (c) eye mask attack (d) tablet attack (e) real user

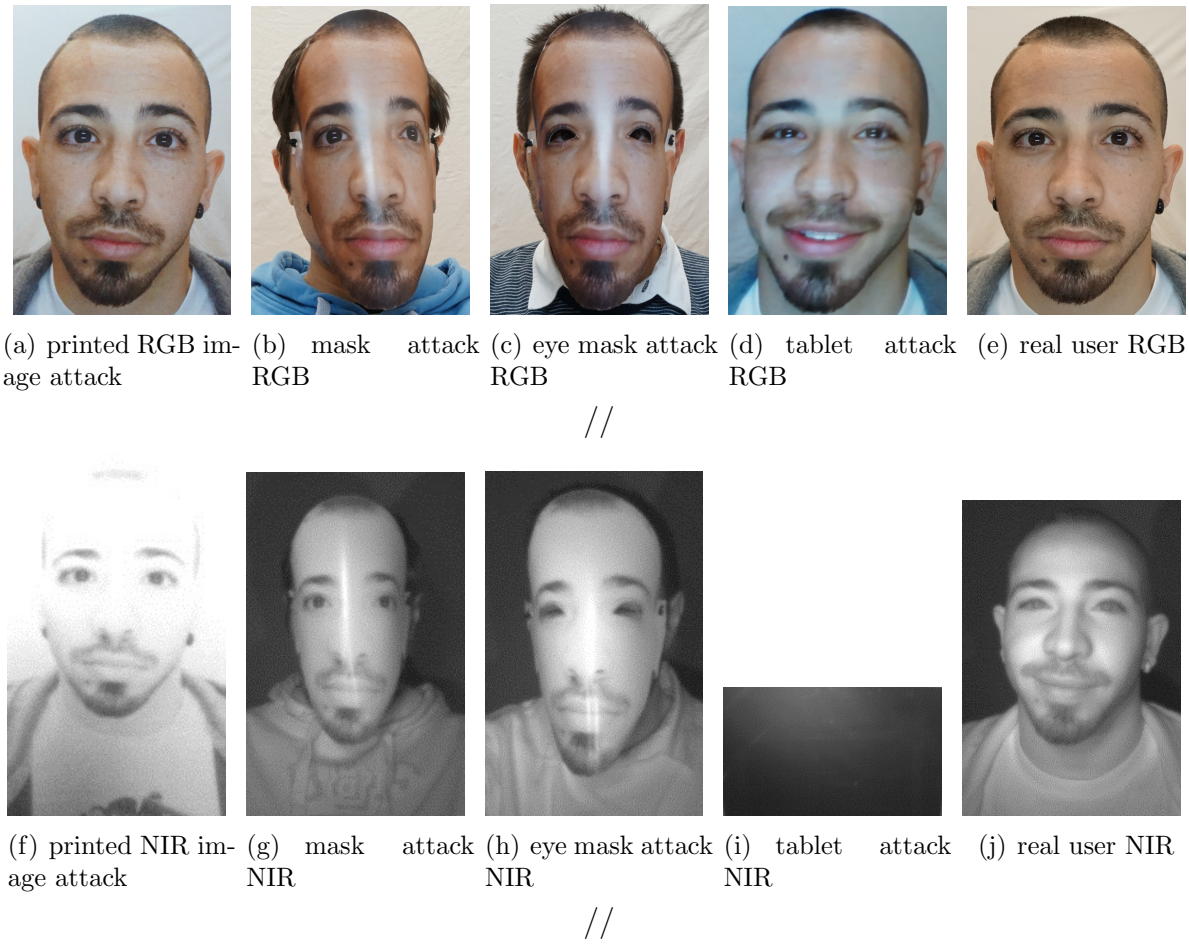
Figure 2.8: Four attacks and real user from RGB FRAV database

As could be seen in figure 2.8 and figure 2.9, five different classes composes this database. One class is the real user class and the other four classes are four spoofing attacks per user:

- Original images of people represented in figure 2.8(d) and figure 2.9(e) for RBG images and figure 2.9(j) for NIR image.
- Images of people printed (attack) represented in figure 2.8(a) and figure 2.9(a) for RBG images and figure **??** for NIR image.
- Images of people with a mask (attack)represented in figure 2.8(b)and figure 2.9(b)for RBG images and figure 2.9(g) for NIR image.
- Images of people with a mask with the eyes cropped (attack)represented in figure 2.8(c) and figure 2.9(c) for RBG images and figure 2.9(h) for NIR image.
- Images of people in a tablet (attack)represented in figure 2.8(d) and figure 2.9(d) for RBG images and figure 2.9(i) for NIR image.

Images of classes can be found in RGB and NIR (not all RGB images has its corresponding NIR image). Characteristics of FRAV images database are the following ones:

- There are 939 people in each RGB class or 195 in each NIR class.
- There is one image per person.
- Each image has it is own shape.



(a) printed RGB image attack (b) mask attack RGB (c) eye mask attack RGB (d) tablet attack RGB (e) real user RGB

//

(f) printed NIR image attack (g) mask attack NIR (h) eye mask attack NIR (i) tablet attack NIR (j) real user NIR

//

Figure 2.9: Four attacks and real user of RGB and NIR FRAV database

– As it real user has all the four attack, all the classes has the same number of samples.

– The faces are centered in the image.

This database is used in two different ways:

1. Using only RGB images (figure 2.8), where there are 933 people, in which each person would have a genuine image and four attack, so 4665 samples are available.

2. Using the RGB and NIR images, where there are 195 people which correspond with NIR images and its corresponding images in RGB. 975 samples are available.

If RGB and NIR (figure 2.9)images are used at the same time, two different methods, for using both types of images together, are used:

– Characteristic level: adding the NIR image as another layer to RGB image, so the resultant image have hightxweightx4 dimensions (NIR images has one layer because it is a gray scale image and RGB images has tree layers, one per each primary color). The network is feed with the resultant images like other times.

– Classification level: after the network training and before feeding the classifier, RGB and NIR would be trained separately and its features would be appended as the input of the classifier.

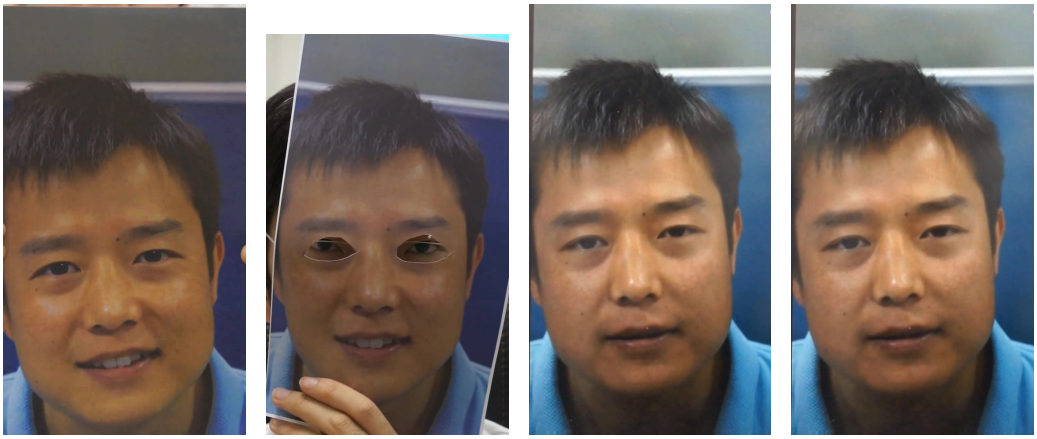
To conclude, this database is going to be used in the three different ways: just RGB images, RGB and NIR images added in characteristic level or classification level.

When the classes are build, two ways are possible to be done: the first one where real people are one class (positive) and the different attacks are other class (negatives), so two classes have been used; and the second way where each attacks correspond with a class, so five classes (4 attacks and 1 real) have been used.

2.6.4 CASIA dataset

The CASIA Face Anti-Spoofing database is a database from the Chinese Academy of Sciences Centre for Biometrics and Security Research (CASIA-CBSR) [24].

A person of CASIA database could be seen in figure 2.10, where three attacks types could be seen (figure 2.10(a), 2.10(b) and 2.10(c)) with the real user (figure 2.10(d)).



(a) printed image attack (b) eye printed image attack (c) tablet attack (d) real user

Figure 2.10: Three attacks and real user from casia database

In the same way as FRAV dataset, this database is formed by real or genuine images of people and three different attacks of the same people:

- Images of people printed (attack) represented in figure 2.10(a).
- Images of people with a mask with the eyes cropped (attack) represented in figure 2.10(b).
- Tablet attack represented in figure 2.10(c)
- Images of real users represented in figure 2.10(d).

Originally, this database is a video database, in which each sample is a different video, but for the experiments developed no videos (entirely or fed directly to the network) have been used. The CASIA database has been used in two different ways:

- Using a single image per person and class. When this database is used, it is going to be referred as CASIA image database.
- Reading three frames per video and saving each frame as a independent image sample. This database is going to be referred as CASIA video database.

The characteristics of the CASIA image database are the following ones:

- There are 49 images per user, so there are 196 unique samples.
- Samples do not have the same size.
- Samples are in RGB space.
- The face of the image is centred.

The characteristics of the CASIA video database are the following ones:

- There are 8 videos per person (two videos for real user, and two per each attack). There are two videos because one is filmed horizontally and the other vertically, one filmed with a smartphone and the other with the frontal camera of a laptop.
- There are 50 different users, so there are 400 different videos.
- For each video 3 frames are read, so there are 1200 unique samples. [**?**]amples are in RGB space. [**?**]aces are centred in the image and blink expression and movement of people are produced.

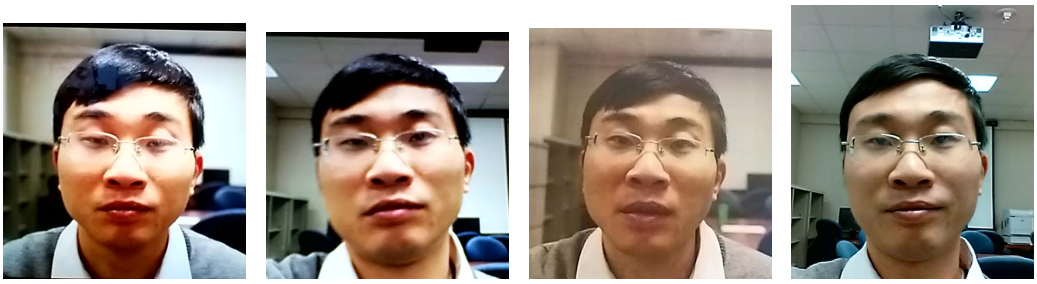
At the time of assigning a class, it could be done using two classes, the positive class to the real users and the negative class to the attacks; If each attack is assigned to a independent class, it would be four different classes, the real user class and three attack classes.

2.6.5 MSU - MFSD database

The MSU Mobile Face Spoofing Database (MFSD) is a video face anti-spoofing database [25].

In figure 2.11 are represented the three attacks and a real user which forms this database:

- Printed photo attack represented in figure 2.11(a).
- Tablet attack where a video is Replayed represented in figure 2.11(a).
- Smartphone attack represented in figure 2.11(a).
- Real user represented in figure 2.11(a).



(a) printed image attack (b) tablet attack (c) smartphone attack (d) real user

Figure 2.11: Three attacks and real user from a person of MFSD database

Originally, the database is a video one, but just one image per user and class is used. The characteristics of the database are the following ones:

- There are 35 images per attack or genuine user. There are 140 unique samples.
- Images are in RGB space.
- Faces are centred in images.
- The size of each image are not equal. Approximately images are 300 pixel heigh and 335 pixel width.

2.7 Metrics

To characterize a system, it is necessary metrics that evaluate it. In this section the used metrics along the thesis are exposed.

Before describing the parameters it is necessary define that the posed problem is bi-class, that means that only two classes would be used:

- Positive class: Are the samples of the real users, the genuine or *bona fide*.
- Negative class: Are the different attacks samples which that pretend to be real users but not.

2.7.1 Cost and Error rate

The first parameter that is used is the cost. The cost is used while the neural network is training, in fact, is the value that must be minimized during the training. The lower value, The better performance of the network.

The cost calculated with the Minibatch Stochastic Gradient Descent (MSGD) is the Negative Log-Likelihood Loss. The MSGD is a variant of the Stochastic Gradient Descent in which the cost is calculated with a mini batch of data, not each sample independently and the Loss is the accumulation [26].

The loss is calculated in the following way:

$$Loss(\theta, D) = -\sum_{i=0}^{|D|} logP(Y = y^{(i)}|x^{(i)}, \theta) \tag{2.6}$$

The error in the validation process is calculated after the logistic regression classification, because is the classifier used during the training process. The error during the testing procedure depends on the used classifier. In both cases, the error represents the number of misclassified samples over the total samples used.

2.7.2 True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN)

If each predicted class is compared with it real target, True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN) values could be calculated. These

metrics are usually used for bi-classes problems.

Those metrics, are gotten when a positive or negative samples is well or misclassified [27]. The classified sample or predicted is compared with it real target.

If a positive sample is classified as positive is a true positive (TP), but if it has been classified as negative is a false negative (FN).
If a negative sample is classified as negative is a true negative (TN), but if it has been classified as positive, is a false positive (FP).

From those four metrics, it could be extracted the confusion matrix for binary classification which is defined in table 2.2 [27, 28]:

Real / Classified	**Positive**	**Negative**
Positve	TP	FN
Negative	FP	TN

Table 2.2: Confusion Matrix

The confusion Matrix resume those four metrics in a simple table. Both the confusion matrix or the parameters individually are widely utilized.

2.7.3 ROC curve and Precision and Recall curve

From the confusion matrix, it is possible calculate others parameters [27]: precision, recall, specificity, accuracy because its values depend on TP, TN, FP, FN:

- The False Positive Rate (FPR) is defined as the proportion of all the negative samples (N) that are classified as positive incorrectly [28]:

$$FPR = \frac{FP}{N} \tag{2.7}$$

  Where:

$$N = FP + TN \tag{2.8}$$

- The True Positive Rate (TPR) is defined as the proportion of all the positive samples (P) that are classified correctly [28]. This parameter could be known as Recall too:

$$TPR = Recall = \frac{TP}{P} \tag{2.9}$$

  Where:

$$P = TP + FN \tag{2.10}$$

– Precision is defined as the proportion of real positive samples that has been classified as positive [27, 28]:

$$precision = \frac{TP}{TP + FP} \tag{2.11}$$

– Accuracy is defined as the proportion of the well-classified samples of all the samples [27]. The classifiers implemented in scikit-learn library return this value as metric.:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.12}$$

The Receiver Operator Characteristic (ROC) curve is the representation how the number of positives samples which has been classified correctly changes with the number of negative samples incorrectly classified. The ROC curve is defined by the parameters False Positive Rate (FPR) and True Positive Rate (TPR) [28].

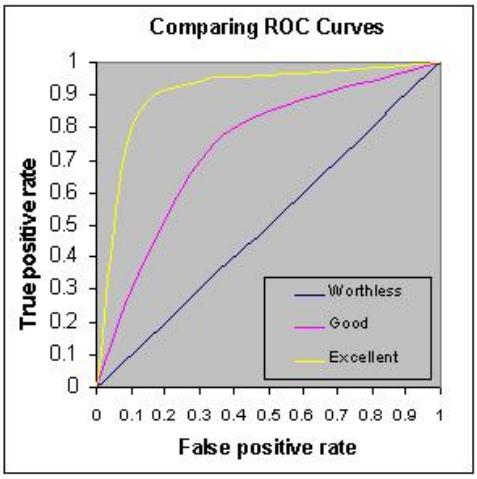


Figure 2.12: ROC curves. Image obtained from [29]

The figure 2.12, obtained from [29], it is shown a ROC graph in which three curves are shown. The yellow one represents a good classification, it is the desired ROC curve, but the blue curve represents a bad classification and it is not the desired result.

From the ROC curve, the Area Under the Curve (AUC) could be obtained, this value is the integral of the ROC curve, and its maximun value is 1 that means a perfect performance of the classifier. If the value of this parameter is lower than 0.7 the classifier performance should be improved significantly.

The Precision and Recall curve is the representation of the Precision and the Recall in the same graph. The desired behaviour of a classification system is a high recall (1)

and a high precision (1) because that would mean that predictions made by the classifier are correct.

2.7.4 APCER and BPCER

The ISO/IEC 30107-3 [30] is the collaboration result of the International Organization for Standardization (ISO) with the International Electrotechnical Commission (IEC).

This ISO defines the terms related to the tests, the reports and the biometric presentation of bioimetrics systems. In addition, the performance methods, specify principles as well as metrics are defined. From this document, the APCER, BPCER and APCER-BPCER curve metrics have been obtained:

Attack Presentation Classification Error Rate (APCER) is defined as the proportion of presentation attacks that has been classified incorrectly (as *bona fide* presentation.)

$$APCER_{PAIS} = \frac{1}{N_{PAIS}} \sum_{i=1}^{N_{PAIS}} (1 - Res_i) \tag{2.13}$$

Bona fide Presentation Classification Error Rate (BPCER) is defined as the proportion of *bona fide* presentations incorrectly classified as presentation attacks.

$$BPCER = \frac{\sum_{i=1}^{N_{BF}} Res_i}{N_{BF}} \tag{2.14}$$

where:

- N_{BF} is the number of *bona fide* presentations
- Res_i is 1 if i^{th} presentation is classified as an attack and 0 if is classified as a *bona fie* presentation.
- N_{PAIS} is the number of attack presentations

The APCER-BPCER curve shows in the same graph both parameters. The ideal system would have a low APCER (0) and a low BPCER (0) because it means that samples are not incorrectly classified.

2.8 Classifiers, Reduction of the dimensionality algorithms and Cross Validation

Classifying is called to the task of sign a a category to a object. The classification task is based in the features of the object obtained of the feature extractor [16], that in the particular case of this thesis is the output of a convolutional neural network.

The output of the convolutional neural network could be bigger enough and some features could not be relevant for the classification. To solve the speed and robustness issues that could appear because of the quantity of features [31], techniques to reduce the dimensionality are used.

Classifiers must be customized to each problem, to find the optimal parameters for each occasion, cross validation technique has been used.

2.8.1 Classifiers

In this section, the classifiers used along the thesis are described.

Logistic Regression

Logistic regression is a probabilistic and a linear classifier. It is customized by a weight matrix W and a bias vector b.

The logistic regression weights and bias define a linear hyperplane which is the decision boundary of the classes. In order to find the parameters, the Maximum likelihood estimation is used during training [32]:

$$\prod_{i=1}^{n} P(y_i|X_i, W, b) \tag{2.15}$$

Given a input vector x, which belongs to the i class (a value of a stochastic variable Y), its probability could be described as follows:

$$P(Y = i|x, W, b) = \frac{e^{W_i x + b_i}}{\sum_j e^{W_i x + b_i}} \tag{2.16}$$

The class of a new sample (y_pred) would be classify as:

$$y_{pred} = argmax_i P(y_i|X_i, W, b) \tag{2.17}$$

That is that the sample would belong to a class depending on position in the space with respect to the hyperplane that separates the classes.

Support Vector Machine

Support Vector Machine (SVM) is a two-class classifier. The smallest generalization error is linked to the *margin* concept. Margin is the perpendicular distance between the closest sample of the database and the calculate hyperplane [33]. An hyperplane is optimal if the margin is the maximum and this margin is calculated (as the same way as logistic regression):

$$\arg\max_{wb} \left\{ \frac{1}{||W||} \min_{n} [t_n(W^T \phi(X_n) + b)] \right\} \tag{2.18}$$

Where w, b are the parameters that should be optimized in order to maximize the distance. t_n are the training samples. ϕ is a fixed feature-space transformation, b is the bias parameter.

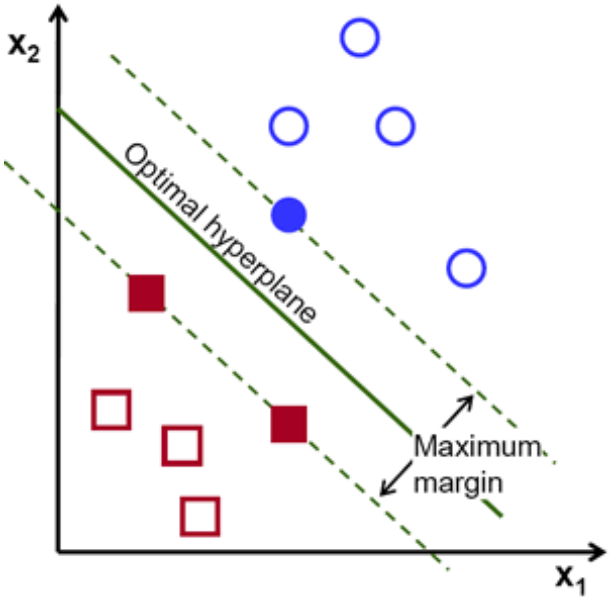


Figure 2.13: Optimal hyperplane and the decision boundary. Image obtained from [34]

In figure 2.13 the optimal hyperplane between two classes are represented with its corresponding margin. In this example the two classes are well differentiated. This image has been obtained from [34].

Based on estimate the hyperplane that the distance between classes, the closest vectors of each class, is maximized [33, 35]. In practice, the margin is determined by C, a parameter that should be chosen by user to get the optimal margin.

The SVM performance is join to a kernel function which allows variability in nonlinearity and flexibility in the model [32, 36]. There are many kernels (polynomial, sigmoid...), but the two used are described:

1. linear: $K(x_i, x_j) = x_i^T)x_j$.
2. radial basis function (RBF): $K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2), \gamma > 0$

K Nearest Neighbours

K- Nearest Neighbour (KNN) is a generative and non parametric classifier. For classifying, the density estimation procedure is used. The difference between this classifiers and others used is this algorithm uses the data directly for classification, without building a model first [32]. The density function is determined by the form [33]:

$$p(x) = \frac{K}{NV} \tag{2.19}$$

Where K is the number of points inside the region R whose volume is V and N is the number of total samples or observations.

This classifiers uses the observation directly to classify and needs all the samples to predict a new one. The probability of a sample x belonging to a class C_k is defined by [33]:

$$p(x|C_k) = \frac{K_k}{N_k V} \tag{2.20}$$

Where N_k are the observations of a class C_k and K_k of it class points are contained in the volume V.

The K value is fixed, should be calculated and optimized by user of each application.

Decision Tree

Decision Tree classifier is based in a natural classification based in a sequence of true/false or yes/no questions [16]. It could be used as a binary classifier or with k classes.

The input data is split to maximize its separation, resulting a tree structure [32] as is described in figure 2.14 (image obtained from [37]). Where depending on the features, a sample changes from a principal branch to a branch of this until a class is signed. The last branches correspond to the classes and the same class could be in different final branches.

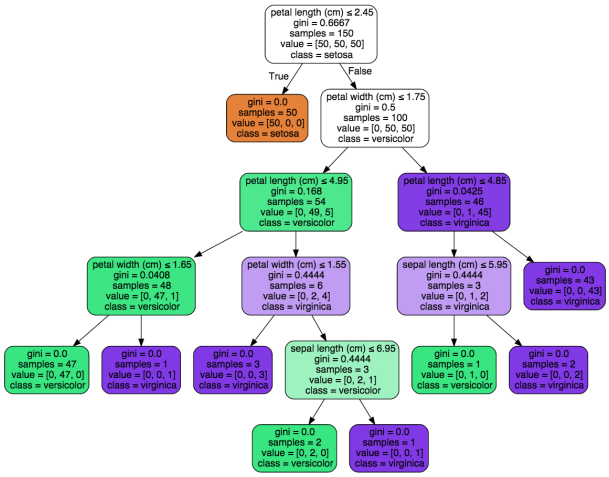


Figure 2.14: Decision Tree Classifier. Image obtained from [37]

2.8.2 Dimensionality reduction algorithms

The objective of those algorithms is transform the characteristic vector into another characteristic vector but whit a lower dimensionality. Linear methods, that projects the dimensional data onto an another space whose dimensionality is lower [16], have been used. The two techniques, the most common used, are described and used along the thesis.

Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) looks for the vectors in the space that best discriminate among classes, that is that LDA pretend to maximize the between-class meause (equation 2.8.2) at the same time as the within-class measure (equation 2.8.2) is minimized [31].

$$S_w = \sum_{j=1}^{c} (\mu_j - \mu)(\mu_j - \mu)^T \tag{2.21}$$

$$S_w = \sum_{j=1}^{c} \sum_{i=1}^{N_j} (X_i^j - \mu_j)(X_i^j - \mu_j)^T \tag{2.22}$$

The data of d dimensions would be projected onto a s dimensions and being $s < d$. The minimum value that s could take would depend on the number of classes n: $s \geq n-1$.

Principal Component Analysis

Principal Component Analysis (PCA) uses a subspace *t)* in which the the variance direction among basis vectors is maximum in the original space (f) [31]. PCA faces the problem of reducing the n dimensional samples vector to a single vector *X_0*. The *X_0* vector would be the result of the sum of the squared distances between *X_0* and various features *X_k* is the smallest [16].

The new subspace is usually smaller than the original space [31].

The linear transformation from one space into another would be denoted as W, and its columns are eigenvalues which has eigenvectors associated.

The feature vectors (y) from the f space would depend on W:

$$y_i = W^T X_i, i = 1, ..., N \tag{2.23}$$

Being N the total number of feature vectors.

2.8.3 Cross Validation

Classifiers are defined by certain parameters. For example,the number of neighbours (k) in KNN classifier is a value that user have to determine. In this case is useful use Cross Validation to determine the value of k.

This method uses the training samples. They are split in groups (*kfolds*) and used to train and test the classifier with different values; in the KNN case, the value k would be change. A metric (score) is calculated and it is possible to determine the value of the classifier in which the metric is the optimum.

This technique has been use to calculate the value which a classifier is defined. For SVM classifier the value C, for KNN classifier the value k, for Decision Tree classifier the depth of the tree, softmax classifier to determine the learning rate when it has not been trained at the same time of the network and for PCA and LDA the number of components.

Chapter 3

Methodology

3.1 LeNet-5

LeNet-5 [38] is the name of a certain architecture of a convolutional network designed for document recognition (handwritten, machine printed characters) developed by Yan Lecun *et al.*

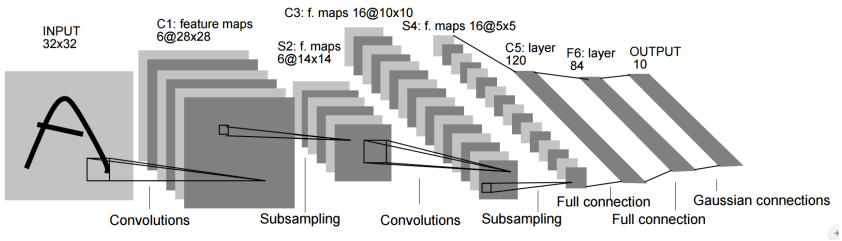


Figure 3.1: LeNet-5 Arquitecture

The basic architecture of LeNet-5 is two convolutional layer, followed each one by a max pooling layer and then a fully-connected layer. This architecture could be visualized in figure 3.1 where it is possible to visualize the input image dimensions across the layers and its final shape.

LeNet is a useful convolutional neural network that is usually used by beginners users to learn deep learning matter because its short architecture and it is implemented in lots of deep learning framework using it to explain the framework. Because of this, LeNet-5 has been used as basis of the project and to learn Theano and convolutional neural networks theory and implementation.

The code of LeNet in Python using Theano library, and its explanation, is openly available in www.deeplearning.net.

3.1.1 LeNet-5 specifications

The specifications of the downloaded LeNet code are the architecture of LeNet-5 used for starting to work with this project is formed by two convolutional layers of size 5x5 and with 20 kernels in the first convolutional layer and 50 in the second one, those are followed (each one) by a max pooling-layer of size 2x2. Those four layer are followed by a fully-connected layer with 500 neurons at the output.

The classifier which has been used is the logistic regression which is trained at the same time as the convolutional neural network. The activation function of the convolutional layers and the logistic regression is tanh. The learning rate used is 0.1 and the network runs by 200 epoch.

The cost function or loss that must be minimized during the training is the negative log-likelihood. LeNet-5 uses the stochastic gradient method with mini-batches (MSGD).

The data used is MNIST digit database, whose characteristics are described in section 2.6.1. The data comes split in three subsets: training, testing and validating. Each subset is used for training, testing or validating respectively.

The data is not fed to the network in one go, each subset is grouped in smalls subsets called batches and whose size is chosen by user. In the code available of LeNet-5 the batch size is 500 samples. The network is fed by batches, so the size of the net depends on the batch size not the (train, test or validate) subset. In this example, the batch size, is the same for the three subsets. When the subset is divide into batches, if there are some samples that are not enough for a batch, those samples are not used.

The network train for a specified number of epoch. Each epoch has as many iterations as necessary to go through all batches of the train subset. The reason of using batches is to define the size of the network and because, usually, the quantity of samples used in deep learning is big (thousand, millions..) and too much memory would be need to build a network of its size and the computational resources available may not be enough.

As the MNIST digit database available with the code has 50000 samples for training, 10000 for testing and 10000 for validating, the number of batches for each subset (with 500 samples for each batch) is 100 for training, 20 for testing and 20 for validating

The training procedure is being realized for too many epochs as users has selected and returns the cost of the procedure. While the trianing is being running, the validation is calculated for each epoch and the validation returns the error of the procedure. The training cost and the validation error are used to know the behavior or the learning

process of the network and how it generalizes with the purpose of choosing the best model.

The test is realized while the training is being executed, more specifically, when the validation has been realized and the results are the best obtained in the whole process until that iteration. The result that is used to compared with others classifiers or with others articles.

In addition to the number of epoch, other way to stop the training procedure is early-stopping, this method is used to avoid over fitting tracking the validation process [39]. The decision of stopping the training depends on *the patience* and is chosen by user.

3.1.2 LeNet-5 Results

While the training is being calculate, the weights are being update in each iteration. When A model is selected or saved, weights are actually what is being chosen or saved. An example of weights is represented in figure **??**here twenty first weights at epoch 10 of the first convolutional layer are represented. So when its being trained, what the network is doing is adapting the weights to the input to get a good performance.

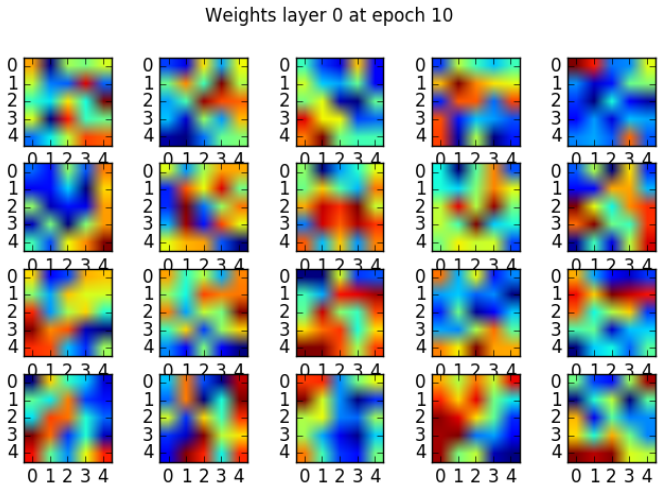


Figure 3.2: Weights at epoch 10 of the first convolutional layer

The training procedure returns the cost function in each iteration to evaluate the training behavior. The cost function at training obtained executing LeNet-5 is repre-

sented in figure 3.3, and its value decreases as the iterations rise converging in almost 0; this curve is the desired one for each training practice, because it is not oscillate abruptly and converges in a very low value logarithmically.

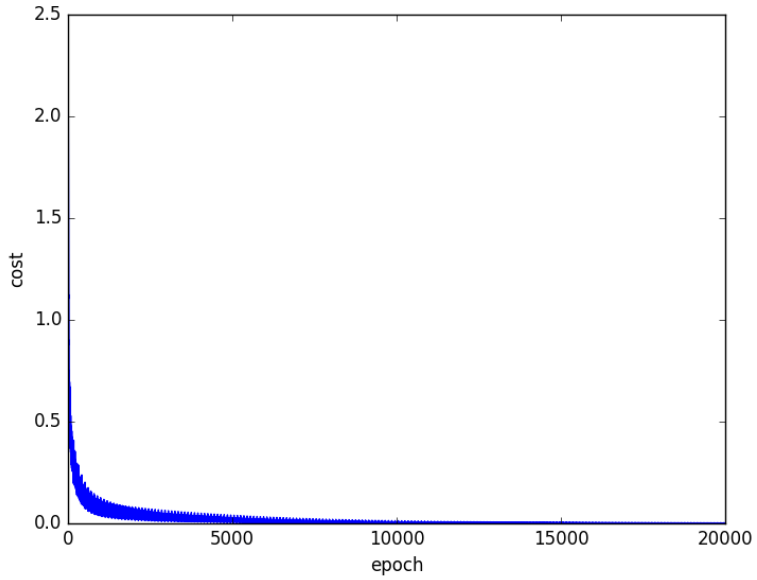


Figure 3.3: Cost function at training running LeNet-5 with MNIST digit database.

The error obtained at validation is represented in figure 3.4, where could be seen that the value decreases logarithmically too converging, approximately, in 1 (a low value) and this behavior of the curve is the desired in a validation process. The convergence value of the validation is not usually as much lower as the training convergence value, and the point where it starts to converge is later than in the training.

The test result has been calculated with the model of the iteration 18300 (epoch 37^{th}), with a validation error of 0.91%. The error rate obtained is 0.92% what it means that 920 samples of 100000 of the testing subset are being misclassified.

3.1.3 Modifying LeNet

Modifications has been made to LeNet-5 architecture. First the batch size has been changed and then the activation function, a normalization has been added o the weight initialization. The database used for those experiments is the MNIST digit database.

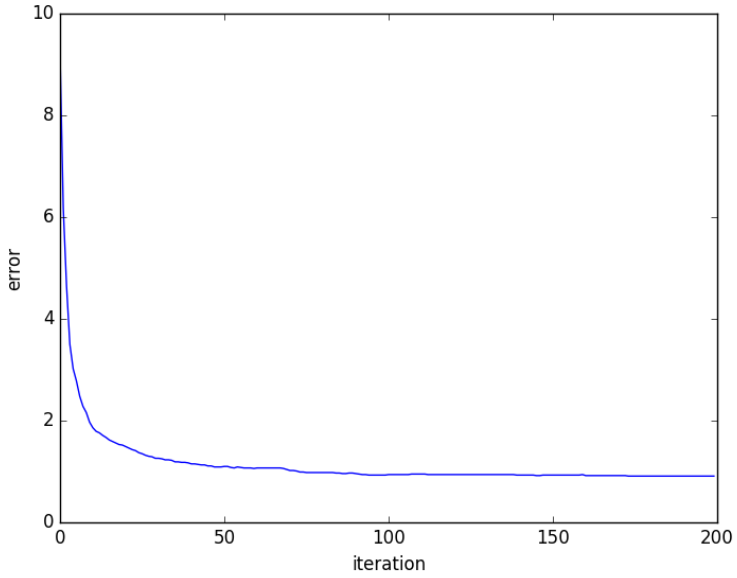


Figure 3.4: Validation error obtained with LeNet-5 with MNIST digit database

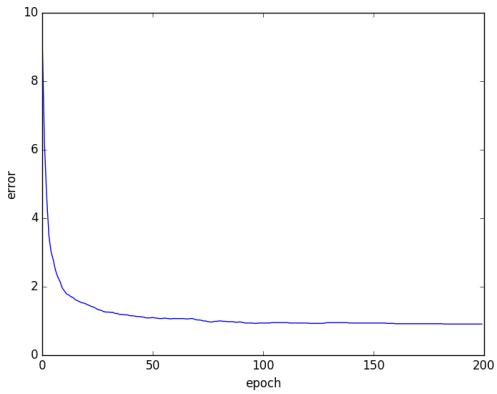
Changing the batch size

Two experiments have been developed in order to compare the results when the batch size is changed and how the network behavior change too with respect to LeNet-5 with the original batch size (500).

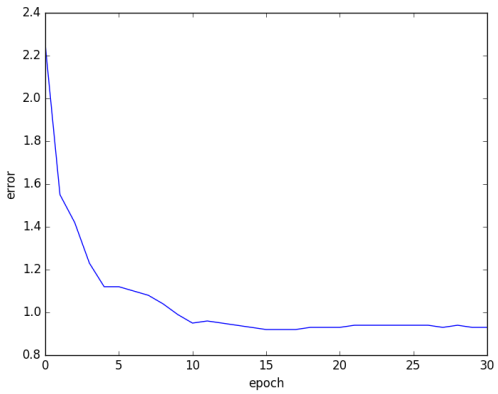
The first experiment is with 20 samples per batch. For the second experiment, the batch size used is 100. In both cases, the training process has been stopped by the early-stopping; at epoch 31^{th} has stopped the first experiment, because from epoch 16^{th} the error at validating was not being improved and for the second experiment, at 33^{th} epoch is when the early-stopping has finished the training.

The validation error for both experiments and the original LeNet are represented in figure 3.5. From images could be seen that the validation error in first epochs is lower (9 % in the first experiment) than the validation error when the batch size is bigger. Whith the batch size equal to 20, the optimal test error rate has been obtained in the first 15^{th} epochs, the same error rate than in the original case (0.92%), but for 100 samples per batch, it has not been possible to get to that error rate, the best test error rate has been 1.04% at iteration 8500.

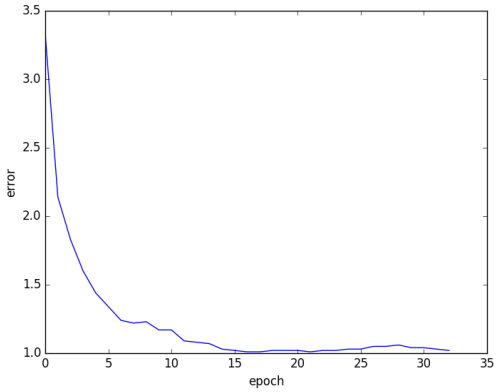
Concluding, more epochs are necessary when the batch size is bigger because there are not enough updates in each epoch [39]. Usually, the value of the batch size used is



(a) batch size = 500 (original size)



(b) batch size = 20



(c) batch size = 100

Figure 3.5: Validation error in each epoch for different sizes of batches.

32 [39] and the choice, generally, is computational.

In figure 3.5 the error in each epoch is represented for 500 batch size, the original size, for a value of 20 and 100. In the original case, the error stars with a value of 9% approx. with the batch size = 20 the error in the first iteration is about 2.4%, and with a bunch of 100 images, the validation error is 3.5%.

With the original size and size equal to 20, it is possible to get to the same minimum, the difference between those examples is that each one get to that conclusion into different epochs. With a batch size equal to 100, the code stopped because of the early-stop with a patience of 10000; it stop in epoch 33,while those epochs, it has been possible to get to a test error of 1.04% in iteration 8500 when the validation error was 1.01%, it has been running with put getting a better validation score for 17 epoch. With a batch size = 20,

the code also has stopped earlier because of the same reason, but in that case it has been possible to get to the same minimum that with the original size; the epoch in which has stopped is 31, it has been running without getting a better validation score for 15 epochs.

Changing activation function, normalization and weights initialization

As the same way as the batch has been changed and its results has been compared in the previous subsection, the activation function has been changed, a normalization layer has been added and weights initialization has been changed.

LeNet-5 does not use any normalization layer, but in this experiment from the different normalization availables (batch normalization, local normalization, ...) Local Response normalization has been added after the max-pooling layers.

The activation function used in LeNet-5 is tanh, it has been changed to rectified linear unit (ReLu) activation function.

With respect to the weight initialization, in LeNet, for the convolutional layers and the fully connected layer, a normalized initialization [40] is used:

$$W \sim U[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}] \tag{3.1}$$

Where n_j is the number of neuron of the current layer and n_j is number of neurons of the following layers. In this experiment, this initialization has been changed to a weight initialization with a Gaussian distribution.

Below the details of each experiment are described:

- Experiment 1: using Local Response Normalization (LRN) In which a normalization has been carried out in the convolutional-max pooling layers.
- Experiment 2: using ReLu as activation function: The activation function tanh has been substituted by ReLu activation function in convolutional and fully connected layers.
- Experiment 3: using ReLu and LRN: The activation function used is ReLu and LRN has been used as normalization layer.
- Experiment 4: changing weights initialization: Weights initialization has been changed by Gaussian. In which mean value that has been used is 0 and std is 0.01. Weights initialization has been changed in convolutional and fully connected layers. Also, bias initialization has been changed by ones.

First, the cost of training process are going to be visualized with the original cost train, without modifying LeNet). In figure 3.6 is represented. Also the validation error (validation cost*100) could be visualized in figure 3.6. The network has been running for 200 epochs.

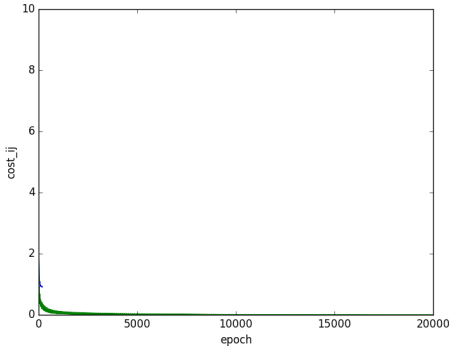
Visualizing the loss during the training, could be affirmed that the network with Gaussian initialization is in a local minimum because the loss has converged as could be seen in figure 3.6(e). The loss of LeNet without being modified (figure 3.6(a)) is the one whose oscillation at training is less than others.

About the error at validation, visualizing the graphs in figure 3.6, it is very similar the curve for original LeNet-5, LeNet-5 with ReLu, LeNet-5 with LRN and LeNet-5 with LRN and ReLu.

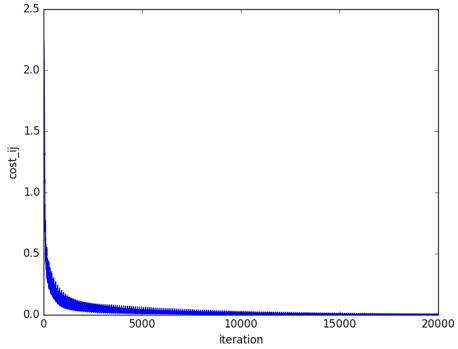
The results obtained, at testing, have been the following ones:

- Original LeNet: Best validation score of 0.91 % obtained at iteration 17400, with test performance 0.92%.
- Experiment 1: using Local Response Normalization: Best validation score of 0.99 % obtained at iteration 13400, with test performance 1.6 %.
- Experiment 2: using ReLu as activation function:Best validation score of 1.04 % obtained at iteration 11900, with test performance 2.4%.
- Experiment 3: using ReLu and LRN: Best validation score of 1.18 % obtained at iteration 19500, with test performance 1.08 %.
- Experiment 4: Gaussian weight initialization: Best validation score of 81.22% obtained at iteration 100, with test performance 80.90%.

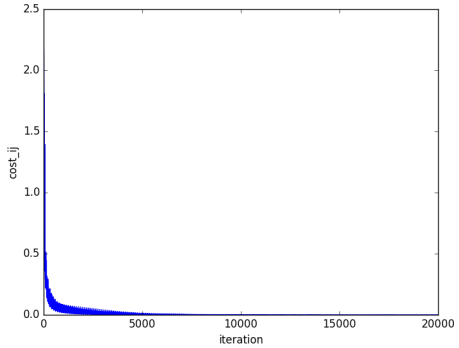
The best configuration for the network is the original one. With Gaussian initialization, the network does not find a local minimum in such a sort of time. Using LRN and ReLu, test result is closer to the obtained with LeNet original, but not as good as the last one. Changing the activation function has not been a good change. Not taking into account original LeNet, the best test performance has been obtained with 1,08% using ReLu and LRN, but the best validation error is 0,99% obtained using just LRN. The values are close of the modifications, but the modification of Gaussian weight initialization.



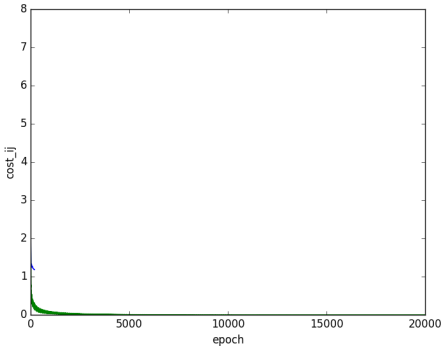
(a) Original LeNet



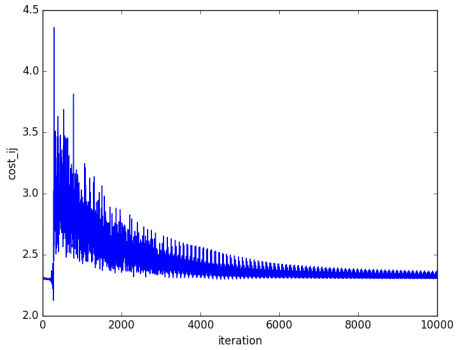
(b) Experiment 1: using LRN



(c) Experiment 2: using ReLu

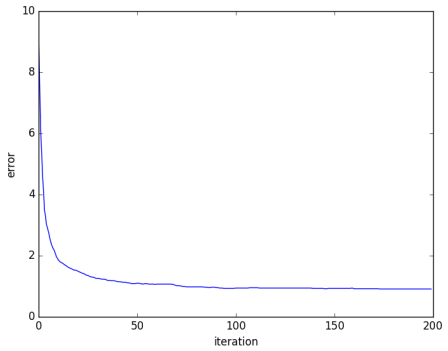


(d) Experiment 3: using ReLu and LRN

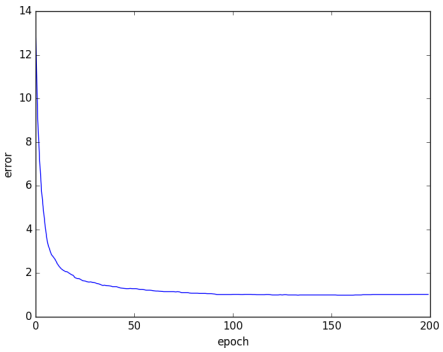


(e) Experiment 4: using Gaussian weights init.

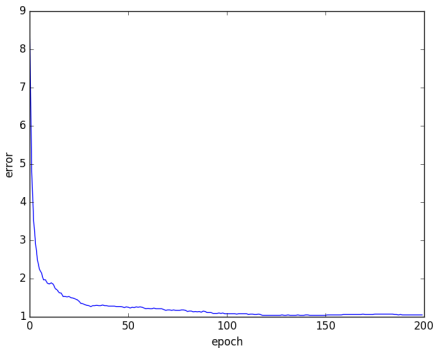
Figure 3.6: Cost function of Lenet and Lenet Modified.



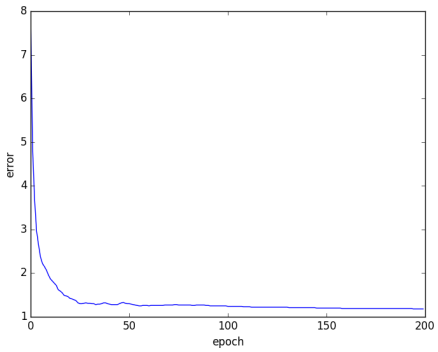
(a) Original LeNet



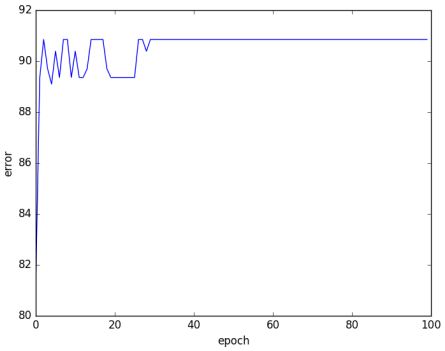
(b) Experiment 1: using LRN



(c) Experiment 2: using ReLu



(d) Experiment 3: using ReLu and LRN



(e) Experiment 4: using Gaussian weights init.

Figure 3.7: Valid error of Lenet and Lenet Modified.

3.1.4 As close as possible as Imagenet

Because of the lack of information about the convolutional neural network described in casia paper, it has been necessary to though the original code that authors use, Imagenet. Authors use the same architecture, although how it has been modified it is not explained.

It has been possible to implement Imagenet, the difference between the implementation and the original one is that the strides used in the first convolutional layer has not been used because the input of the images need to be bigger to have more than one neuron in the last layer.

The convolutional neural network has been tested with different databases: FRAV, CASIA and MFSD and different experiments have been carried out with each one.

Network configuration for each experiment

The configuration of each experiment of each network are described in the following lines. When it is said that Gaussian weight initialization has been used, the mean value is 0 and the std used is 0.01 in all the times ans bias has been initialized to 1, if not, bias has been initialized to 0. When Gaussian initialization has not been used, weights are sampled randomly from a uniform distribution in the range [-1/fan-in, 1/fan-in], where fan-in is the number of inputs to a hidden unit [copy-paste from deeplearning.net].

The goal of the next experiments is getting an optimal architecture changing the parameters. the difference between using a normal distribution o a Gaussian distribution of weights initialization has been tested and using SVM with linear kernel and RBF kernel has been tried.

FRAV database (Common parameters: n_epochs=400, nkerns=[96, 256, 386, 384, 256], batch_size=20):

- frav1: Gaussian weights initialization. SOFTMAX used as classifier. Learning rate = 0.001 Figure 3.8.
- frav_gaussian_initinizialization: Gaussian weight Initialization. Learning rate = 0.001. SOFMTAX used as classifier. Figure 3.9 .
- svm_gauss: Using SVM (with RBF kernel) as classifier. Gaussian weight Initialization. Learning rate = 0.01 3.10.
- svm_genera: Using as a classifier SVM with RBF kernel. Learning rate = 0.01 3.18.

– svm_linear: Classifying with SVM (linear) and Gaussian weight initialization. Learning rate = 0.01

CASIA (images) (nkerns=[96, 256, 386, 384, 256]):
First, four test has been carries out (in which the learning rate has been changed and the number of epochs. trying to get the best learning rate configuration. In four test the batch size used is 25 samples, the classifier used at testing is Softmax and the weight initialization used is normal distribution.:

– Test1: learning_rate=0.01, nepochs=400.

– Test2: learning_rate=0.001, n_epochs=400.

– Test3: learning_rate=0.0005, n_epochs=400.

– TEst4: learning_rate=0.001, n_epochs=1000,

It has not been possible getting a train loss that converge in a minimum in none of the four tests. A good train loss should decrease in each epoch until converge in a minimum. But in the tests, the - casia_gaussian_init: gausiana de pesos con mean 0 y std 0.01. SOFMTAX como clasificador. learning_rate=0.01, n_epochs=400, nkerns=[96, 256, 386, 384, 256], batch_size=20
- svm_gauss: Utilizando svm (rbf) como clasificador, con inizializacion normal. learning_rate=0.01, n_epochs=400, nkerns=[96, 256, 386, 384, 256], batch_size=20
- svm_general: utilizando SVM (rbf) con inicializacion de pesos gaussiana. learning_rate=0.01, n_epochs=400, nkerns=[96, 256, 386, 384, 256], batch_size=20
- svm_linear: SVM (linear) con inicializacion de pesos gaussiana. learning_rate=0.01, n_epochs=400, nkerns=[96, 256, 386, 384, 256], batch_size=20

MFSD (learning_rate=0.01, n_epochs=400, nkerns=[96, 256, 386, 384, 256], batch_size=20):
- svm_gauss: Utilizando svm (rbf) como clasificador, con inizializacion gaussianana.
- svm_genera:utilizando SVM (rbf) con inicializacion de pesos gaussiana
- svm_linear: SVM (linear) con inicializacion de pesos gaussiana

IMPORTANT: The valid graphs are made with SOFTMAX independly of the classifier used to test.

Results with FRAV database

In this section, cost (at training) and error (at validating) is going to be visualized. First when FRAV database has been trained with Gaussian initialization and with a learning rate = 0.001 3.8. Second with the same learning rate, but Gaussian initialization for

weights 3.9. Third, decreasing the learning rate to 0.01 and with normal initialization 3.11 and the last one, whit the same learning rate but with Gaussian weights initialization 3.10.

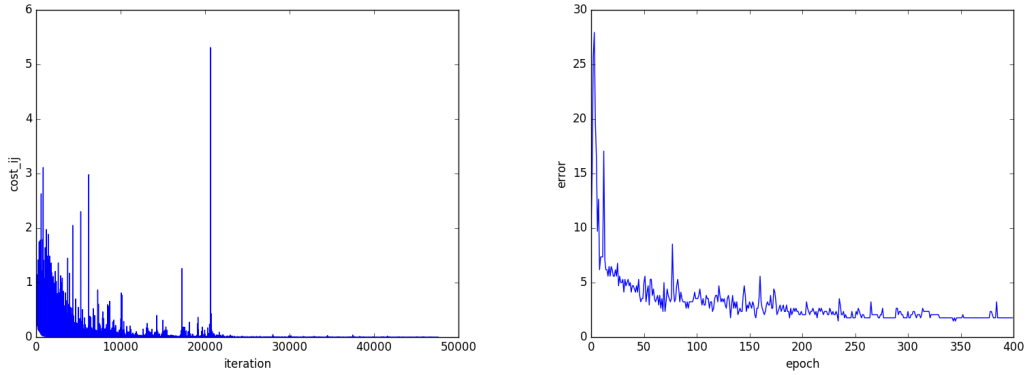


Figure 3.8: Cost at training and error at validating. Normal initialition. Learning rate = 0.001 (frav1).

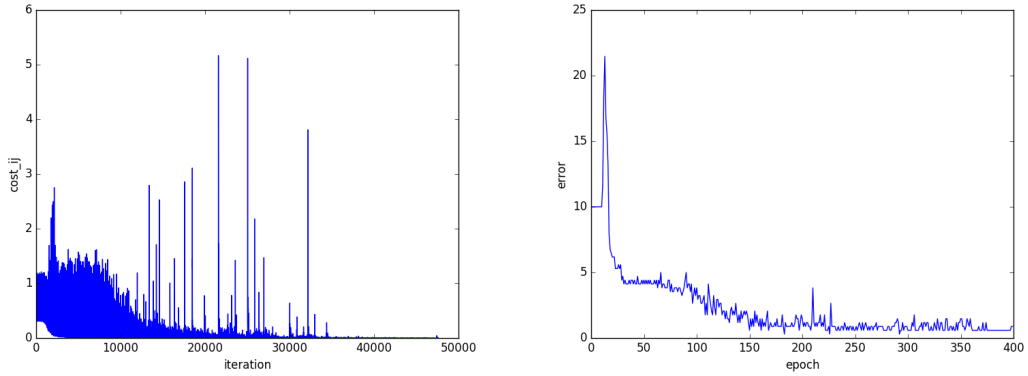


Figure 3.9: Cost at training and error at validating. Gassian initialization. Learning rate = 0.001 - frav_gaussian_init

First FRAV experiment (SOFTMAX as classifier and regular initialization) gives good results. The cost converges to 0 at training, the validation gets a 1.470588 % best error rate with test performance of 5 %. In testing, just 10 samples has been misclassified (7 samples of class 0 and 3 of class 1, from 679 test samples as total. The ROC and precision-recall curves could been visualized figure 3.12

It could be seen in the graphics that the Gaussian initialization does not matter when the learning rate is 0.01, but the cost changes when the learning rate is 0.001. In this

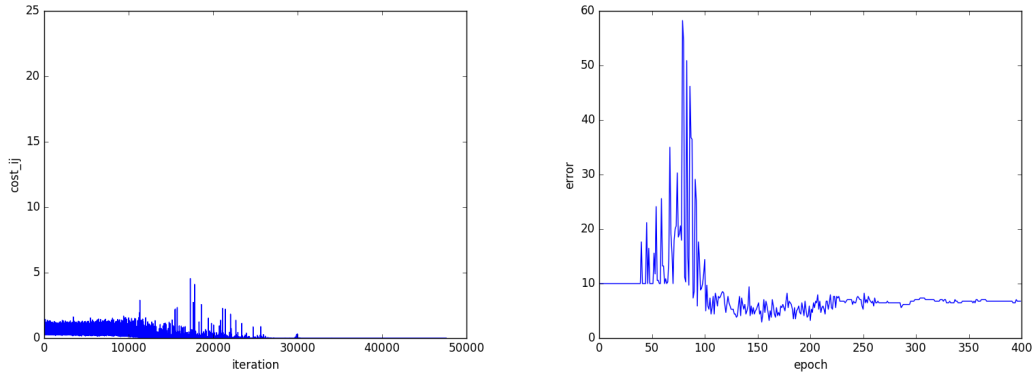


Figure 3.10: Cost at training and error at validating - Gaussian initialization. learning rate = 0.01 frav svm_gauss.

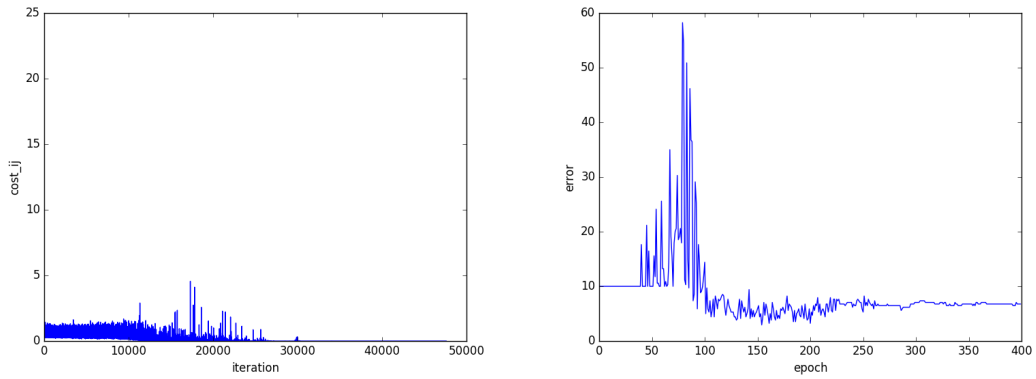


Figure 3.11: Cost at training and error at validating - Noraml initialization. learning rate = 0.01 frav svm_general.

case, the learning rate 0.001 should be the chosen one.

In the table **??** The positive and negative rates are visualized from the different classifier (softmax and SVM) the two different weights initialization and the two learning rates used. The results of the table are the same when the learning rate is 0.01 independently of the weight initialization or the kernel used to classify. The metrics rate are not too bad, with the learning rate the results are worse, the learning rate is too big.

casia results

For Casia, different experiments has been carried out in order to train the network, using SOFTMAX as classifier and normal weight initialization:

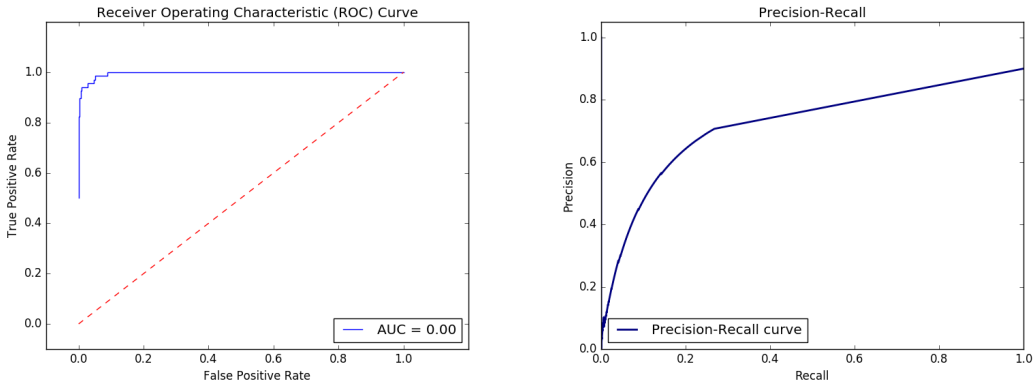


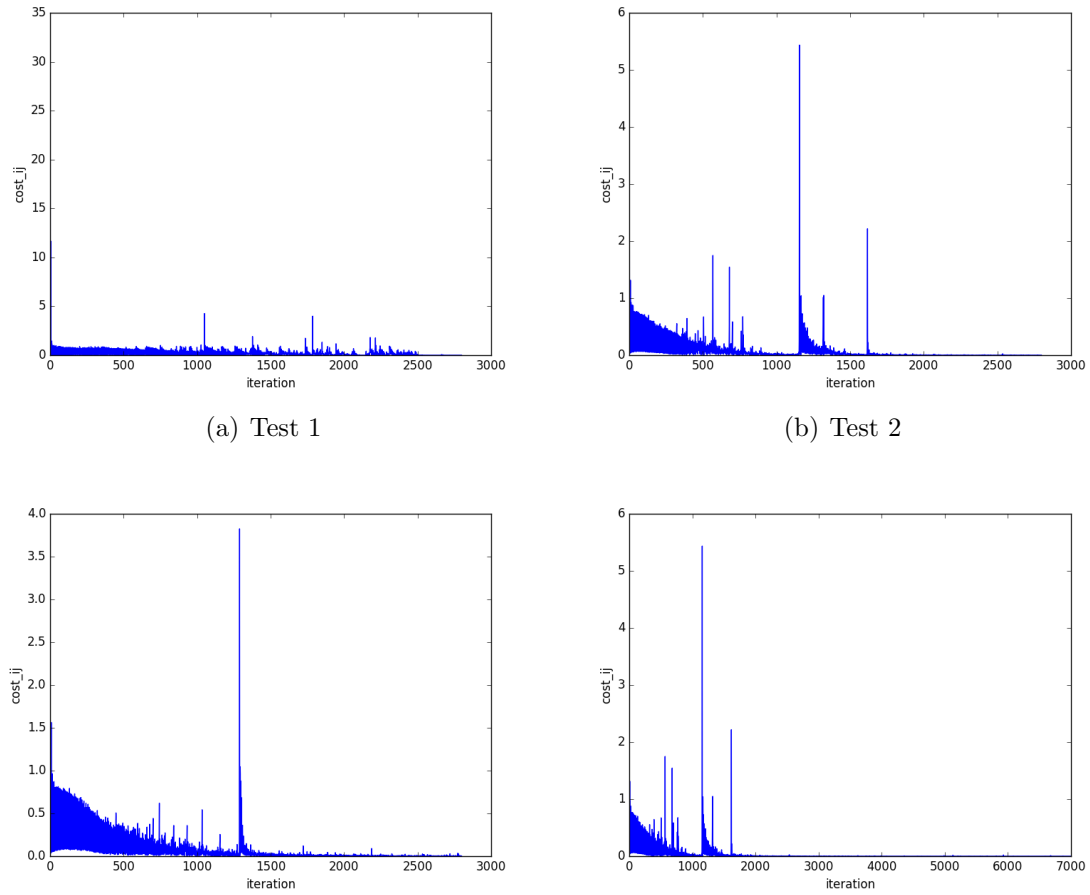
Figure 3.12: ROC and Precision- Recall courve - Noraml initialization. learning rate = 0.01 frav svm_general.

Classifier	Weight initialization	learning rate	TP	TN	FP	FN
Softmax	Normal	0.001	61	609	3	7
Softmax	Gaussian	0.001	66	605	7	2
SVM RBF(C=5)	Normal	0.01	63	593	19	5
SVM RBF(C=5)	Gaussian	0.01	63	593	19	5
SVM lineal(C=5)	Gaussian	0.01	63	593	19	5

- Test 1: Learning rate = 0.01 y 400 epoch.
- Test 2: Learning rate = 0.001 y 400 epoch.
- Test 3: Learning rate = 0.0005 y 400 epoch.
- Test 4: Learning rate = 0.001 y 1000 epoch.

In figure 3.13 the cost at training could be visualized. In general, should decrease varying its value but decreasing logarithmically. In the first experiment (test 1), the value varies but in a small range, and in general it is constant, in the other three experiments, the cost decreases and this is what must happen, but in test 2 the loss converges two times, after converge the first time, then it increase again ans converges again.

In figure 3.14 it is possible to see that the error changes but not in a logarithmically way. It increase and decreases its value in each epoch but it all experiments it gets in a 42% or 40% error. The desired curve should be as the loss one, but the error should not decrease as much as the training loss does.



(a) Test 1

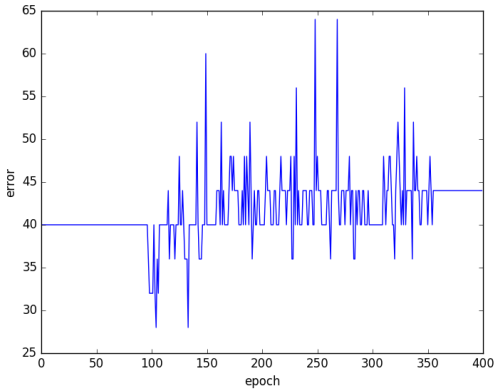
(b) Test 2

(c) Test 3

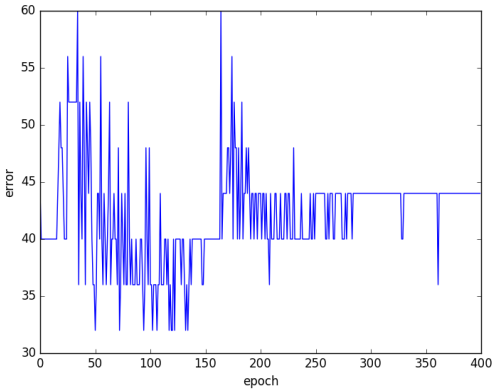
(d) Test 4

Figure 3.13: Training loss of four test Casia database

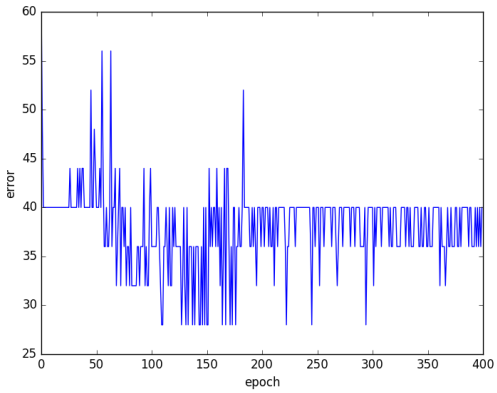
As the same way that in the first experiment that FRAV database converges is that would be expected from Casia, but it has not gotten.



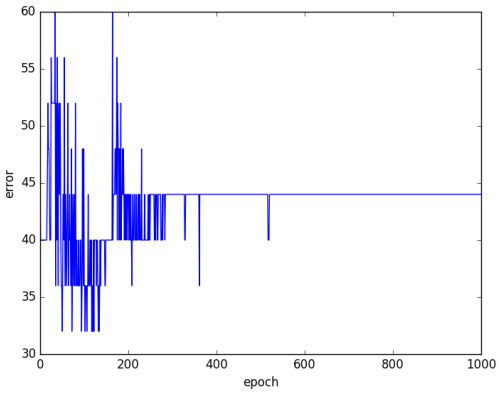
(a) Test 1



(b) Test 2



(c) Test 3



(d) Test 4

Figure 3.14: Validation error of four test Casia database

At the same way as FRAV, the classifier and the weight initialization has been changed in order to know how the networks behavior with these changes. The learning rate used for this experiment is 0.01. First, the cost (at training) and the error (at validating) are going to be visualized when the weight initialization is normal 3.16. Second when the weight initialization used is Gaussian 3.15.

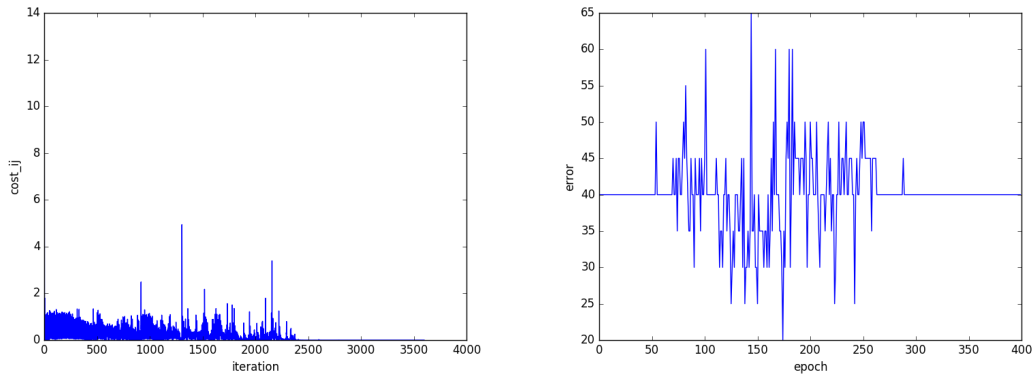


Figure 3.15: Cost at training and error at validating -casia svm_gauss.

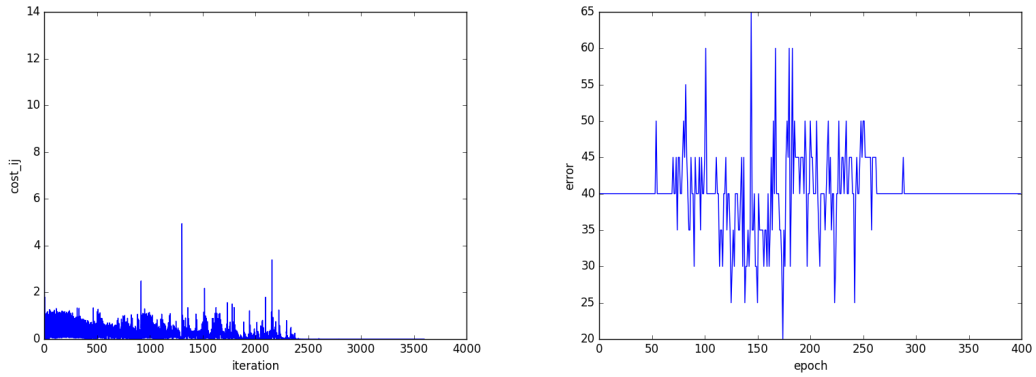


Figure 3.16: Cost at training and error at validating -casia svm_general.

As the same way than in FRAV database, the cost and at training the error at validating has not changed. The positive and negative rates are the following ones in the three experiments: TP = 8; TN = 21; FP = 3; FN = 8.

MFSD results

As the same way in FRAV and CASIA, but now with MFSD, it is going to be tested the network with a learning rate of 0.01, Gaussian initialization 3.17 and normal distribution initialization 3.18 and classifying the test with SVM (RBF kernel and linear) and softmax.

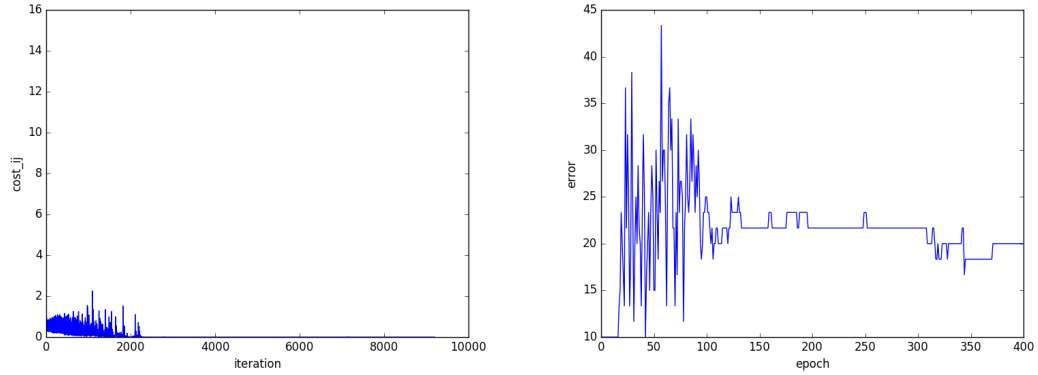


Figure 3.17: Cost at training and error at validating - mfsd svm_gauss.

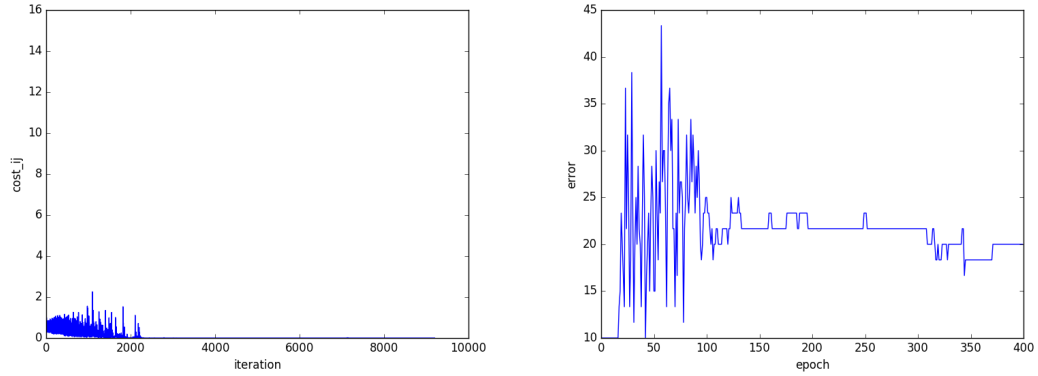


Figure 3.18: Cost at training and error at validating - mfsd svm_general.

The same problem occurs. The train and valid graphs are the same in both cases, independently of the initialization.
Also, the result at testing is the same in three cases: TP = 12, TN = 3, FP = 105, FN = 0

3.1.5 New database

From images, a new database has been developed. Images are read and re-sized directly to 128x128. The reason of using 128x128 is the size that authors in the Casia paper use. The databases explained in **??**:

-	FRAV	FRAV (rgb+nir)	CASIA images	CASIA video	MFSD
n train samples class 0	157	133	26	255	30
n train samples class 1	459	417	111	81	68
n test samples class 0	19	16	16	540	3
n test samples class 1	167	141	23	180	25
n valid samples class 0	10	8	7	105	2
n valid samples class 1	83	70	13	39	12

I can not obtain results with Casia RGB + NIR appended in classifier because it said that there is not space enough.

experiments

With that databases. Some experiments has been carried out:

- general experiment, with Gaussian weight initialization, classification with SVM RBF and SOFTMAX.
- general experiment but with a small database in order to make over-fitting in the network and check it. It has been used 20 train, test and validation images in all databases but MFSD that has been used 14 images for each subset.
- The same experiment that above but the test has been realized with the same subset that in training, this is to check that the network has over-fit or should have over-fitted.
- The same as above but decreasing the learning rate from 0.01 to 0.001.

From images, could be concluded that in general, decreasing the learning rate for this experiment has not been a good idea.

In table **??** could be seen the positive and negatives rates when has been used SVM RBF to classify. The result obtained with FRAV (rgb + NIR) is really good because just 3 samples have been missclaffied from 140 images.

I do not know why the test is the same for minidataset and minidatset tested with itself.

-	Optima CSVM	TP	TN	FP	FN
FRAV	0.05	136	24	11	9
FRAV (rgb+nir)	0.1	113	24	1	2
CASIA images	5	9	2	0	9
CASIA videos	0.1	478	75	105	62
MFSD	10	19	1	8	0

Looking the graphs where a minidataset has been used (20 images or 14), if the cost (training) is visualized, could be expected that the train is learning the images because the cost decreases to 0 (almost zero) so that means that if it is tested with itself the error should be 0, but that does not happen.

The conclusion is the needed of a balanced database, at least to do this experiment. In which the number of class 0 samples are the same that the number of class 1 samples, because the network would not learn in the same way if in some cases the number of samples of attack class is four times than the number of samples of class 1, just predicting

0 would have 25% accuracy, and having less than 5 samples in validation or testing is not a good generalizer (2 samples in class 0 MFSD database).

3.2 Final architecture

The architecture utilized in the final, and the most important experiment is described in this section.

The neural network is composed by five convolutional layers: the first and second convolutional layers (CL1 and CL2) , whose kernel sizes are 11x11 and 3x3 respectively, are followed by a local response normalization layer and a max pool layer whose size is 2x2. The third convolutional layer (CL3), with a kernel size of 3x3 , the next layer is a convolutional one (CL4) of size 3x3 followed by a max-pool layer whose size is 2x2. The next two layers are Dropouts layers (DL1 and DL2) with 4096 neurons. The next layer us a Fully-connected layer (FL) with 4096 neurons at the input and 2000 layers at the output.

The activation function used in each layer is the ReLu. The weights have been initialized pseudo-randomly (a random initialization that could be repeated selecting he same seed) with a Gaussian distribution and the bias has been initialized with 1.

It has been used the minibatch Stochastic Gradient Descend and in the training, the used classifier, is the logistic regression.

The learning rate is fixed at a 0.01 value and a bath size of 20, except when the MFSD database is used that he batch size is 14.

The dataset used in this experiment are the FRAV with the only RGB images, the FRAV database with the RGB and NIR added at the characteristic level and the classifier level. The MFSD database has been used too and both CASIA database has been used, image and video CASIA database.

For each database, it has been split randomly into train, test and validate subsets. Two classes has been used, class 0: the real users class and class 1: the attacks class.

For testing some classifiers has been utilized, and they are fed by the output of the convolutional neural network last layer, the Fully-connected layer.

The classifiers used to classify the features of the output of the CNN and get the results are the SVM (with RBF and linear kernel), KNN, Decision Tree and logistic regression. Also, PCA and LDA techniques has been used with each classifier separately to reduce

the dimensionality of the features.

The classifiers, before use them, has been personalize to each and particular time (for each database and if LDA or PCA is used). For that, cross validation has been used, more concretely, the *cros_val_Score()* function from sklarn has used with 10 folders.

- For SMV classifier, the C parameter has been searched among the following values: 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 2, 3, 5 and 10.
- For KNN classifier, the number of neighbours (K)has been searched among the following values: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28 and 30.
- For Deep Trees classifier, the depth of the tree has been searches among the following values: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28 and 30.
- For PCA, the number of components has been found in a range of 3 to 5000, in 3 to 3 steps.
- For LDA, the number of components has been found in a range of 1 to the length of the characteristic vector in 10 to 10 steps.

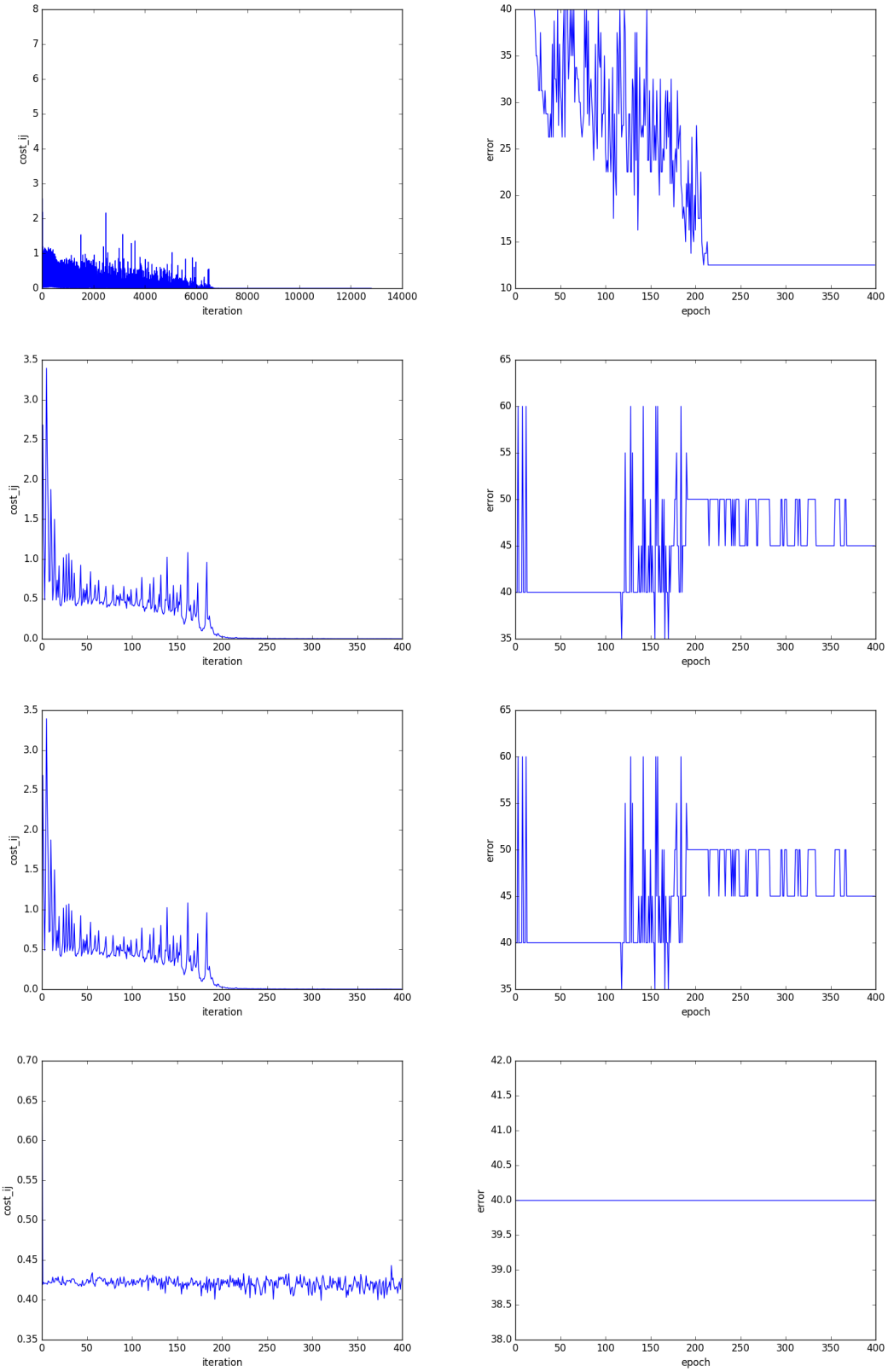


Figure 3.19: cost and error of the tree experiments with frav.

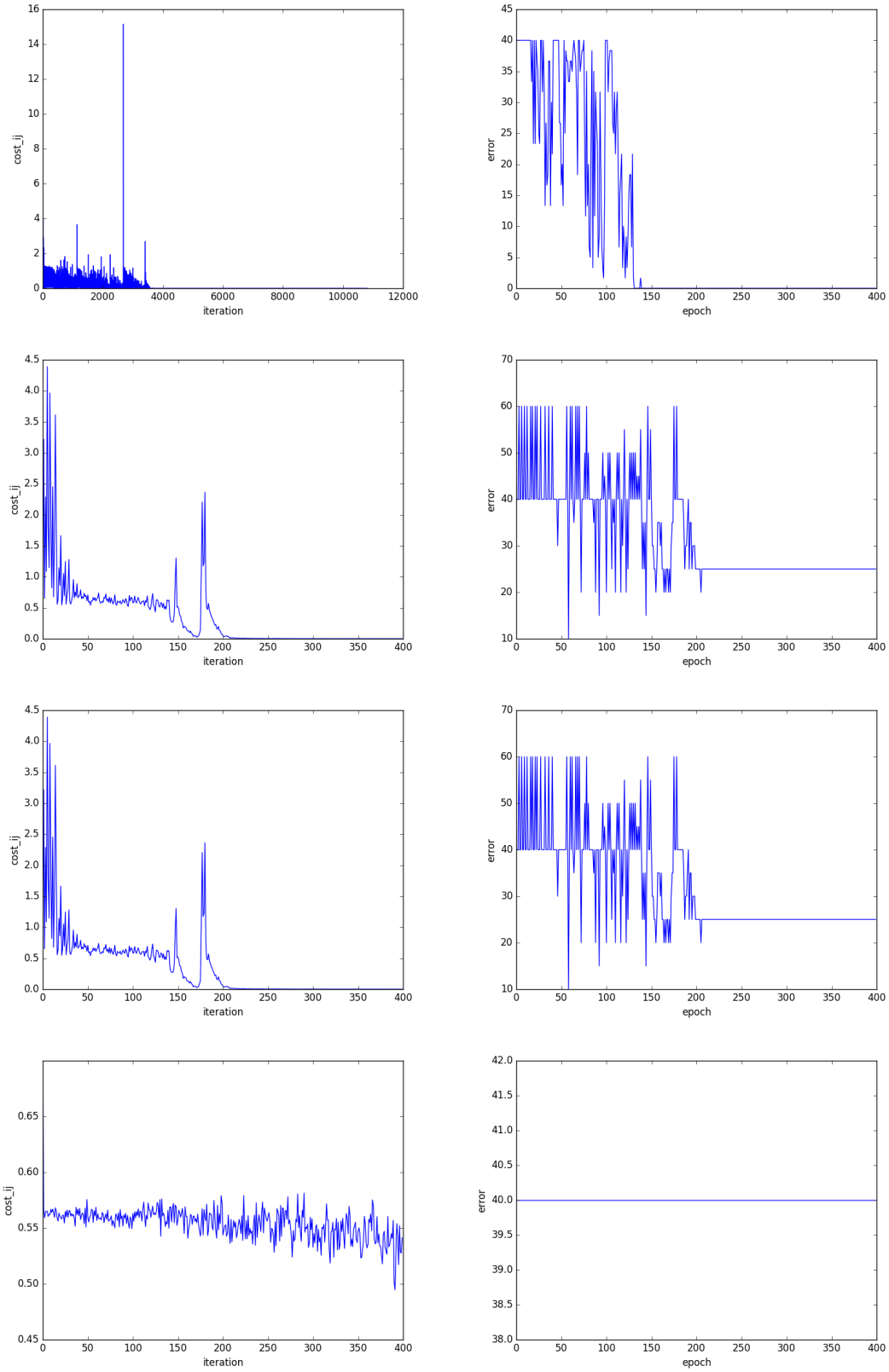


Figure 3.20: cost and error of the tree experiments with FRAV (rgb + nir) image level images.

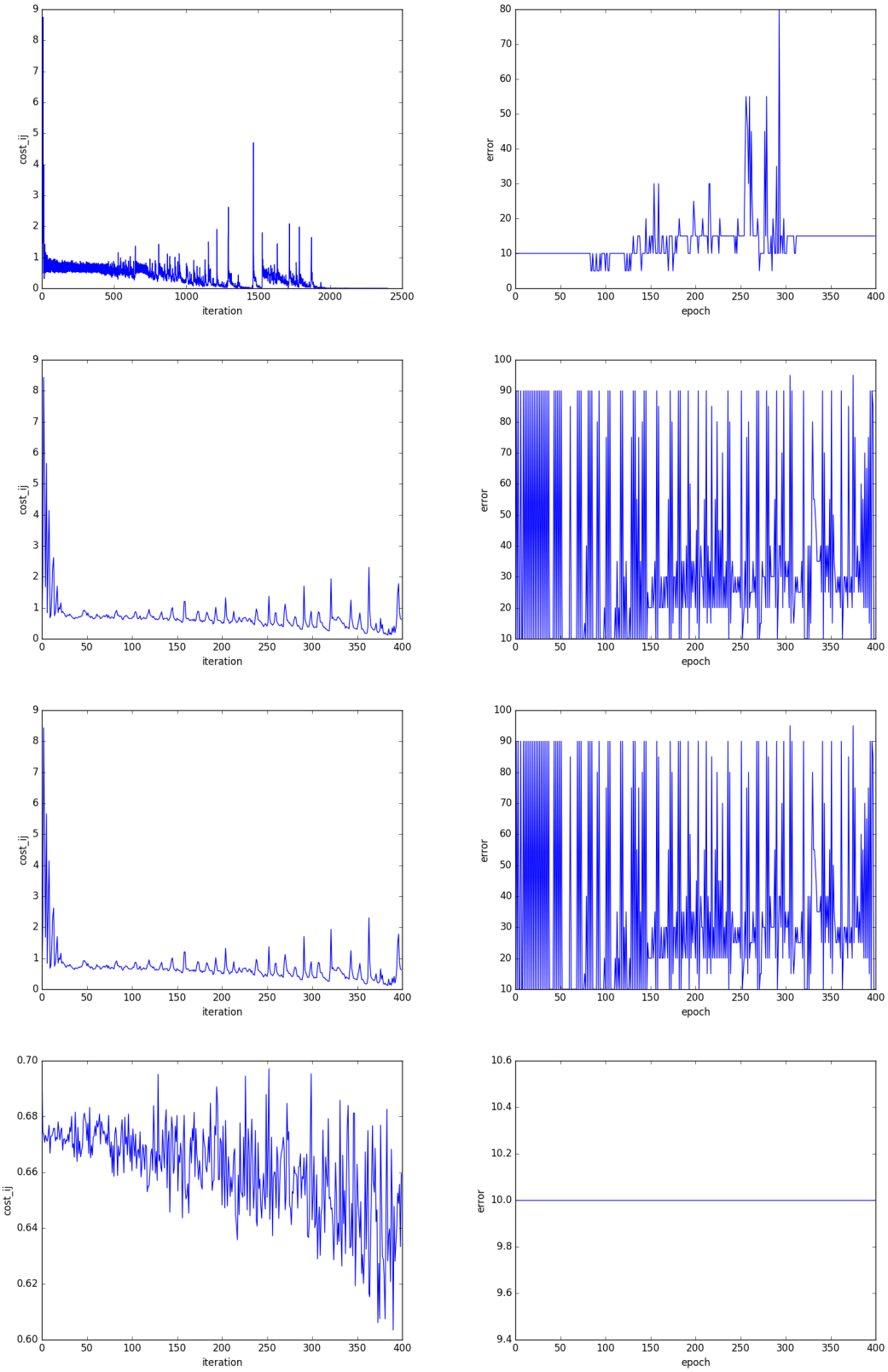


Figure 3.21: cost and error of the tree experiments with CASIA images.

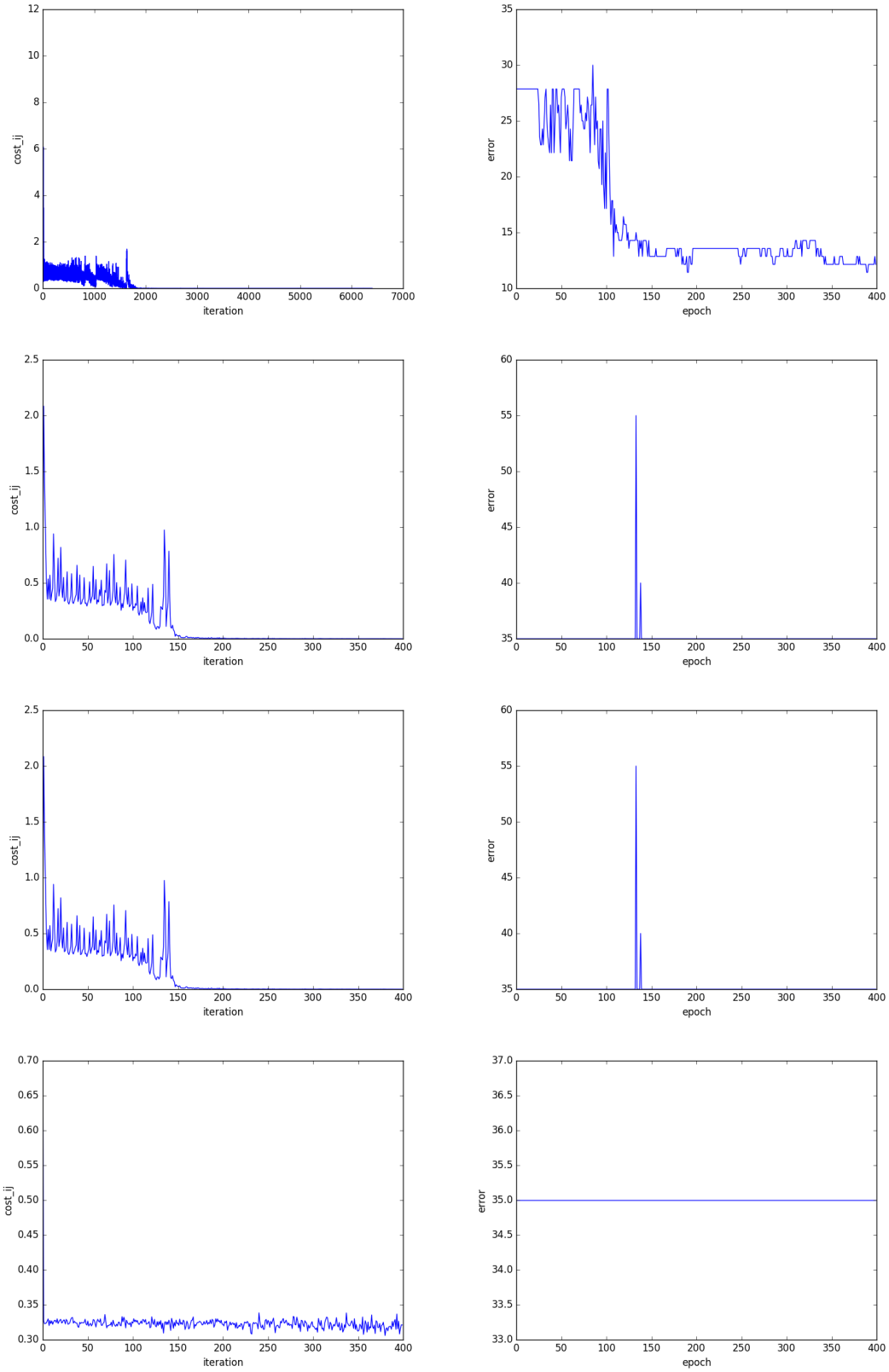


Figure 3.22: cost and error of the tree experiments with CASIA videos.

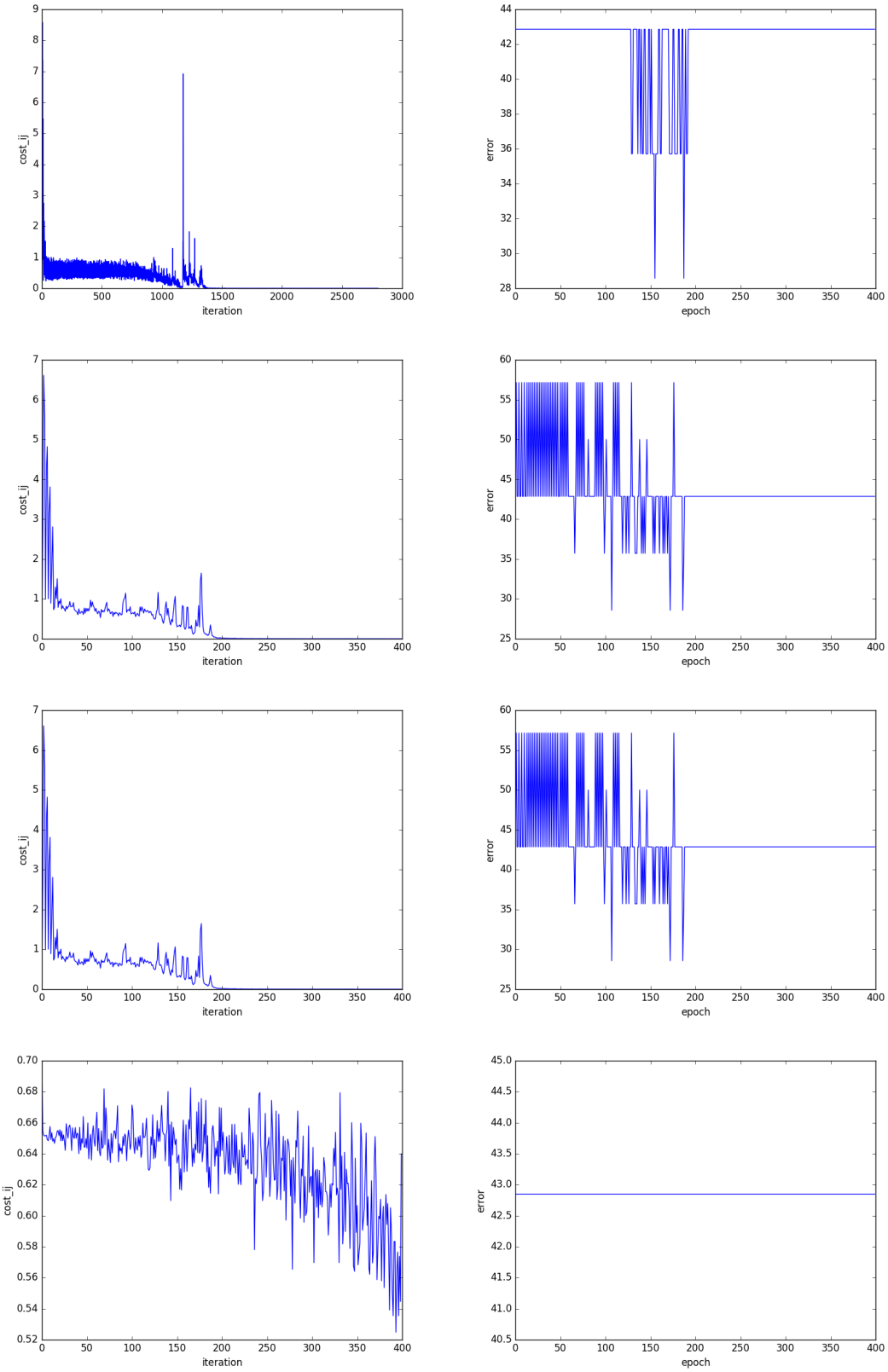


Figure 3.23: cost and error of the tree experiments with MFSD images.

Chapter 4

Results

In this chapter the results obtained are presented as well as the training process that is going to be discussed.

4.1 Training and validating process

The convolutional neural networks has been trained independently for each database as is explained in 3.2. In this section, the training and validation process is presented for each database.

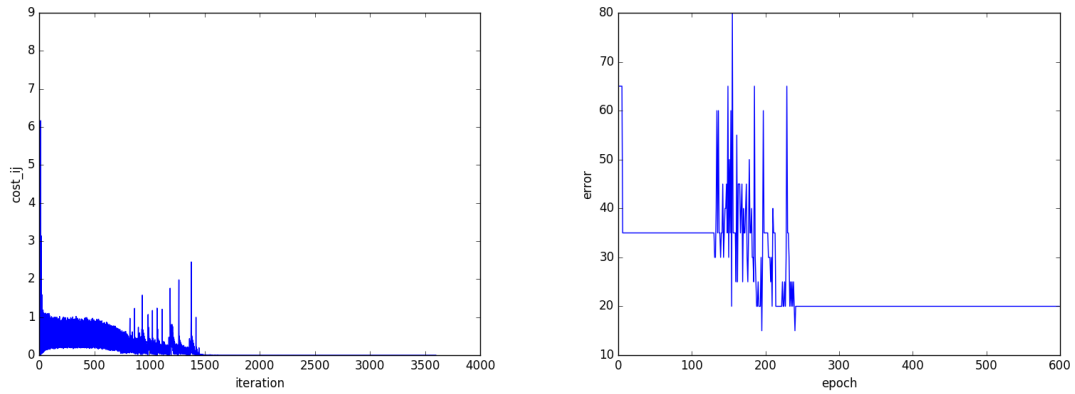
4.1.1 CASIA Image database

In figure 4.1 the cost and training are represented (4.1(a) and 4.1(a) respectively). The training process converge in a low values, smaller than 0.0001; while the validation process converges in 20% error from the 250th epoch. The iterations where the training process oscillates sharply agree with the epochs where the cost oscillates sharply too. After those oscillations, the validation gets constant and the training error variates in its lowest values.

The saved model to realize the test performance is the obtained at iteration 1176 with a 15% validation error.

4.1.2 CASIA Video database

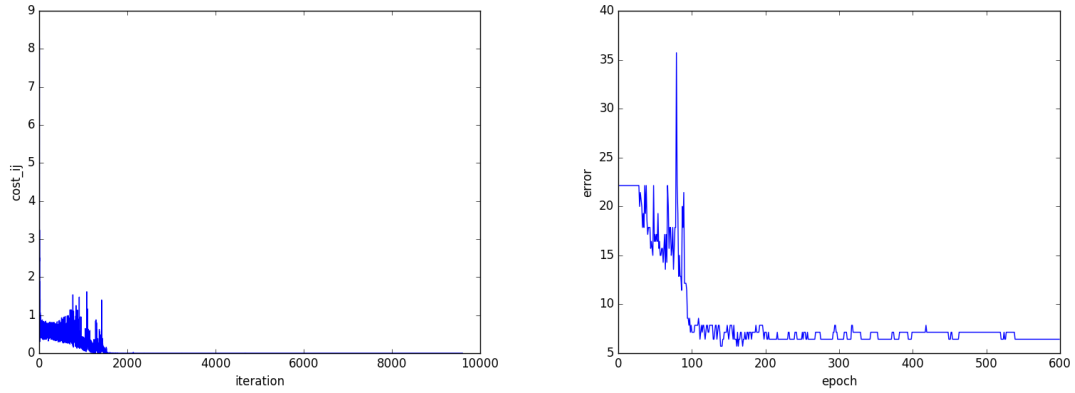
The cost and error obtained at training and validation processed when CASIA video database is used, are represented in figure 4.2. The minimun cost is almost 0 and is obtained before 2000th iteration, as is represented in figure 4.2(a). The validation converges



(a) Cost at training (b) Error at validation

Figure 4.1: Cost (a) and error (b) at training CASIA image database

at the same time as the training, where oscillate between 7% an 6%.



(a) Cost at training (b) Error at validation

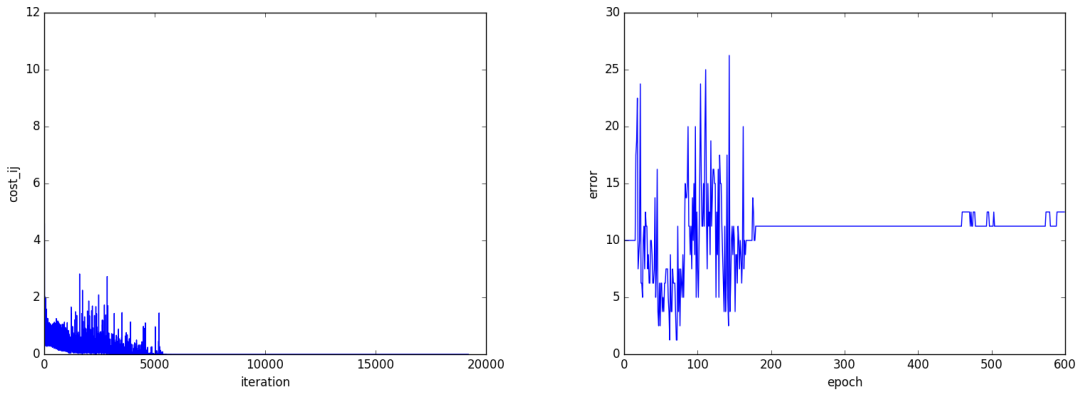
Figure 4.2: Cost (a) and error (b) at training CASIA video database

The best validation performance is obtained at iteration 2240 with a 5.71% validation error.

4.1.3 FRAV database

The cost and error obtained at training an validation processes respectably are represented in figure 4.3. The cost, that could be seen in figure 4.3(a) get a low value, when it

converges before the 5000th iteration, the cost obtained is 0.000001. The validation error, represented in figure 4.3(a), fluctuate a lot, the curve should decrease, but it does not and in 200 epoch, the error is constant in 11.25%. The generalization at validation is not very good and before the 100th epoch the network seems to overfit.



(a) Cost at training

(b) Error at validation

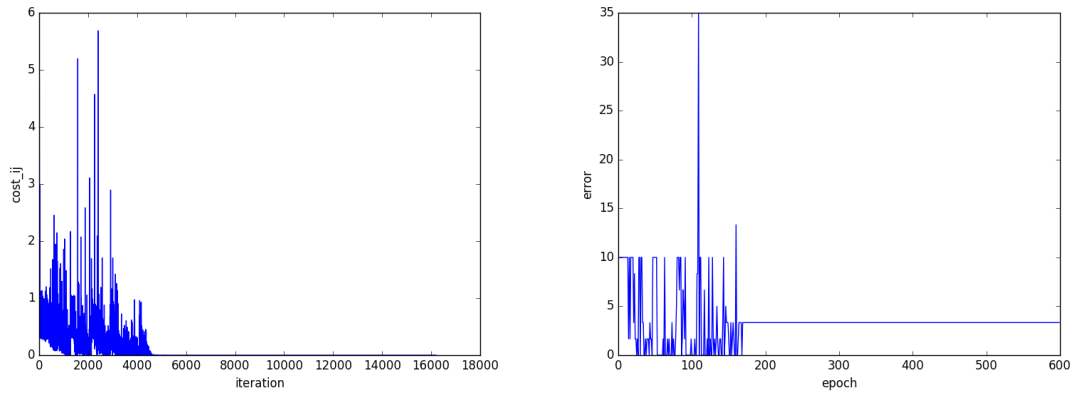
Figure 4.3: Cost (a) and error (b) at training RGB FRAV image database

The best validation score of 1.25% has been obtained at iteration 2016 which is saved as model for testing.

4.1.4 FRAV RGB+NIR (feature level) database

The RGB+NIR FRAV database, whose images has been added before the training process, has a cost and error which is represented in figure 4.4. The cost at training, that could be seen in 4.4(a) oscillates in the 3500 first iterations until decrease its value to 0 in the last 100 iterations. The cost at validation oscillates in the same oscillation cost period, where the value gets 0% in different times. After this, the value gets in 3,3%. Although the validation performance does not decrease in a desirable way, it gets a 0% error which means that the generalization is very good.

The best model is obtained at 702 iteration, when the validation gets 0% for the first time.



(a) Cost at training

(b) Error at validation

Figure 4.4: Cost (a) and error (b) at training RGB and NIR FRAV (feature level) image database

4.1.5 FRAV RGB+NIR (classification level) database

In figure 4.5 is represented the training cost and the validation error when the database has been trained, first the RGB subset and then NIR subset. This two subsets are trained independently and features of each best model are concatenated to test performance.

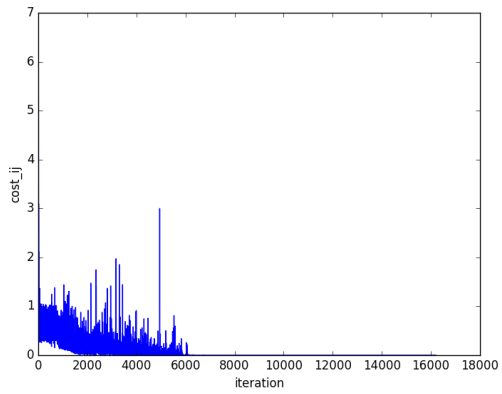
In RGB and NIR cost training, after decreasing its value got a very low value as in others databases. About the cost in validation, in both cases gets 0%, the difference is that in RGB, this lowest value is gotten twice at epoch 124 and 125 and then the value is increased until get constant at 10%, what seems an overfit. But in NIR validation error, it oscillates until gets 0% value and is constant until the end.

The best RGB model is the obtained at iteration 3347, when 0.0% is gotten at validation error. The best NIR model is the obtained at iteration 674, the first time that the validation error gets 0%.

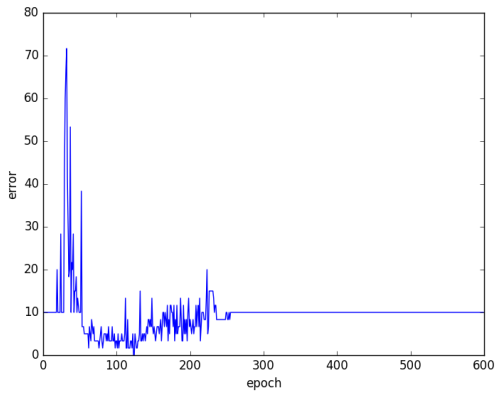
4.1.6 MFSD database

The training results obtained for MFSD database are represented in figure 4.6. The cost decreases slowly until get a desirable cost value, near to 0%. The validation error curve graph (figure 4.6(b)) is not as good as the cost curve. The lower error value is obtained in the first 100 iterations, which is constant and then is increased until 35%.

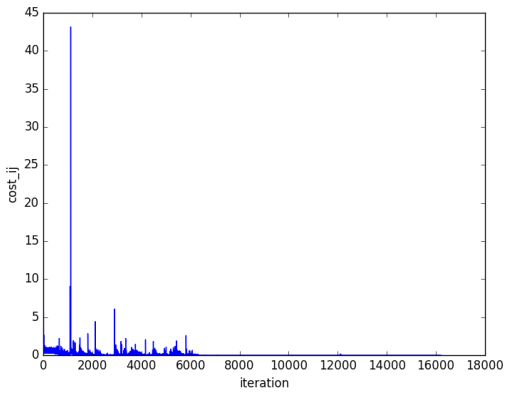
The best model is the obtained at iteration 7, in the first epoch, where the validation error is 14,28%.



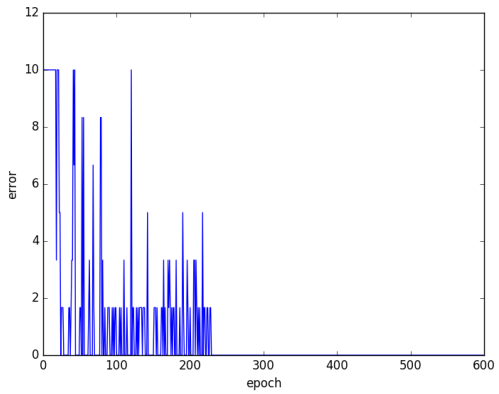
(a) Cost at training RGB subset



(b) Error at validation RGB subset



(c) Cost at training NIR subset



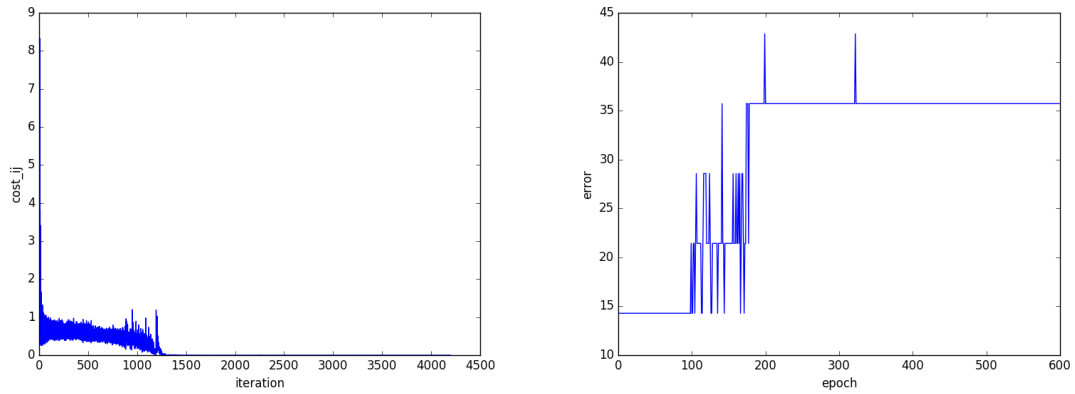
(d) Error at validation NIR subset

Figure 4.5: Cost (a) and error (b) at training RGB subset, cost (c) and error (d) at training NIR subset for FRAV (RGB+NIR at classification level) image database

4.2 Testing process

From the best CNN model, for each database, A SVM (with lineal and RBF kernel), KNN, Decision Tree and logistic regression has been trained. Also the testing has been realize after training a LDA and PCA (which have been trained too for each database) with each classifier.

The test performance for each classifier, has been realized after chosen the best classifier model. The results are going to be exposed independently of each database and then



(a) Cost at training

(b) Error at validation

Figure 4.6: Cost (a) and error (b) at training MFSD image database

compared among all databases.

Best obtained results are highlighted in bold in each table.

4.2.1 CASIA Image database

The results obtained testing the CNN model for CASIA image database are shown.

Classifier	APCR	BPCR	Classifier + PCA n components = 103	APCR	BPCR	Classifier + LDA n components = 1	APCR	BPCR
SVM - RBF C = 0.5	0	0.125	SVM - RBF C = 1	0	0.125	SVM - RBF C = 0.5	0	0.125
SVM - lineal C = 0.001	0	1	SVM - lineal C = 0.001	0	1	SVM - lineal C = 0.005	0	1
KNN k = 2	0	0.125	KNN k = 2	0	0.125	KNN k = 4	0	0.125
Decision Tree Depth = 12	0	0.5	Decision Tree Depth = 4	0	0.125	Decision Tree Depth = 2	0	0.125
Logistic Regression l. rate = 0.01	0	0.125	Logistic Regression l. rate = 0.0001	0	0.125	Logistic Regression l. rate = 0.0001	0	0.125

Table 4.1: APCR and BPCR classifying results using CASIA image database

In table 4.1 are exposed the APCR and BPCR parameters for each classifier. All the attacks samples are well classified because APCR value is 0. The number of genuine samples misclassified is what changes, it gets 1 for SVM with kernel lineal,all the positives samples have been misclassified, and decision tree whose number of misclassified real user are 4 (the half of the total real samples of the test subset). The rest of the classifiers

takes a 0.125 APCR value, just one sample is misclassified, the same sample which could be seen in figure 4.7.

With respect to APCR and BPCR it is not possible to get the best combination with which the results are better.

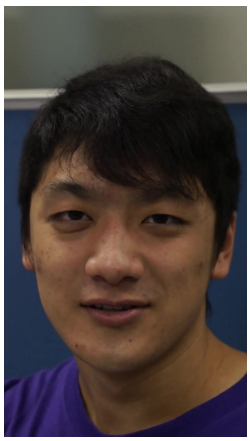


Figure 4.7: Casia image misclassified sample.

4.2.2 CASIA Video database

Next analyzed results are from CASIA video database.

Classifier	APCR	BPCR	Classifier + PCA n components = 283	APCR	BPCR	Classifier + LDA n components = 1	APCR	BPCR
SVM - RBF C = 0.1	0.14	0.53	SVM - RBF C = 0.5	0.17	0.46	SVM - RBF C = 0.05	**0.16**	**0.44**
SVM - lineal C = 0.001	0	1	SVM - lineal C = 0.001	0	1	SVM - lineal C = 0.001	0	1
KNN k = 2	0.19	0.42	KNN k = 2	0.2	0.42	KNN k = 2	0.16	0.47
Decision Tree Depth = 2	0.19	0.49	Decision Tree Depth = 2	0.15	0.53	Decision Tree Depth = 2	0.15	0.46
Logistic Regression l. rate = 0.01	0.15	0.49	Logistic Regression l. rate = 0.005	0.23	0.34	Logistic Regression l. rate = 0.005	0.23	0.37

Table 4.2: APCR and BPCR classifying results using CASIA video database

In table 4.2 APCR and BPCR results are summarized. The best result has been obtained with SVM classifier and kernel RBF (C= 0.05) after applying LDA with 1 component. The worst values obtained are the SVM with linear kernel, no matters if PCA

or LDA have been used, because all the positives samples have been classified as negatives.

From the table it is not possible to know if LDA and PCA are significant because with some classifiers APCR and BPCR results have been improve or worsened.

Also The last conclusion that could be gotten from the table is that Attacks samples are better classified that genuine samples, it is due to the fact that has been trained with more negative samples.

4.2.3 FRAV database

The RGB FRAV database results are presented and analyzed.

Classifier	APCR	BPCR	Classifier + PCA n components = 463	APCR	BPCR	Classifier + LDA n components = 1	APCR	BPCR
SVM - RBF C = 0.5	0.06	0.06	SVM - RBF C = 1	**0.04**	**0.06**	SVM - RBF C = 0.1	0.05	0.06
SVM - lineal C = 0.005	0	1	SVM - lineal C = 0.005	0	1	SVM - lineal C = 0.5	0.05	0.06
KNN k = 16	0.07	0.06	KNN k = 26	0.06	0.06	KNN k = 20	0.05	0.06
Decision Tree Depth = 2	0.04	0.11	Decision Tree Depth = 2	0.06	0.11	Decision Tree Depth = 2	0.06	0.06
Logistic Regression l. rate = 0.01	0.01	0.17	Logistic Regression l. rate = 0.05	0.06	0.06	Logistic Regression l. rate = 0.005	0.06	0.06

Table 4.3: APCR and BPCR classifying results using FRAV RGB database

APCR and BPCR results are in table 4.3. In general, results are very good because values are lower than 0.5. With SVM linear kernel, all the samples are classified as negatives except when it has been used when LDA, that results are as good as the obtained with RBF kernel.

The best result is obtained when SVM with RBF kernel has been used with C = 1 and using LDA with 1 component. LDA improves the results obtained when neither PCA nor just the classifier has been used, although the improved is not too much.

The misclassified samples are repeated among the classifiers. The two samples shown in figure 4.8 represents the two most misclassified images, almost every classifier have classified it incorrectly.

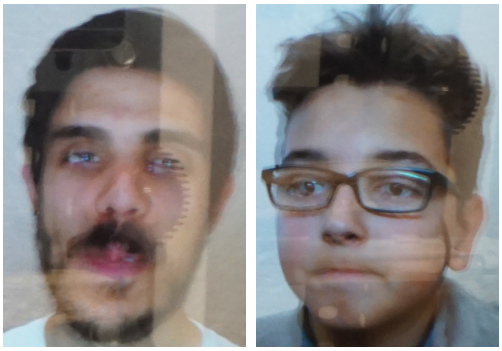


Figure 4.8: RGB FRAV misclassified samples.

4.2.4 FRAV RGB+NIR (feature level) database

Results obtained at classifying RGB+NIR (feature level) database are shown following.

Classifier	APCR	BPCR	Classifier + PCA n components = 463	APCR	BPCR	Classifier + LDA n components = 1	APCR	BPCR
SVM - RBF C = 2	0.02	0	SVM - RBF C = 5	0.02	0	SVM - RBF C = 0.1	0.04	0
SVM - lineal C = 0.005	0	1	SVM - lineal C = 0.001	0	1	SVM - lineal C = 10	0.05	0
KNN k = 6	0.09	0	KNN k = 12	0.12	0	KNN k = 10	0.04	0
Decision Tree Depth = 2	**0**	**0**	Decision Tree Depth = 2	0.01	0	Decision Tree Depth = 2	0.05	0
Logistic Regression l. rate = 0.01	**0**	**0**	Logistic Regression l. rate = 0.05	0.06	0	Logistic Regression l. rate = 0.05	0.07	0

Table 4.4: APCR and BPCR classifying results using FRAV RGB+NIR (feature level) database

Table 4.4 shows the APCR and BPCR values that have been obtained for each classifier. Except for SVM with linear kernel, which has classified all the samples as negatives when just the classifier has been used or if PCA has been aplied too, classifications have produced good results, the APCR and BPCR values are very low, close to 0. In fact, Decision Tree and logistic regression classify correctly all the samples because APCR and BPCR are equal to 0.

PCA and LDA have not improved results significantly. SVM lineal with LDA does not classify all the samples as negatives. But when PCA and LDA is applied with logistic regression and Decision tree, the classification is not perfect as it is when none is applied.

In general, the same samples are miscclasified in the classification tasks. The two most misclassified samples are represented in figure 4.9.

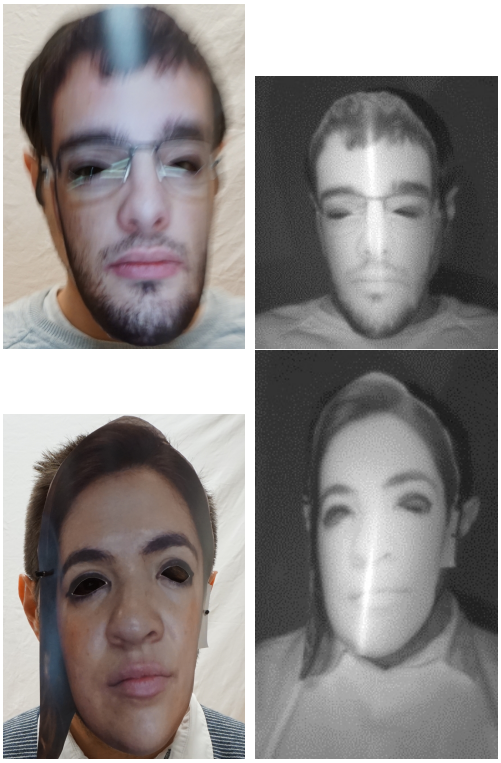


Figure 4.9: RGB+NIR FRAV (feature level) misclassified samples.

4.2.5 FRAV RGB+NIR (classification level) database

Results obtained when RGB+NIR FRAV database (classification level) is used are described.

APCR and BPDR results are summarized in table 4.5. Results are closer to 0, but none classifier classifies perfectly. There is possible to see that with LDA that APCR results are better and BPCR results are perfect.

SVM linear classifier classifies incorrectly positives samples although PCA is used too, with LDA that does not happen.

In most of cases, classifiers just misclassify one sample, the same sample represented in figure 4.10. Classifiers that misclassify more than one sample, the sample 4.10 is one of the misclassified.

Classifier	APCR	BPCR	Classifier + PCA n components = 463	APCR	BPCR	Classifier + LDA n components = 1	APCR	BPCR
SVM - RBF C = 10	0.01	0	SVM - RBF C = 1	0.02	0	SVM - RBF C = 2	0.01	0
SVM - lineal C = 0.001	0	1	SVM - lineal C = 0.001	0	1	SVM - lineal C = 1	0.01	0
KNN k = 6	0.32	0	KNN k = 14	0.04	0	KNN k = 6	0.01	0
Decision Tree Depth = 2	0.04	0.07	Decision Tree Depth = 2	0.05	0.07	Decision Tree Depth = 2	0.01	0
Logistic Regression l. rate = 0.01	0	0.29	Logistic Regression l. rate = 0.05	0.07	0	Logistic Regression l. rate = 0.005	0.02	0

Table 4.5: APCR and BPCR classifying results using FRAV RGB+NIR (classification level) database

Figure 4.10: RGB+NIR FRAV (classification level) misclassified samples.

4.2.6 MFSD-MSU database

MFSD-MSU database results are described following.

The classification with this database is not good because all the samples are classified as negatives, as could be seen in table 4.6 where , independently of the classifier or if LDA and PCA are used, the BPCR value is always 1 and the APCR value is 0.

4.2.7 Comparative among databases

Classifier	APCR	BPCR	Classifier + PCA n components = 463	APCR	BPCR	Classifier + LDA n components = 1	APCR	BPCR
SVM - RBF C = 0.01	0	1	SVM - RBF C = 0.001	0	1	SVM - RBF C = 1	0	1
SVM - lineal C = 0.001	0	1	SVM - lineal C = 0.001	0	1	SVM - lineal C = 10	0	1
KNN k = 24	0	1	KNN k = 30	0	1	KNN k = 30	0	1
Decision Tree Depth = 12	0	1	Decision Tree Depth = 2	0	1	Decision Tree Depth = 2	0	1
Logistic Regression l. rate = 0.01	0	1	Logistic Regression l. rate = 0.0001	0	1	Logistic Regression l. rate = 0.0001	0	1

Table 4.6: APCR and BPCR classifying results using MFSD-MSU database

Chapter 5

Conclusions and future work

5.1 Conclusions

In this section the conclusions obtained along the mothodology and results are presented and discused.

- From [31]: when a small (or non-representative) trianing data set is used, there is no guarantee that LDA will outperform PCA (esto deberia de compararse con los resultados que se obtinen y parafrasear la frase si es necesario).
- Best results with FRAV because of the quality of images
- A balanced database and which more samples would reproduce better results.

5.2 FUture work

Ademas de lo que tengo en el conocllusion de mi ordenador:

It would be interesting test the neural network and the classifier which work the best in a database with other databases.

Bibliography

[1] J. Wayman, A. Jain, D. Maltoni, and D. Maio, *An Introduction to Biometric Authentication Systems*, pp. 1–20. Springer London, 2005.

[2] "Take biometric security systems to the next level." http://www.rechters.pl/take-biometric-security-systems-to-the-next-level/. Accessed: 2017-05-04.

[3] J. Galbally, S. Marcel, and J. Firrez, "Biometric antispoofing methods: A survey in face recognition.," *IEEE Access*, vol. 2, pp. 1530–1552, 2014.

[4] D. Wen, H. Han, and A. K. Jain, "Face spoof detection with image distortion analysis," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 4, pp. 746–761, 2015.

[5] K. D. I and M. Grgic, "A survey of biometric recognition methods," in *Proceedings. Elmar-2004. 46th International Symposium on Electronics in Marine*, pp. 184–193, 2004.

[6] Z. Xu, S. Li, and W. Deng, "Learning temporal features using LSTM-CNN architecture for face anti-spoofing," in *3rd IAPR Asian Conference on Pattern Recognition, ACPR 2015, Kuala Lumpur, Malaysia, November 3-6, 2015*, pp. 141–145, 2015.

[7] M. K. Hani and S. S. Liew, "A convolutional neural network approach for face verification," in *International Conference on High Performance Computing & Simulation, HPCS 2014, Bologna, Italy, 21-25 July, 2014*, pp. 707–714, 2014.

[8] J. Yang, Z. Lei, and S. Z. Li, "Learn convolutional neural network for face anti-spoofing," *CoRR*, vol. abs/1408.5601, 2014.

[9] I. Chingovska, A. Anjos, and S. Marcel, "On the effectiveness of local binary patterns in face anti-spoofing," Idiap-RR Idiap-RR-19-2012, Idiap, 2012.

[10] L. Sun, G. Pan, Z. Wu, and S. Lao, *Blinking-Based Live Face Detection Using Conditional Random Fields*, pp. 252–260. Springer Berlin Heidelberg, 2007.

[11] D. Kriesel, *A Brief Introduction to Neural Networks*. 2007.

[12] R. Rojas, *Neural Networks: A Systematic Introduction*. Springer-Verlag New York, Inc, 1996.

[13] M. F. Bear, B. W.Connors, and M. A. Paradiso. Williams & Wilkins, 1996.

[14] B. J. A. Kröse and P. P. van der Smagt, *An Introduction to Neural Networks*. The University of Amsterdam, 8th ed., 1996.

[15] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural Network Design*. USA: Martin Hagan, 2nd ed., 2014.

[16] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000.

[17] *Analysis of Deep Convolutional Neural Network Architectures*, vol. 21, University of Twente, 2014.

[18] D. Eigen, J. T. Rolfe, R. Fergus, and Y. LeCun, "Understanding deep architectures using a recursive convolutional network," *CoRR*, vol. abs/1312.1847, 2013.

[19] C. Farabet, R. Paz, J. Perez-Carrasco, C. Zamarreos, A. Linares-Barranco, Y. LeCun, E. Culurciello, T. Serrano-Gotarredona, and B. Linares-Barranco, "Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convnets for visual processing," *Frontiers in Neuroscience*, vol. 6, p. 32, 2012.

[20] T. D. Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, 2016.

[21] "Sample digits of mnist handwritten digit database." https://www.researchgate.net/figure/264273647_fig1_Fig-18-0-9-Sample-digits-of-MNIST-handwritten-digit-database. Accessed: 2017-04-19.

[22] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," Tech. Rep. 07-49, University of Massachusetts, Amherst, 2007.

[23] "Automated border control gates for europe." http://abc4eu.com/. Accessed: 2017-04-19.

[24] Z. Zhang, J. Yan, S. Liu, Z. Lei, D. Yi, and S. Z. Li, "A face antispoofing database with diverse attacks," in *5th IAPR International Conference on Biometrics, ICB 2012, New Delhi, India, March 29 - April 1, 2012*, pp. 26–31, 2012.

[25] D. Wen, H. Han, and A. K. Jain, "Face spoof detection with image distortion analysis," *IEEE Trans. Information Forensics and Security*, vol. 10, no. 4, pp. 746–761, 2015.

[26] D. Stutz, "Understanding convolutional neural networks," tech. rep., Fakultt fr Mathematik, Informatik und Naturwissenschaften, 2014.

[27] M. Sokolova, N. Japkowicz, and S. Szpakowicz, "Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation," in *Proceedings of the 19th Australian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, pp. 1015–1021, 2006.

[28] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pp. 233–240, 2006.

[29] "The area under an roc curve." http://gim.unmc.edu/dxtests/roc3.htm. Accessed: 2017-04-19.

[30] "Sc37 iso/iec jtc1," 2014.

[31] A. M. Martinez and A. C. Kak, "Pca versus lda," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. 23, pp. 228–233, 2001.

[32] S. Dreiseitl and L. Ohno-Machado, "Logistic regression and artificial neural network classification models: a methodology review," *Journal of Biomedical Informatics*, vol. 35, no. 56, pp. 352 – 359, 2002.

[33] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[34] "Opencv - undestanding svm." http://docs.opencv.org/3.1.0/d4/db1/tutorial_py_svm_basics.html. Accessed: 2017-04-24.

[35] D. Ciobanu, "Using svm for classification," *Acta Universitatis Danubius. OEconomica*, no. 5(5), pp. 209–224, 2012.

[36] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification," tech. rep., Department of Computer Science, National Taiwan University, 2003.

[37] "Scikit-learn - decision tree." http://scikit-learn.org/stable/modules/tree.html. Accessed: 2017-04-28.

[38] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[39] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *CoRR*, vol. abs/1206.5533, 2012.

[40] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010.

List of Figures

List of Tables