‘Learning convolutional Neural Network for Face Anti-Spoofing”

Beatriz Gómez Ayllón

April 28, 2017

Abstract

Add abstract

Contents

Part I

MEMORY

Chapter 1

Introduction

1.1 Introduction

This document blaaa blaaa

1.1.1 Anti-spoofing

bla bla bla

1.2 Convolutional Neural Network background theory

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

1.2.1 Introduction

Convolutional Neural Network (CNN) are a specific type of Neural Networks (NN).

1.3 Programming language and frameworks

1.3.1 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.

1.3.2 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 [1] 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.

1.4 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 ...).

1.4.1 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 1.1, image obtained

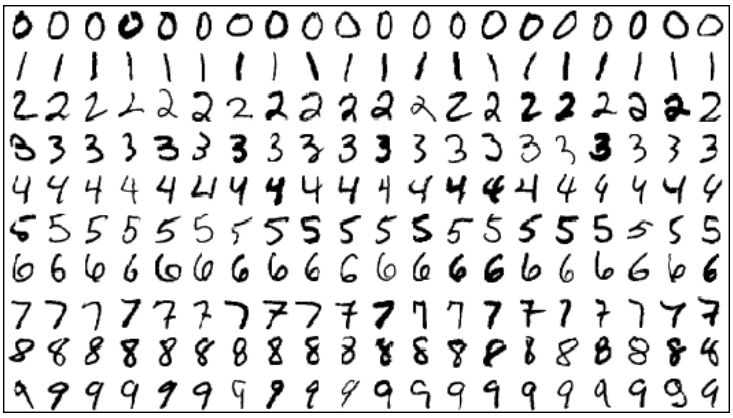


Figure 1.1: MNIST digit images database. Image obtained from [2]

from [2] 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.

1.4.2 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/ [3].

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.

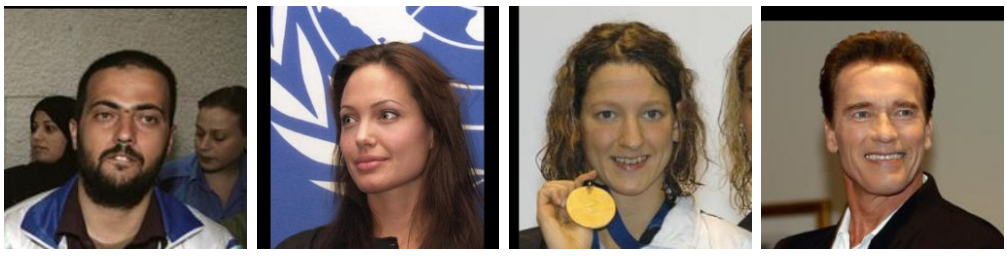


Figure 1.2: Samples of LFW database

Four examples of the images of this database could be seen in figure 1.2 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.

1.4.3 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 [4].

One exaple of RGB images of FRAV database are shown in figure 1.3 and another example could be seen in 1.4. 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 1.3: Four attacks and real user from RGB FRAV database

As could be seen in figure 1.3 and figure 1.4, 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 1.3(d) and figure 1.4(e) for RBG images and figure 1.4(j) for NIR image.
- Images of people printed (attack) represented in figure 1.3(a) and figure 1.4(a) for RBG images and figure **??** for NIR image.
- Images of people with a mask (attack)represented in figure 1.3(b)and figure 1.4(b)for RBG images and figure 1.4(g) for NIR image.
- Images of people with a mask with the eyes cropped (attack)represented in figure 1.3(c) and figure 1.4(c) for RBG images and figure 1.4(h) for NIR image.

(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 1.4: Four attacks and real user of RGB and NIR FRAV database

– Images of people in a tablet (attack)represented in figure 1.3(d) and figure 1.4(d) for RBG images and figure 1.4(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.

– 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 1.3), 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 1.4)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.

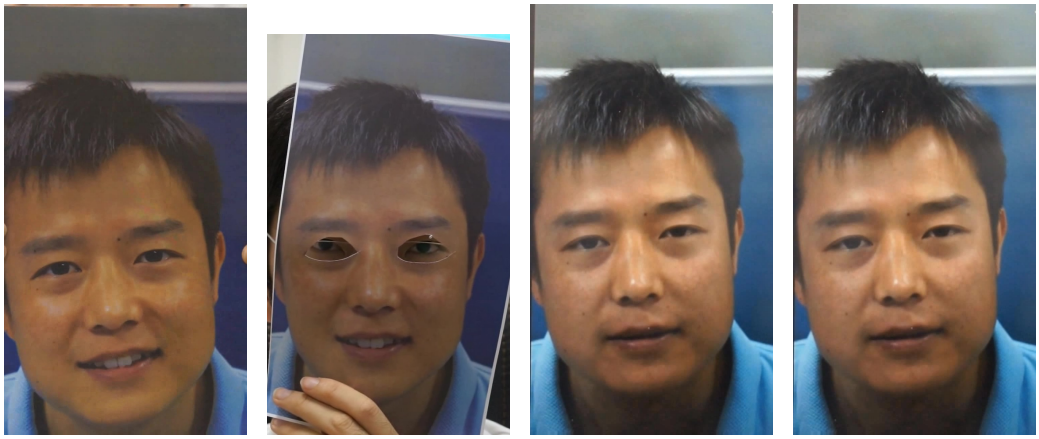
1.4.4 CASIA dataset

The CASIA Face Anti-Spoofing database is a database from the Chinese Academy of Sciences Centre for Biometrics and Security Research (CASIA-CBSR) [5].

A person of CASIA database could be seen in figure 1.5, where three attacks types could be seen (figure 1.5(a), 1.5(b) and 1.5(c)) with the real user (figure 1.5(d)).

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 1.5(a).



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

Figure 1.5: Three attacks and real user from casia database

- Images of people with a mask with the eyes cropped (attack) represented in figure 1.5(b).
- Tablet attack represented in figure 1.5(c)
- Images of real users represented in figure 1.5(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.

1.4.5 MSU - MFSD database

The MSU Mobile Face Spoofing Database (MFSD) is a video face anti-spoofing database [6].

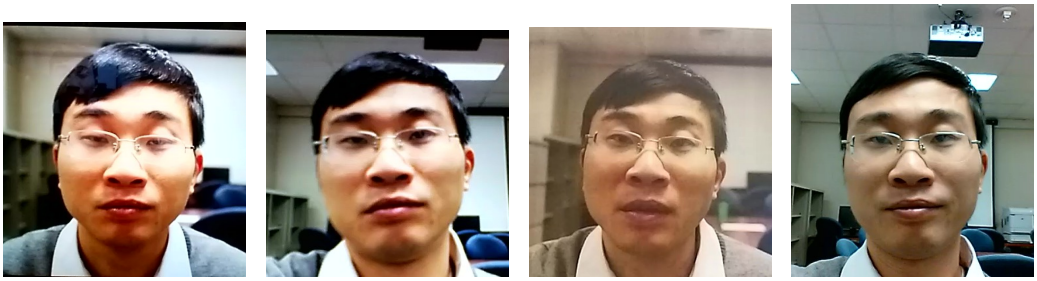
In figure 1.6 are represented the three attacks and a real user which forms this database:

– Printed photo attack represented in figure 1.6(a).

– Tablet attack where a video is Replayed represented in figure 1.6(a).

– Smartphone attack represented in figure 1.6(a).

– Real user represented in figure 1.6(a).



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

Figure 1.6: 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.

1.5 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.

1.5.1 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 [7].

The loss is calculated in the following way:

$$Loss(\theta, D) = -\sum_{i=0}^{|D|} logP(Y = y^{(i)}|x^{(i)}, \theta) \tag{1.1}$$

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.

1.5.2 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 [8]. 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 1.1 [8, 9]:

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

Table 1.1: Confusion Matrix

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

1.5.3 ROC curve and Precision and Recall curve

From the confusion matrix, it is possible calculate others parameters [8]: 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 [9]:

$$FPR = \frac{FP}{N} \tag{1.2}$$

Where:

$$N = FP + TN \tag{1.3}$$

– The True Positive Rate (TPR) is defined as the proportion of all the positive samples (P) that are classified correctly [9]. This parameter could be known as Recall too:

$$TPR = Recall = \frac{TP}{P} \tag{1.4}$$

Where:

$$P = TP + FN \tag{1.5}$$

– Precision is defined as the proportion of real positive samples that has been classified as positive [8, 9]:

$$precision = \frac{TP}{TP + FP} \tag{1.6}$$

– Accuracy is defined as the proportion of the well-classified samples of all the samples [8]. The classifiers implemented in scikit-learn library return this value as metric.:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1.7}$$

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) [9].

The figure 1.7, obtained from [10], 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.

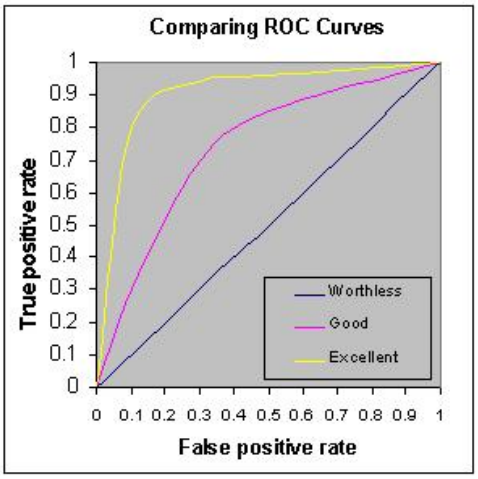


Figure 1.7: ROC curves. Image obtained from [10]

1.5.4 APCER and BPCER

The ISO/IEC 30107-3 [11] 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.)

$$APCER_{PAIS} = \frac{1}{N_{PAIS}} \sum_{i=1}^{N_{PAIS}} (1 - Res_i) \tag{1.8}$$

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

$$BPCER = \frac{\sum_{i=1}^{N_{BF}} Res_i}{N_{BF}} \tag{1.9}$$

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.

1.6 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 [12], 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 [13], 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.

1.6.1 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 [14]:

$$\prod_{i=1}^{n} P(y_i|X_i, W, b) \tag{1.10}$$

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:

$$P(Y = i|x, W, b) = \frac{e^{W_i x + b_i}}{\sum_j e^{W_i x + b_i}} \tag{1.11}$$

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

$$y_{pred} = argmax_i P(y_i|X_i, W, b) \tag{1.12}$$

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 [15]. An hyperplane is optimal if the margin is the maximum and this margin is calculated (as the same way as logistic regression):

$$\arg\max_{wb} \left\{ \frac{1}{||W||} \min_n [t_n(W^T \phi(X_n) + b)] \right\} \tag{1.13}$$

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.

In figure 1.8 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 [16].

Based on estimate the hyperplane that the distance between classes, the closest vectors of each class, is maximized [15, 17]. 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 [14, 18]. There are many kernels (polynomial, sigmoid...), but the two used are described:

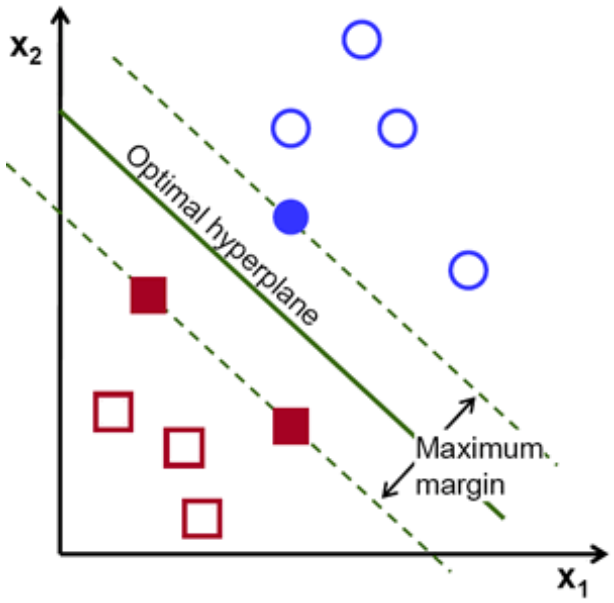


Figure 1.8: Optimal hyperplane and the decision boundary. Image obtained from [16]

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 [14]. The density function is determined by the form [15]:

$$p(x) = \frac{K}{NV} \tag{1.14}$$

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 [15]:

$$p(x|C_k) = \frac{K_k}{N_k V} \tag{1.15}$$

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 [12]. 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 [14] as is described in figure 1.9 (image obtained from [19]). 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.

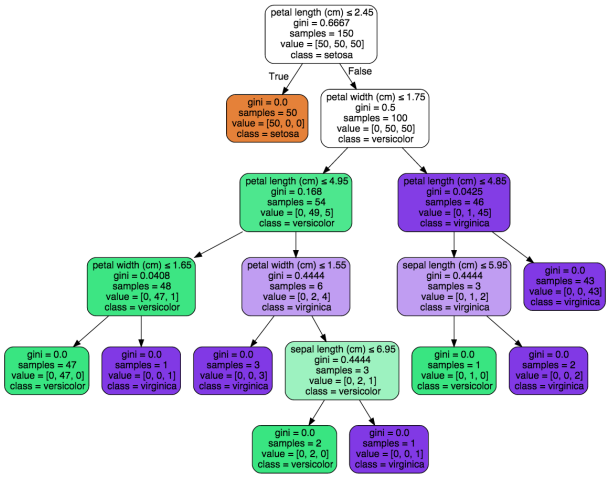


Figure 1.9: Decision Tree Classifier. Image obtained from [19]

1.6.2 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 [12], 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 [13].

Principal Component Analysis

Principal Component Analysis (PCA) uses a subspace in which the the variance direction among basis vectors is maximum in the original space [13]. 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 citeDuda.

1.6.3 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 2

Methodology

2.1 LeNet-5

LeNet-5 [20] 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.*

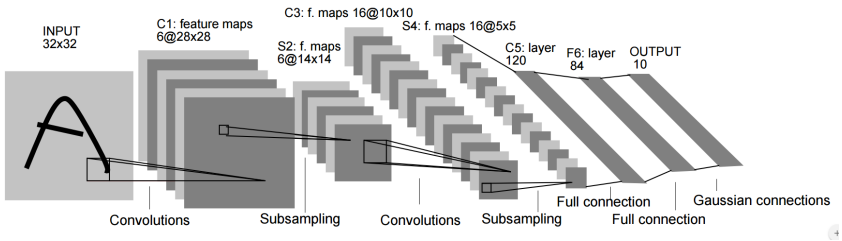


Figure 2.1: LeNet-5 Arquitecture

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 2.1 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.

2.1.1 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 1.4.1. 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 [21]. The decision of stopping the training depends on *the patience* and is chosen by user.

2.1.2 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.

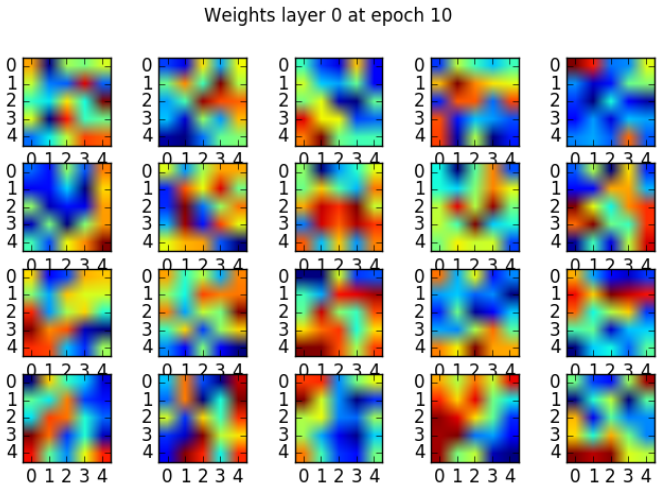


Figure 2.2: Weights at epoch 10 of the first convolutional layer

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 2.3, 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.

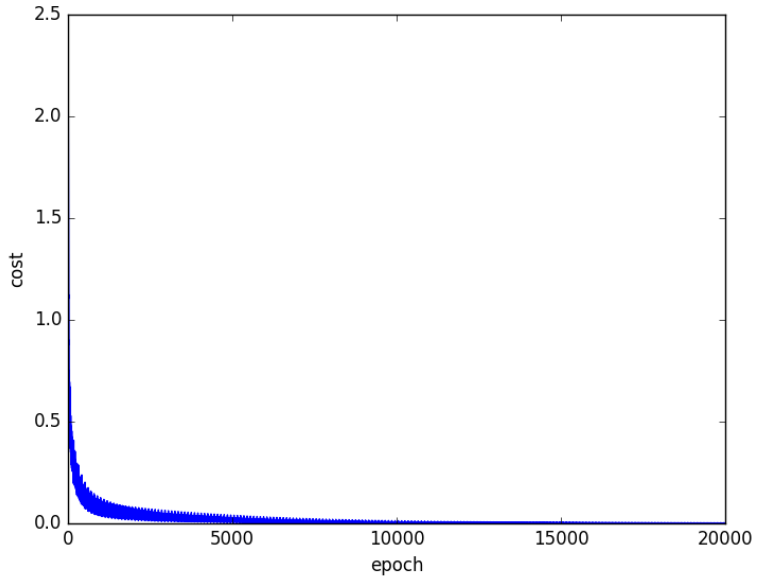


Figure 2.3: Cost function at training running LeNet-5 with MNIST digit database.

The error obtained at validation is represented in figure 2.4, 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.

2.1.3 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.

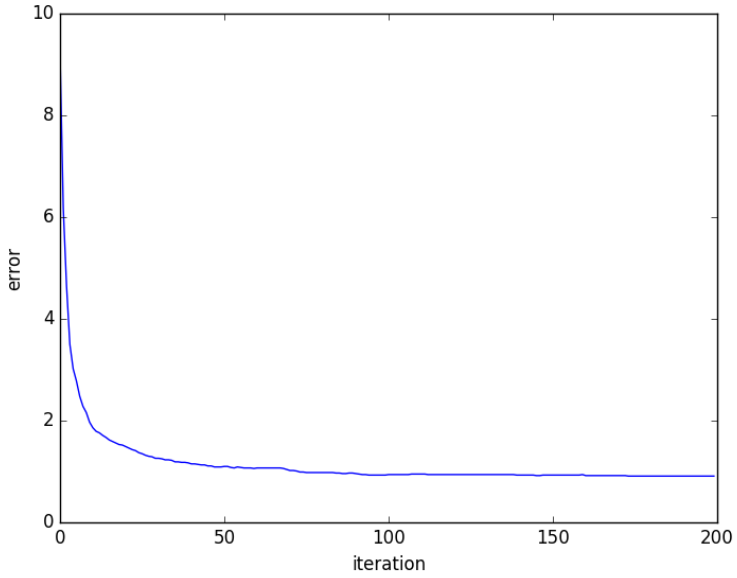


Figure 2.4: Validation error obtained with LeNet-5 with MNIST digit database

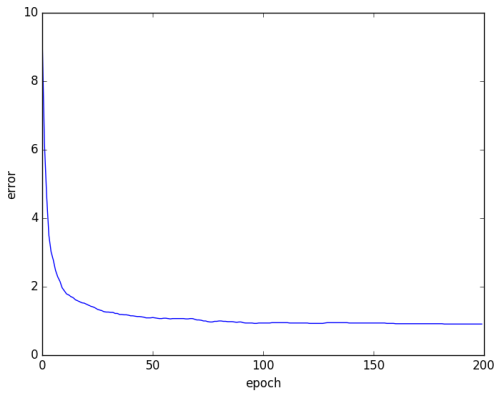
Changing the batch size

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).

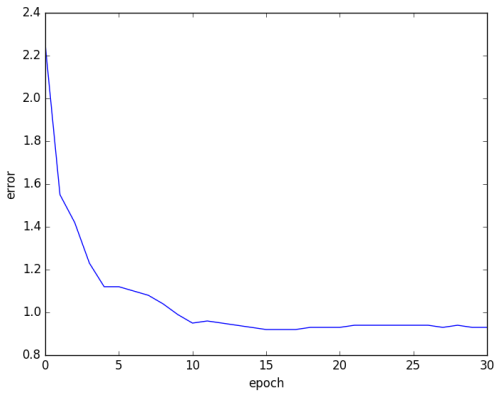
The first experiment is with 20 samples per batch. For the second experiment, the batch size used is 100. In both cases, the training process has been stopped by the early-stopping; at epoch 31^{th} has stopped the first experiment, because from epoch 16^{th} the error at validating was not being improved 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 2.5. 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.

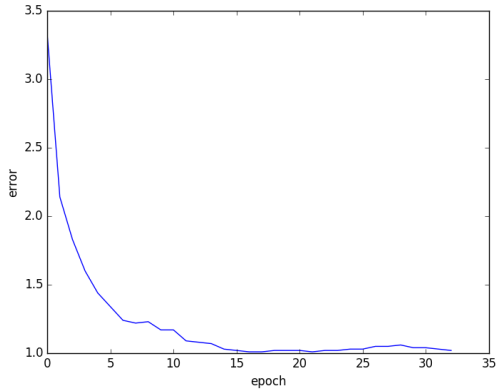
Concluding, more epochs are necessary when the batch size is bigger because there are not enough updates in each epoch [21]. Usually, the value of the batch size used is



(a) batch size = 500 (original size)



(b) batch size = 20



(c) batch size = 100

Figure 2.5: Validation error in each epoch for different sizes of batches.

32 [21] and the choice, generally, is computational.

In figure 2.5 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 [22] is used:

$$W \sim U[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}] \tag{2.1}$$

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.

Above 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 2.6 is represented. Also the validation error (validation cost*100) could be visualized in figure 2.6. The network has been running for 200 epochs.

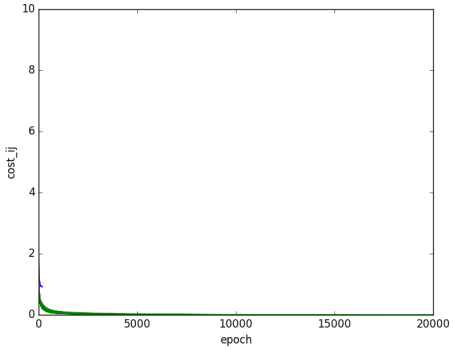
Visualizing the loss during the training, could be affirmed that the network with Gaussian initialization is in a local minimum because the loss has converged as could be seen in figure 2.6(e). The loss of LeNet without being modified (figure 2.6(a)) is the one whose oscillation at training is less than others.

About the error at validation, visualizing the graphs in figure 2.6, 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.

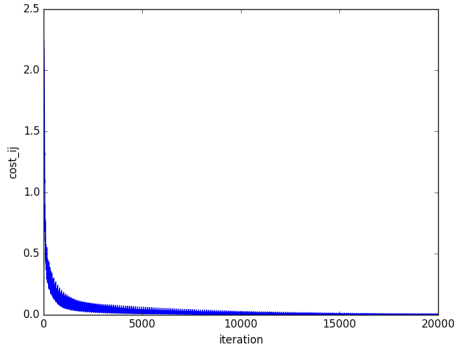
The results obtained, at testing, have been the following ones:

- 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%.

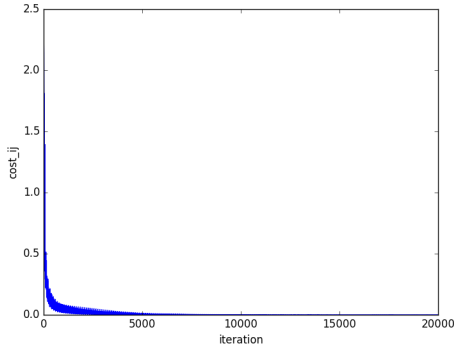
The best configuration for the network is the original one. With Gaussian initialization, the network does not find a local minimum in such a sort of time. Using LRN and ReLu, test result is closer to the obtained with LeNet original, but not as good as the last one. Changing the activation function 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.



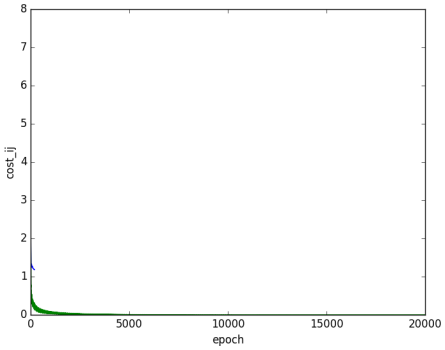
(a) Original LeNet



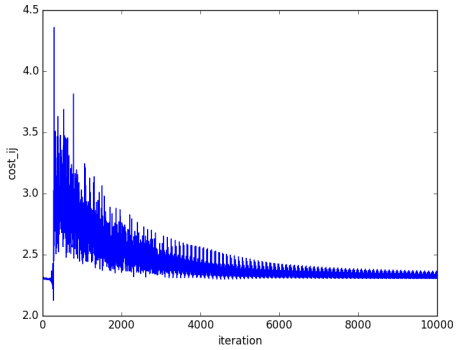
(b) Experiment 1: using LRN



(c) Experiment 2: using ReLu

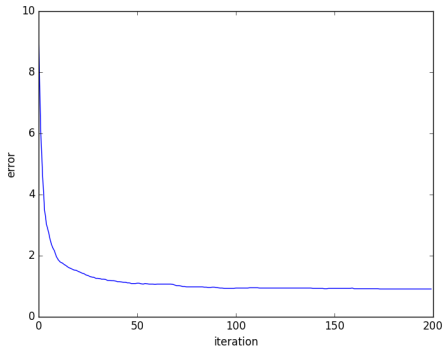


(d) Experiment 3: using ReLu and LRN

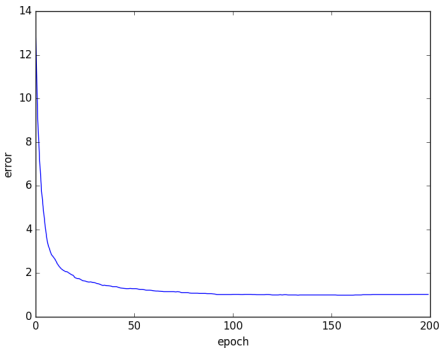


(e) Experiment 4: using Gaussian weights init.

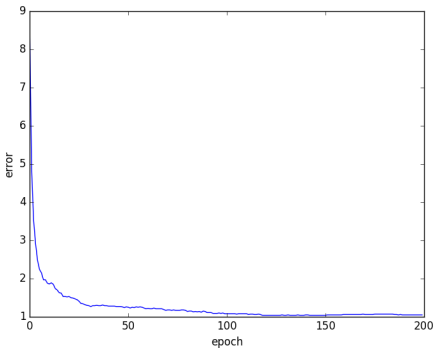
Figure 2.6: Cost function of Lenet and Lenet Modified.



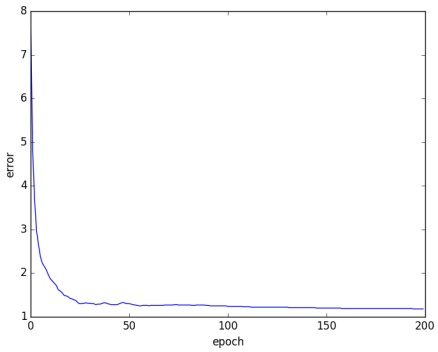
(a) Original LeNet



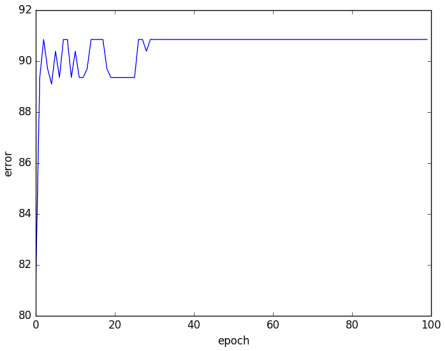
(b) Experiment 1: using LRN



(c) Experiment 2: using ReLu



(d) Experiment 3: using ReLu and LRN



(e) Experiment 4: using Gaussian weights init.

Figure 2.7: Valid error of Lenet and Lenet Modified.

2.2 Start working with Faces databases

Once LeNet-5 architecture has been understanding, the used database is changed because the final goal is using a convolutional neural network with different faces databases.

2.2.1 Using Labeled Faces in the Wild

With the purpose of reading and working with face images, the Labeled Faces in the Wild (LFW) database is used. Because of that, a script has been developed where images are precessed. All images are pseudo-randomized and grouped in training set (49% of the total database), testing test (30% of the total data) and validating set (21% of the data).

To split the data, train_test_split function from sklearn.cross_validation has been used. This function pseudo-randomize the data, so you can repeat the randomized split data in the same way assigning the same seed to the function.

For the next experiment, 500 samples are used for each batch, so 13 batches are going to be used for training, 6 for validation and 6 for testing. Images has been resized to 28x28 size, as MNIST digit database available with the code.

The parameters has not been changed, but the number of epoch that has been decreased to 12, and the number of neurons at the output of the logistic regression that has been changed from 10 to 5748 (the number of classes or different people).

The results obtained are not good, because of the fact that the network has not been optimized to this purpose, the number of epoch should be more and for each class there are a few samples and should be much more.

Figure 2.8 represents the validation error % in each epoch, and it could be seen that in the last epoch, the error is the smallest one, and the test error in that point is 95.125000 %, a really high error rate because of the bag learning procedure.

Changing learning rate

Despite the fact that the configuration of the networks is not to this database, learning rate is going to be changed so it is possible to know how it affects.

In order to know how the net works with different learning rates, it has been changed to 0.001 and the number of epoch has been raised to 50.

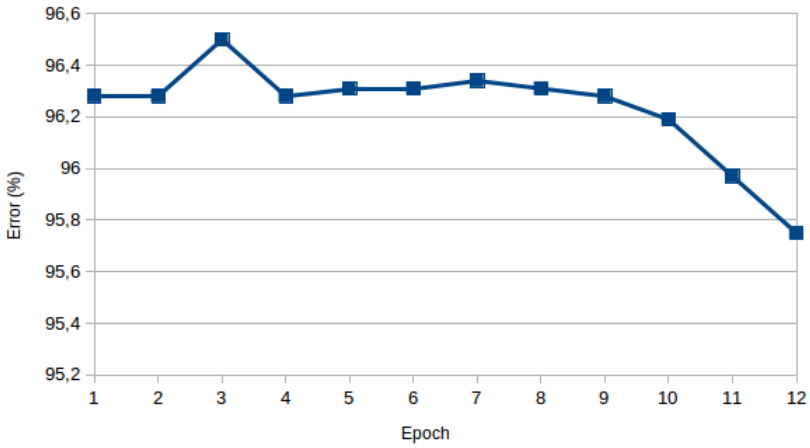


Figure 2.8: Error of Lenet using LFW.

The error at validating in each epoch could be seen in figure 2.9 , where it is shown that the net does not learn because the learning rate is too small and it would need more epochs. The cost function at training of the training could be seen in figure 2.10, where the cost it has been reducing during epochs, but it has not been reduced significantly, from 8.7 to 8.1.

A 97,73% error of test performance has been gotten, which has been obtained in iteration 13.

The conclusion is the learning rate is too small to this configuration of the net and the data given.

If the learning rate is increased to 0.01: Best validation score of 96.266667 % obtained at iteration 481, with test performance 95.733333 %

If learning rate is increased to 0.1:Best validation score of 96.300000 % obtained at iteration 13, with test performance 95.733333 %

If learning rate = 0.5 Best validation score of 96.3 % obtained at iteration 143, with test performance 95.73%

In conclusion, if the learning rate is too big, the network do not get a optimal minimum and if the learning rate is too small it takes too mch iteration to learn or getting to to optimal minimum.

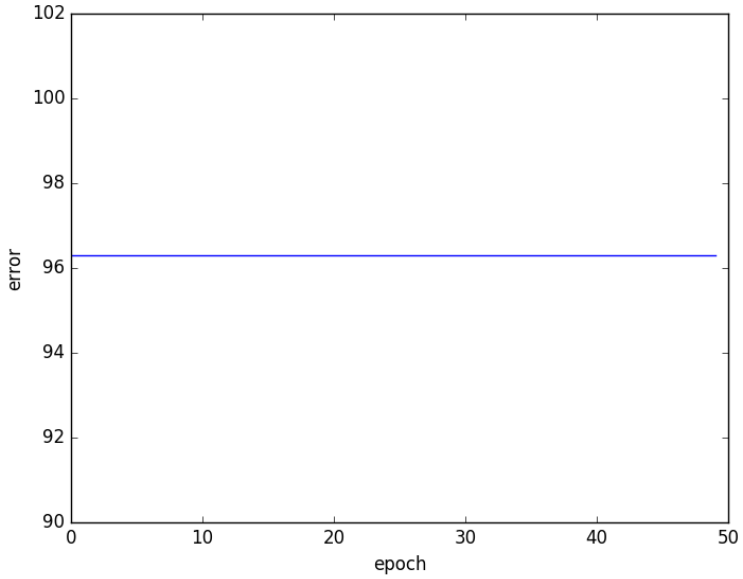


Figure 2.9: Error of LeNet-5 using LFW with a learning rate of 0.001.

Changing convolutional parameters

Convolutional layers are built with the Theano function *theano.tensor.nnet.conv2d*. The size of convolutional layers depends on the number of filters that users would like, the dimension of images, it is not possible to use the same convolutional layer for 3d images (rgb) than grey scale images (1 dimension), and also depends on the high and width user would like to give.

The output of a convolutional layer depends on the size of the filter and the size of input images. At the output, a new bunch of images are created from the input images and the characteristics of the layer.

Also, it is important to consider the batch size, because the layer is not fed by a individual image; the layer is fed by the bunch of images.

Lets have a bit of fun with convolutional layers, the number of filters and the size of them it is going to be changed. A learning rate of 0.1 is going to be used for 50 epoch.

In the first example, the number of filters of the first layer is going to be increased from 20 to 40, and the number of the second layer from 40 to 60.

In figure 2.14 could be seen the error which has been gotten in each epoch, where the best vest validation score of 94.9% has been obtained at iteration 559, with test

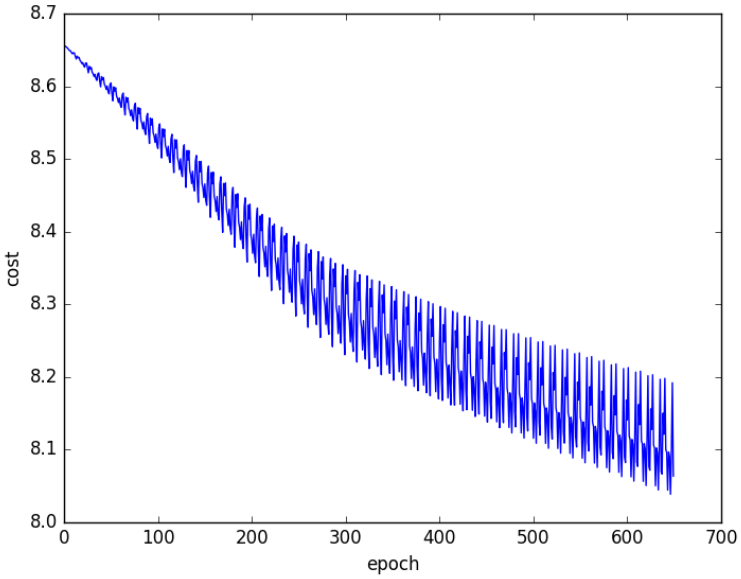


Figure 2.10: Cost of Lenet using LFW with a learning rate of 0.001.

performance 95%.

If the number of filters has been increased, the time that the code takes to run is increased significantly. Also, the results of the network changes, this is a parameter that should have in consideration.

Also, it is possible to modify the filter shape, in the previous examples, the size of both kernels were 5x5, in this example, the second one, the first convolutional layer would have a filter size of 3x3, the second one would keep its original size.

It is interesting seeing how the error of the network has been improved when the number of kernels has been changed and, in this example, has been improved too when the size of the filter has been changed.

The error in each epoch of this example could be seen in figure 2.15, where the best validation score obtained has been 94.567% error iteration 611, with test performance error 93.967%.

In order to see the difference between using a big size filter and one of a small size, in the next example, third example, 40 and 60 kernels are going to be used, and the size of each one is 3 and 10 for layer 0 and layer 1 respectively. In epoch number 40, the weighs of each layer has been saved and in figure X are represented. At irst sight, it is possible to see that with a big size.

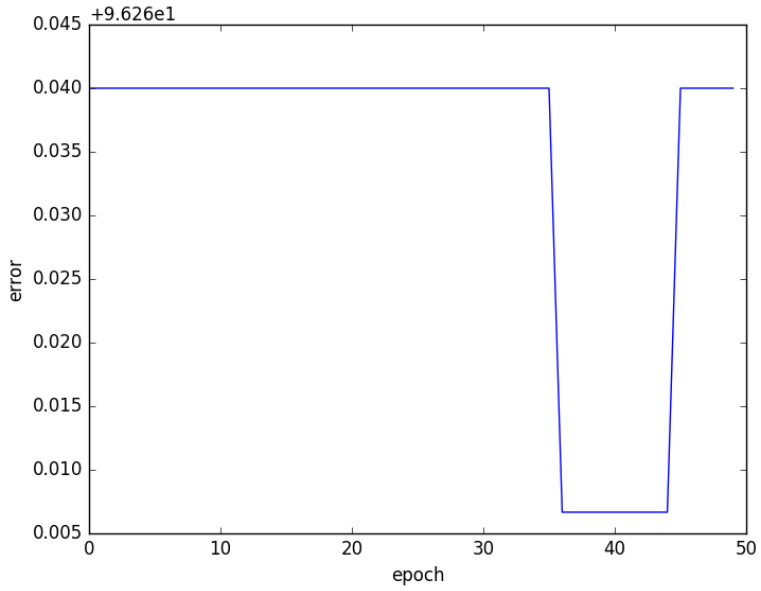


Figure 2.11: Error of Lenet using LFW changing learning rate to 0.01.

Also, it is possible to see the output at the convolutional layer, in figure 2.17, the first 25 output images are shown for both layers.

The result of the third example is a best validation score of 95.73 % obtained at iteration 546, with test performance 95.23 %.

The cost function of those three experiments are represented in 2.18

Using ReLu as an activation function instead of tanh

Originally LeNet uses as activation function tanh(), but in this section, ReLu (Rectified linear units) activation function is going to be used. In equation 2.2 is shown how it is defined.

$$f(x) = max(0, x) \tag{2.2}$$

The error and the cost in each epoch could be seen in figure 2.19

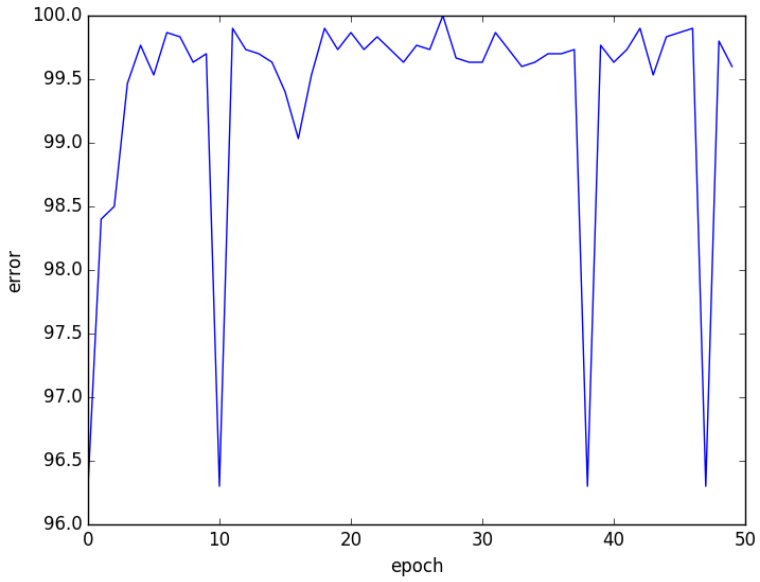


Figure 2.12: Error of Lenet using LFW changing learning rate to 0.1.

2.2.2 Using FRAV dataset

One of the databases used with the final architecture and configuration of the network is FRAV database. LeNet-5 is tested with this database in this current section.

The experiments made with this database are just with RGB images, so 939 images of people has been used, 489 samples are used for the training subset, 162 samples has been used in the validation subset and 279 samples in the test subset.

Because images has not the same shape, they have been re-sized into 252x180, this new shape is proportional 0.7*height = weight because all images studied save that proportion. In addition, making images smaller also gives the security of not having memory problems, because the huge quantity of used images.

The network has been tested with this databases in the two ways of classify images, with two classes (genuine and attacks) and five classes (genuine and four classes, one per type of attack).

The first experiment made was based in the neural network LeNet, without changing parameters, but batch size because 500 is too big, there are not enough samples:

– 25 epoch.

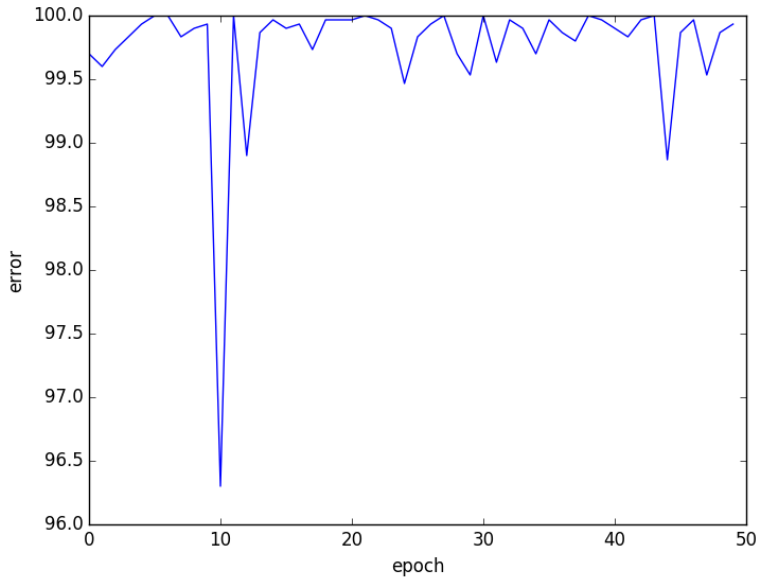


Figure 2.13: Error of Lenet using LFW changing learning rate to 0.5.

- 20 and 50 kernels of each convolutional layer.
- 50 batch size.
- 0.01 learning rate.
- Logistic regression with 10 neurons.

The obtained results are with 5 classes classification and there were not as bad as the one with labeled faces in the wild (LFW), in this database there are more samples per class.

Figure 2.20 shows the validation error of five classes, the best validation error has token place in epoch 7 with test performance 60.4%. It has token 68.78 minutes to run.

In figure 2.21, the validation error in different epoch could be visualized using two classes to classify. It could be seen that in each epoch the validation error is the same. Test error performance at the first epoch is 23.2 %. The total time of the running has been 64.0167 minutes.

The time in both cases, classifying with two or five classes are similar, but not the result, is better when just two classes are used and no differentiation is made among

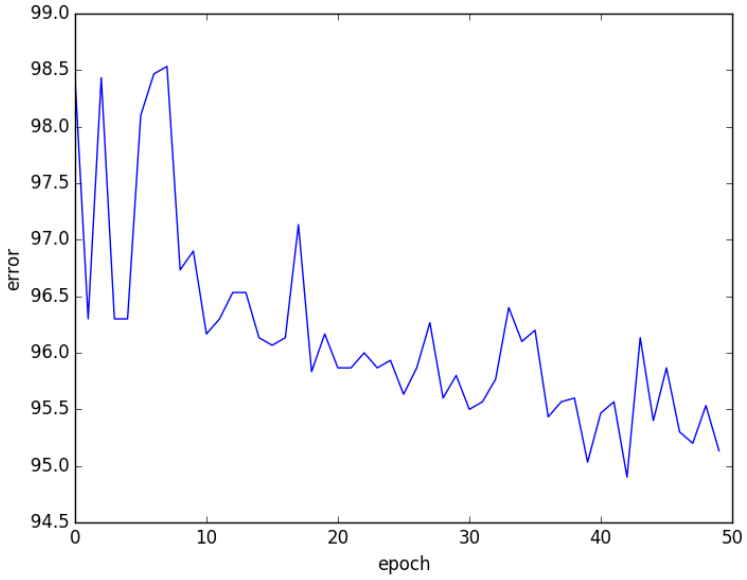


Figure 2.14: Error of Lenet using LFW changing number of kernels in example 1.

attacks. There would be need more samples and more time to get a better score if five classes classification is wanted.

2.3 Defining a new architecture

From the LeNet-5 code, which is proportioned by deeplearning.net, changes has been developed to build a new architecture [23].

The new architecture of the Convolutional Neural Network is formed by two convolutional layers, the first one with 48 filters and the second one with 96. The size of the filters is 5x5. Each convolutional layer is followed by a max pool layer of size 2x2.

The activation function has been changed to the rectified linear activation function (ReLu), a function implemented in Theano has been used to this purpose *theano.tensor.nnet.relu*. For the weight initialization, a normalized distribution of weights and bias, the same which was implemented in LeNet-5: weights are sampled randomly from a uniform distribution.

Because of the huge number of images, the batch size has been changed to 20. And the learning rate is 0.001. The number of neurons at the output of the hidden layer are 100. The classifier is the sigmoid function. 25 epoch have been used for training the network.

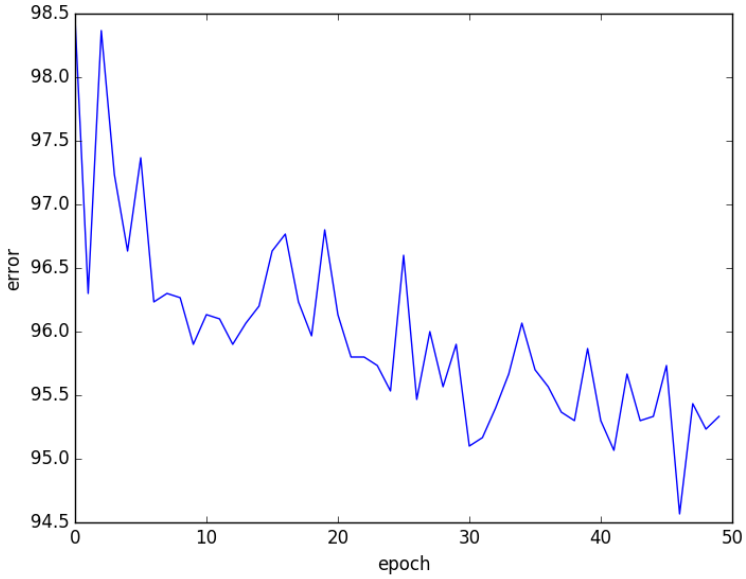


Figure 2.15: Error of Lenet using LFW changing number of kernels and the filter size in example 2.

the used database in this experiment is Casia database and its samples have been pre-processed, images are resized into 52x104, the relation between width and height is proportional to the original size of images, but they have not the same shape. At the time to split the data into train, test and validation, different seeds have been used with the purpose of test the net with different combination of the data.

two different ways of using classes has been used, first, two classes has been used, one the imges of the genuine people and another class with the atacks, the last way of separate classes is assigning one different class to a different attack.

2.4 Regenerating the databases

The databases have been generated in a different way from the used before. In this new way, the possibility of access to the misclassified samples is possible. SO the characteristic of misclassified images could be studied. This is not used now, but in the future it could be useful.

In order to build the new database, a script as been programmed manually because

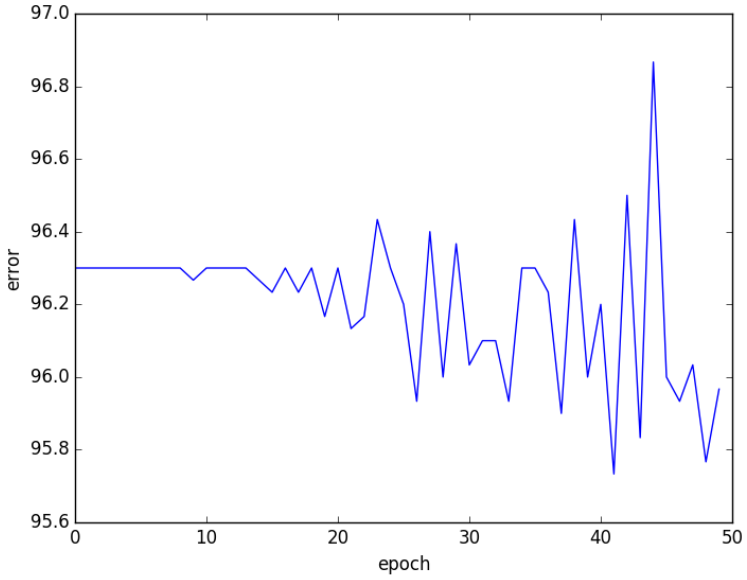


Figure 2.16: Error of Lenet using LFW changing number of kernels and the filter size in example 3.

the test-train split sklearn function does not let know what image has been read and now it is possible to compare the right class of each image with the predicted one.

All this work has been changed because it is interesting to know the characteristics of people (if has beard, glasses or long hair) that are not well classified.

It has been necessary to shuffle the data twice in order to mix it properly and training, testing and validating test would have data of the five classes.

Adding images in characteristic level

In order to get this particular goal, each image is re-sized proportionally to original image to 52x104 dimensions. Each NIR image is appended to its correspondent RGB image.

In order to create a list of images where images are not putted in order, each image followed by another could be of a different class. Two shuffles has been necessary, the first one with a seed = 0.5 and the second one with a seed = 0.1, so the randomize order of images could be repeated.

For training, the 70% of the total data has been used, for testing the 20% and for

	Class 0	Class 1	Class 2	Class 3	Class 4	Total of samples
Training set	94	119	116	145	76	550
Testing set	33	38	37	7	42	157
Validating set	30	0	4	5	39	78

Table 2.1: Distribution of samples FRAV (RGB + NIR) database

validating the testing 10%. There are 157 images in each class and the distribution is represented in the table **??**.

The code runs for 150 epoch with a learning rate of 0.001 and a batch size of 20 images.

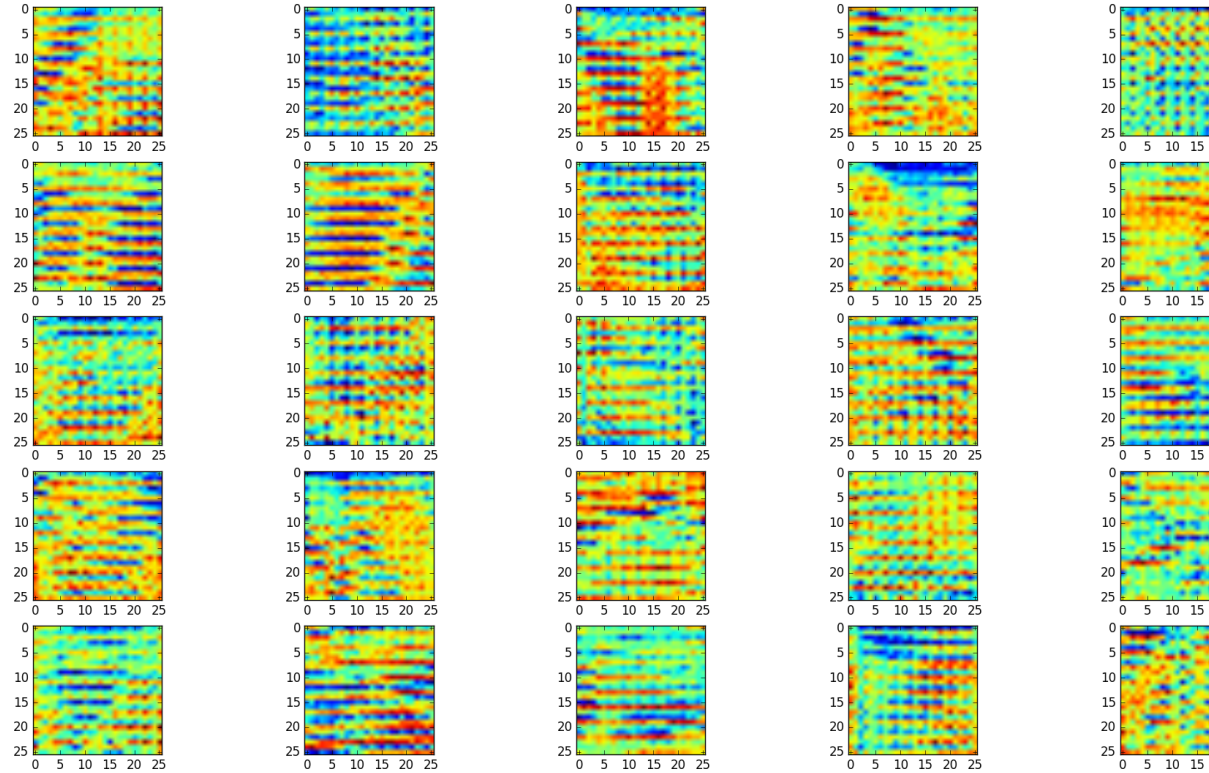
The disadvantage of using mini-batches is that there are some images remain and the quantity os less than a mini-batch, that images are not used, so from 157 testing images, just 140 has been used.

The test error that has been gotten is 30% at iteration 1863 where the best validation score was gotten (26,67%). Where:

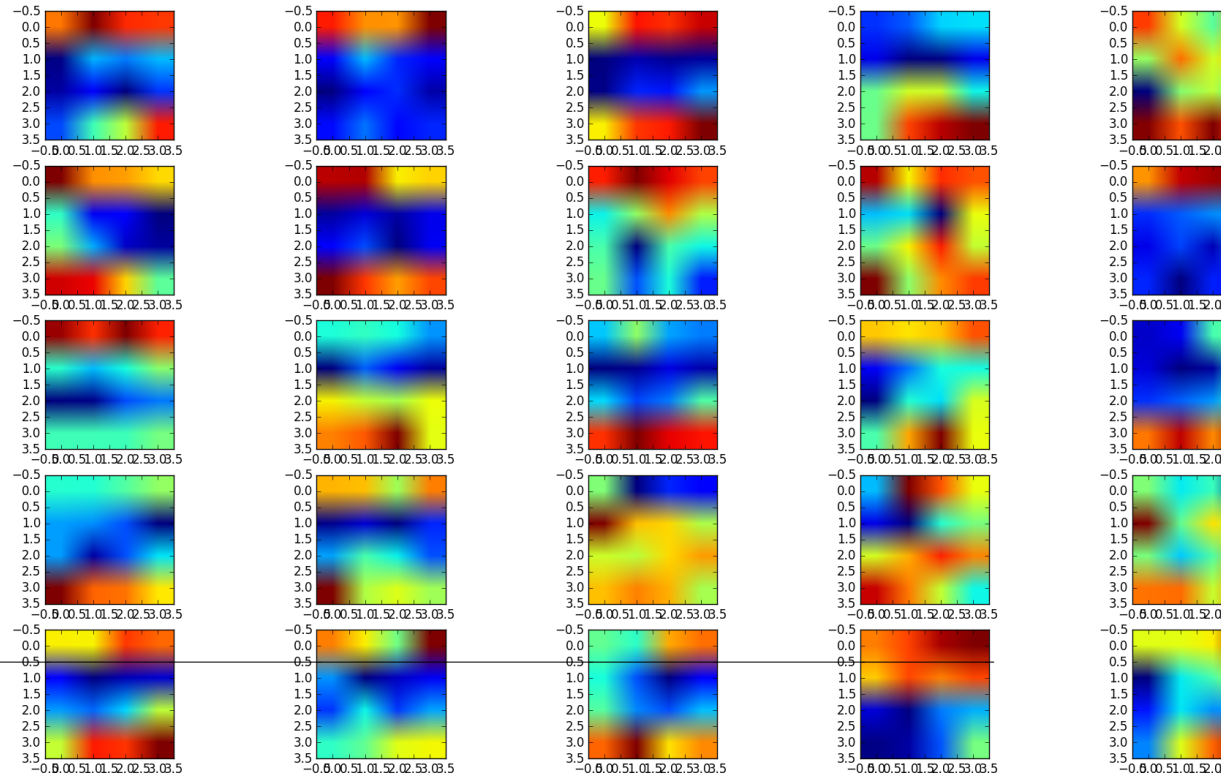
- Class 0 has been misclassified 14 times
- Class 1 has been misclassified 6 times
- Class 2 has been misclassified 7 times
- Class 3 has been misclassified 2 times
- Class 4 has been misclassified 2 times

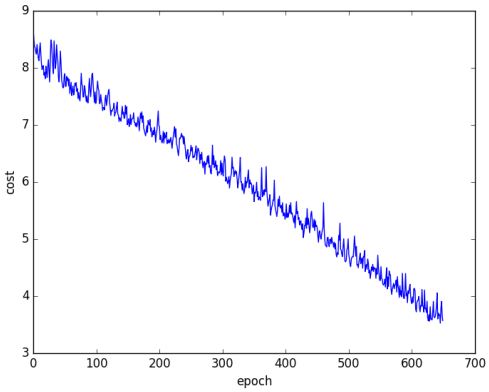
To get that results 3,04 hours was needed to run the code.

25 first images at the output of the first convolutional layer.

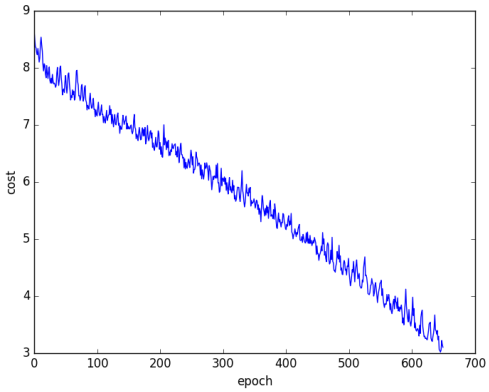


25 first images at the output of the second convolutional layer.

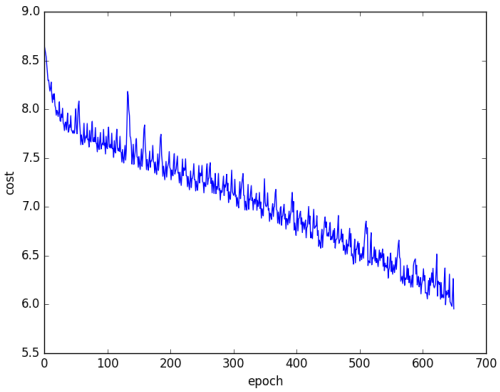




(a) Example 1

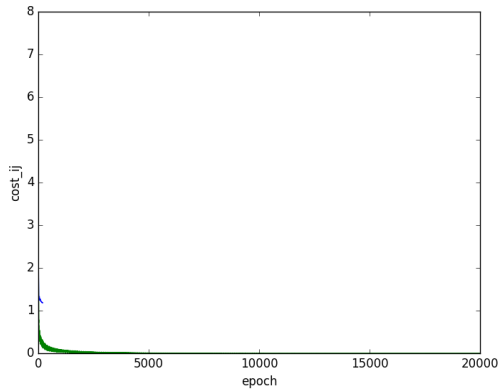


(b) Example 2

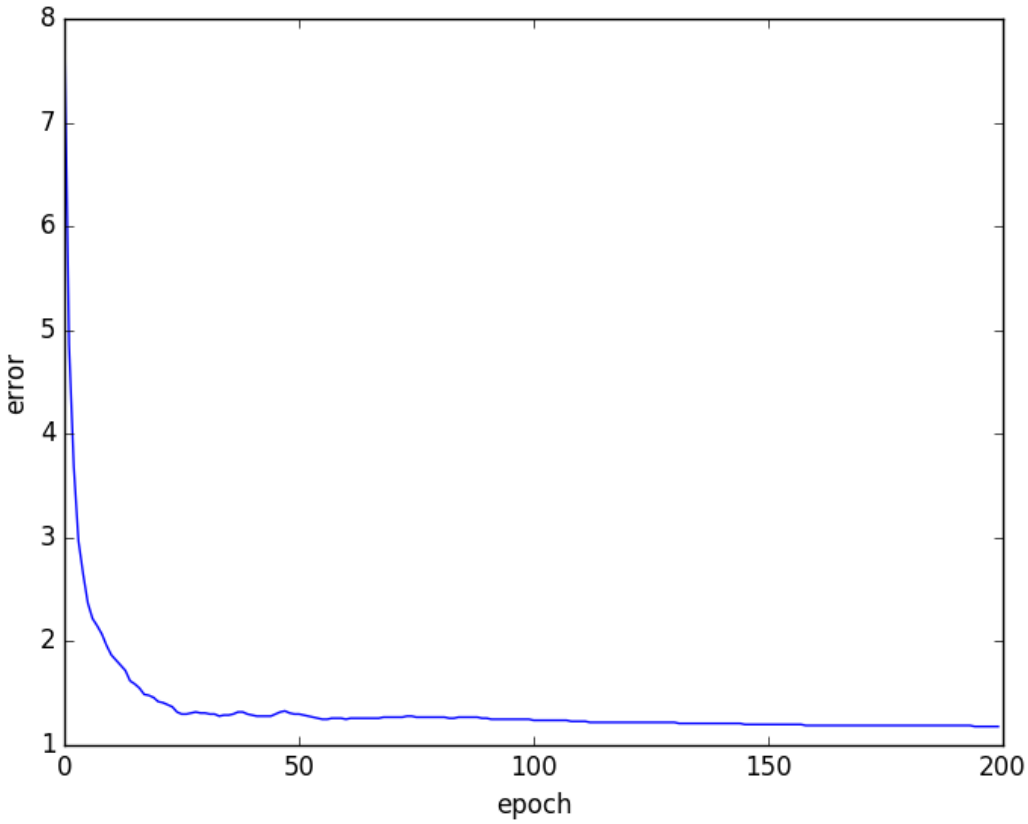


(c) Example 3

Figure 2.18: Cost function at training of the three examples chanching the convolutional parameters.



(a) Cost at training



(b) Error at validation

Figure 2.19: Error and cost using ReLu instead of tanh

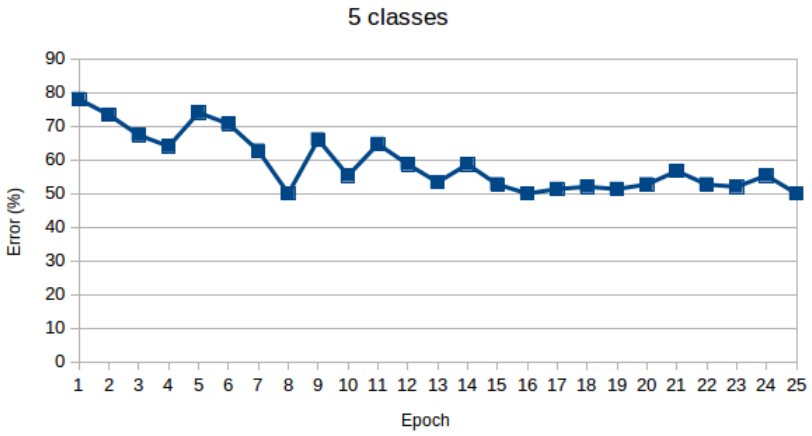


Figure 2.20: Error using FRAV database and five classes.

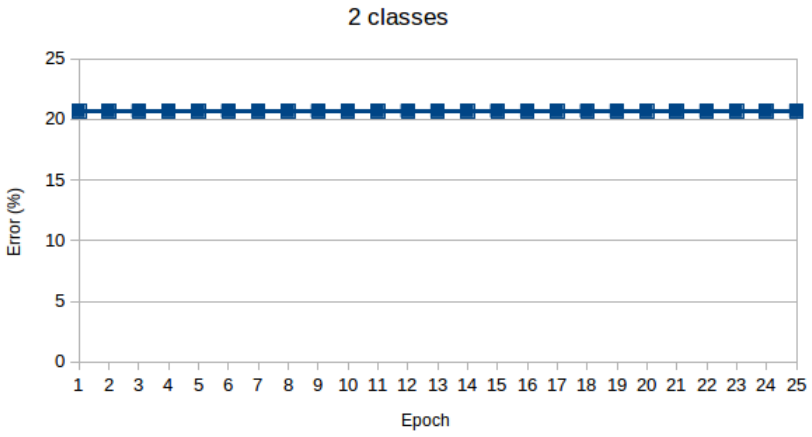


Figure 2.21: Error using FRAV database and two classes.

2.4.1 Architecture implemented in Casia videos

In this section, the architecture used in the paper of the Casia videos would be repeated partially. The paper is learn convolutional neural network for face anti-spoofing.

The architecture followed in the paper is the same used in Imagenet, it is formed by five convolutional layers, followed by three fully-connected layers. The two first conv layer and the last one are followed by a max-pool layer. The two first conv layers are followed by response normalization layers too. Authors use ReLu like activation function in each layer. The first two fully-connected layers are followed by two dropout layers, and the last layer output is followed by softmax layer as a classifier.

Authors do not explain anything else about the architecture. In the paper of Imagenet is said that:

- The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.
- The first convolutional layer filters the 2242243 input image with 96 kernels of size 11113 with a stride of 4 pixels.
- The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size 5 5 48.
- The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers.
- The third convolutional layer has 384 kernels of size 3 3 256 connected to the (normalized, pooled) outputs of the second convolutional layer.
- The fourth convolutional layer has 384 kernels of size 3 3 192.
- The fifth convolutional layer has 256 kernels of size 3 3 192.
- The fully-connected layers have 4096 neurons each.
- The maxpool size filter si 2x3 because it reduces the error 0.4% - 0.6% compared to 2x2 filters.
- In Imagenet uses 224x224 images.

It is not explained how authors change the architecture of Imagenet.

The data used to this experiment is the Casia database, the one that uses in the paper, although authors, use Reply-attack too.

The data is composed by two folders, a training folder and a test data, in each folder there are some user folders, in each user folders there are two videos from the real user, tho videos of the user with a mask, two videos with the mask and with a hole in the eyes and two videos with a digital screen where a user is shown in the screen. 3 attacks are presented, so four classes are used (real users, users with mask, users with mask and eyes and a digital screen). It is not specified how many frames authors use, so it s supposed that they use all the video.

For each frame, the face is looked for in the image with viola jones algorithm implemented in opencv, and the image is cropped and saved, but in that image cropped there is no background, so different scales, 1.4, 1.8, 2.2, 2.6, are used to get background, because in learn convolutional neural network for face anti-spoofing and then images are re-sized to 128x128.

In order to carry out this experiment, a better computer has been needed because it was no possible to run with the same that the utilized in the previous experiments.

But with a better computer, it is not possible read more than 2 or 3 frames per video to run the net, because it has a huge architecture with a big quantity of filters per layer.

So the experiment with the videos has not been possible to be carried out.

In addition, is it not possible to carry out the architecture of imagenet with the size of Casia images because if it is started with 128x128 images, at the end, images sizes are images of ¡ 1px.

In order to know how strides work, a python file has been created called understanding strides, in which a pickle format file is loaded. This pickle format file is the output of a layer, and it is possible to see the size of images of the layer. So it is possible to know the size of images after striding and this number could be saved to be written in the conv layer.

In this computer, the theano function used to build the maxpool layer has changed bacuse of Theano version, the previous function used in Theano *theano.tensor.signal.downsample.max_pool_2d* has been replaced by *theano.tensor.signal.pool.pool_2d* using the mode *max*.

To sum up, In this first experiment, the architecture is formed by the convolutional and pooled out layers but without strides. The folder where this experiment has been developed has been in frav_casia_imagenet. The architecture is just based in conv, pool and hidden layer (no dropout, softmax or normalization layer). In figure 2.22 it is possible to see how the error is descending in each epoch and get stabilized with a 5% aprox. error.

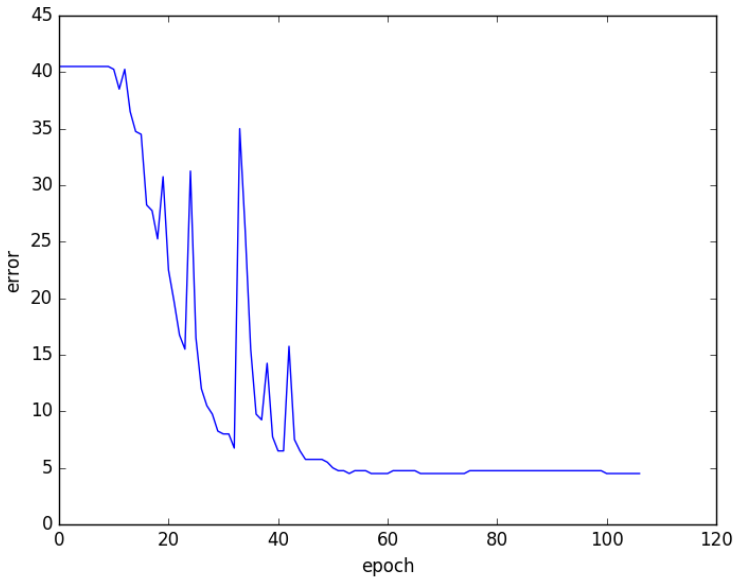


Figure 2.22: Error of Lenet in the first try to build imagenet.

The code had been running for 364,39 hours. And the true positive rate, true negative rate, false positive rate and false negative rate are available in table **??**.

TP	TN	FP	FN
156	623	10	11

Table 2.2: TP, TN, FP, FN rates in the first trying of imagenet.

2.4.2 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 2.23.
- frav_gaussian_initinizialization: Gaussian weight Initialization. Learning rate = 0.001. SOFMTAX used as classifier. Figure 2.24 .
- svm_gauss: Using SVM (with RBF kernel) as classifier. Gaussian weight Initialization. Learning rate = 0.01 2.25.
- svm_genera: Using as a classifier SVM with RBF kernel. Learning rate = 0.01 2.33.

– 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 2.23. Second with the same learning rate, but Gaussian initialization for weights 2.24. Third, decreasing the learning rate to 0.01 and with normal initialization 2.26 and the last one, whit the same learning rate but with Gaussian weights initialization 2.25.

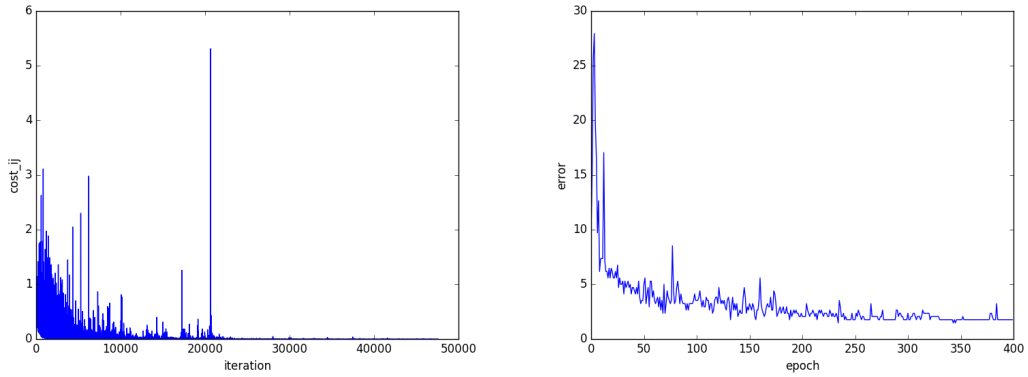


Figure 2.23: Cost at training and error at validating. Normal initialition. Learning rate = 0.001 (frav1).

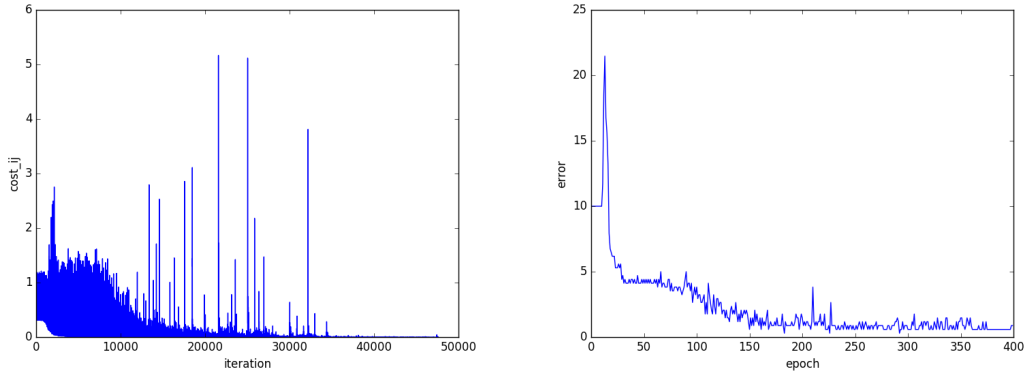


Figure 2.24: Cost at training and error at validating. Gassian initialization. Learning rate = 0.001 - frav_gaussian_init

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 2.27

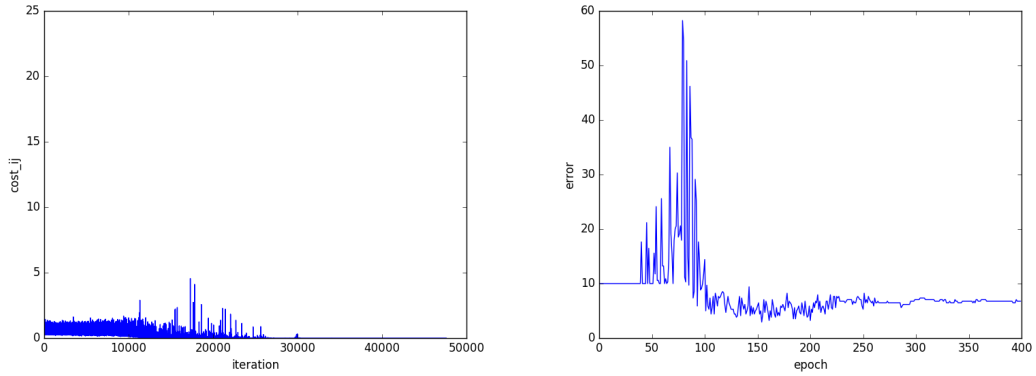


Figure 2.25: Cost at training and error at validating - Gaussian initialization. learning rate = 0.01 frav svm_gauss.

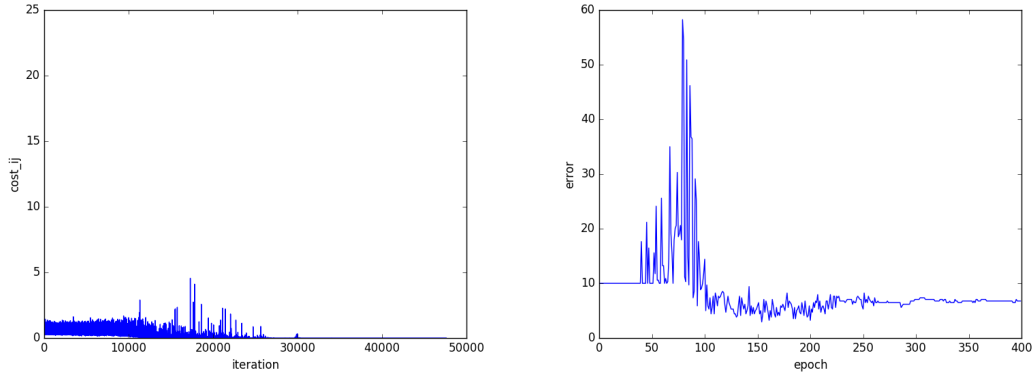


Figure 2.26: Cost at training and error at validating - Noraml initialization. learning rate = 0.01 frav svm_general.

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 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.

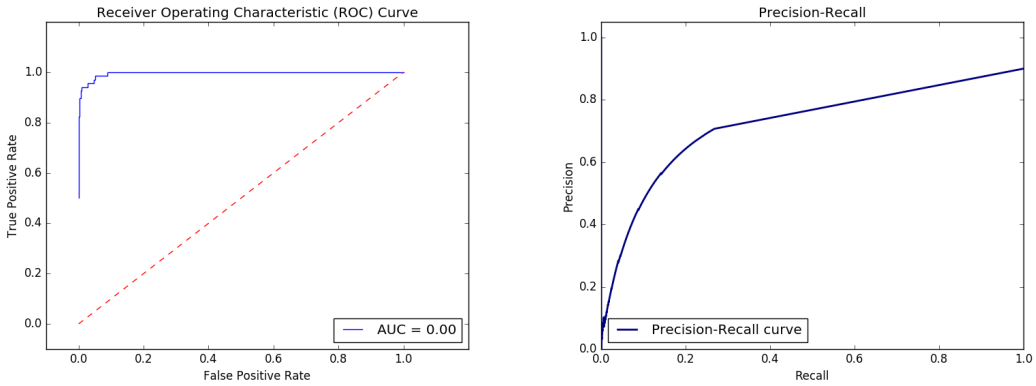


Figure 2.27: ROC and Precision- Recall courve - Noraml initialization. learning rate = 0.01 frav svm_general.

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

casia results

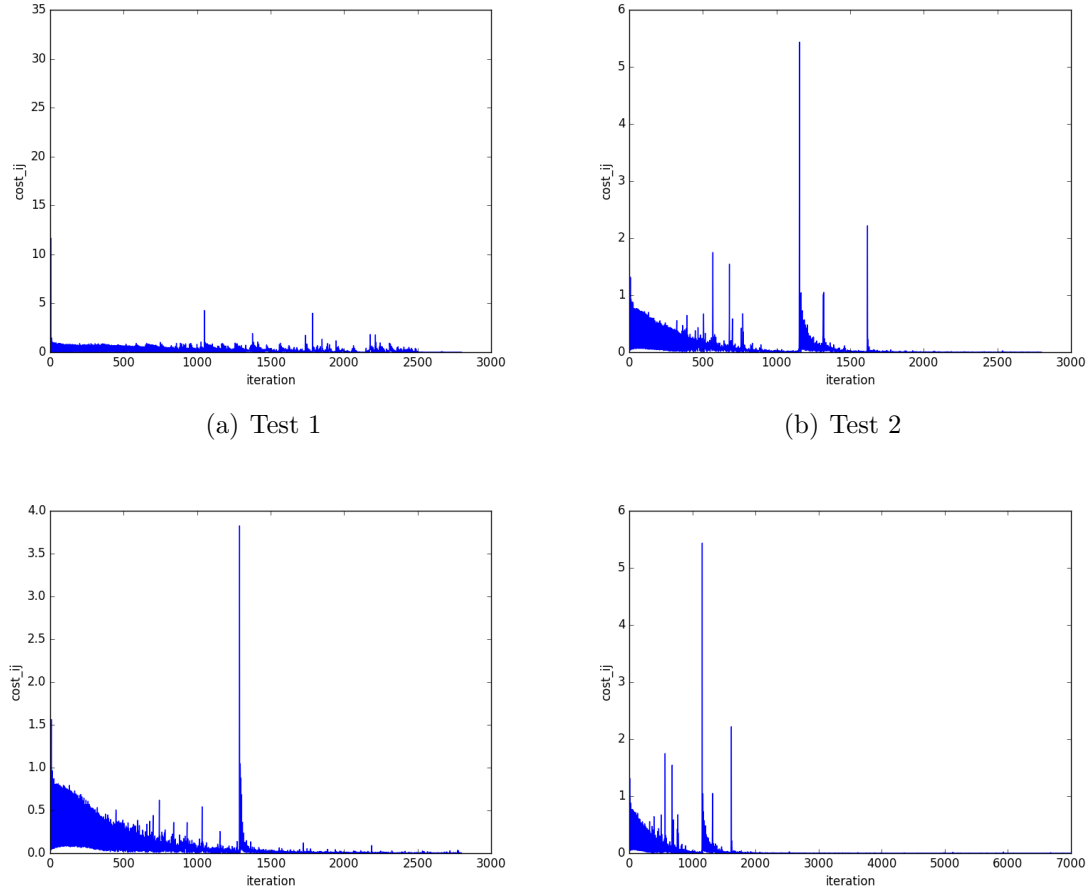
For Casia, different experiments has been carried out in order to train the network, using SOFTMAX as classifier and normal weight initialization:

- 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 2.28 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 2.29 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

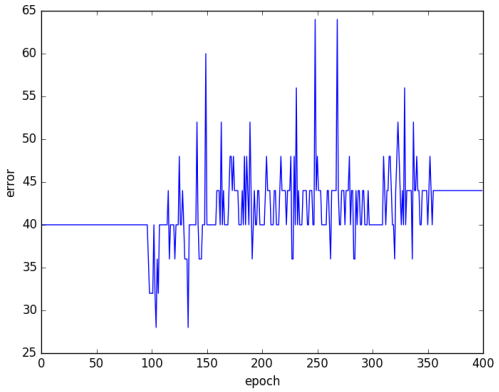
(b) Test 2

(c) Test 3

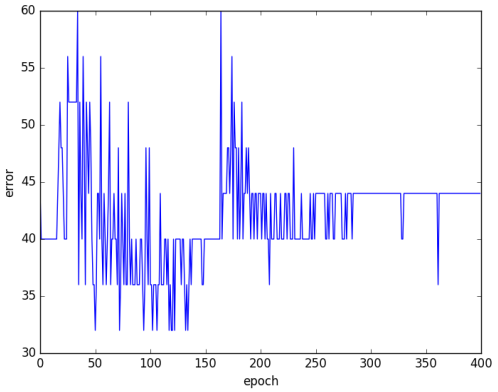
(d) Test 4

Figure 2.28: Training loss of four test Casia database

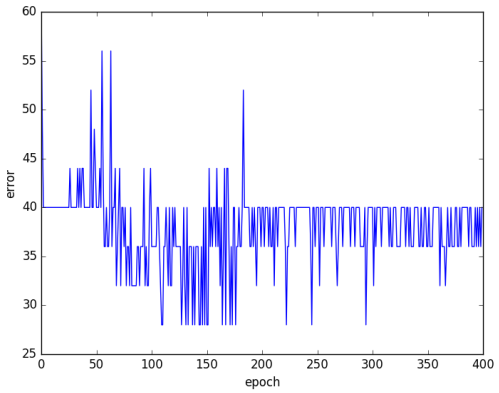
As the same way that in the first experiment that FRAV database converges is that would be expected from Casia, but it has not gotten.



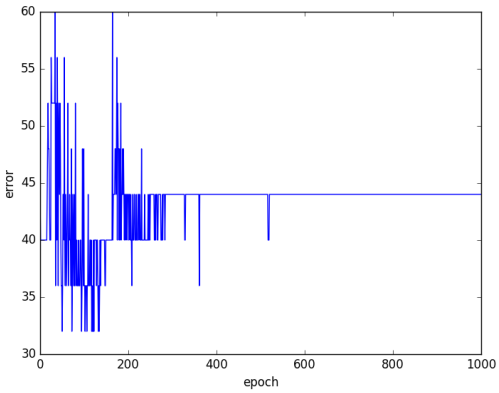
(a) Test 1



(b) Test 2



(c) Test 3



(d) Test 4

Figure 2.29: 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 2.31. Second when the weight initialization used is Gaussian 2.30.

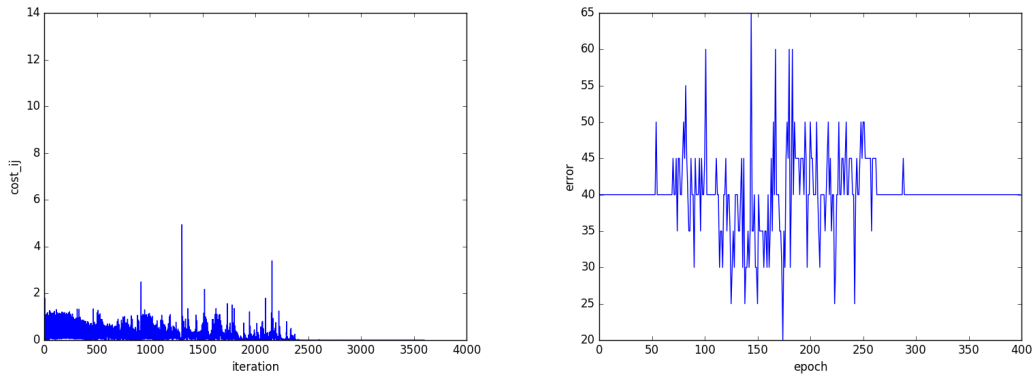


Figure 2.30: Cost at training and error at validating -casia svm_gauss.

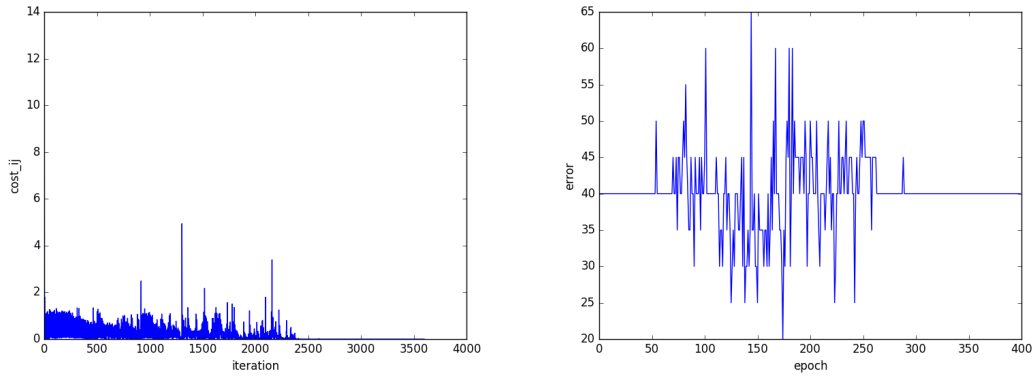


Figure 2.31: Cost at training and error at validating -casia svm_general.

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 2.32 and normal distribution initialization 2.33 and classifying the test with SVM (RBF kernel and linear) and softmax.

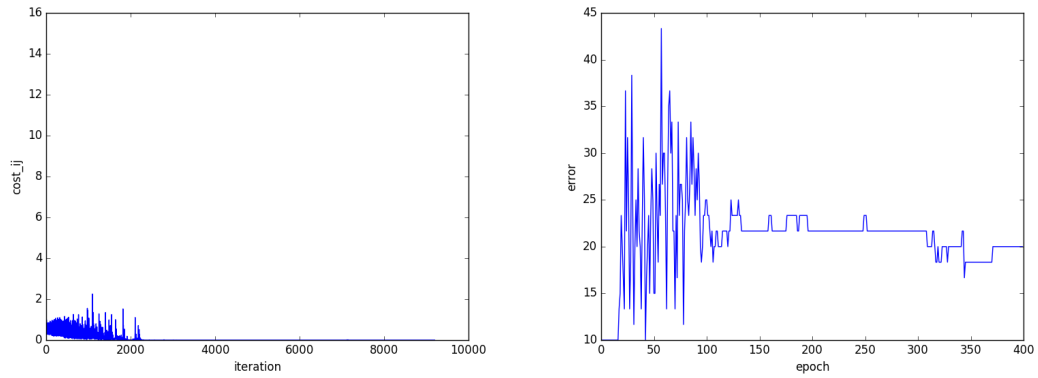


Figure 2.32: Cost at training and error at validating - mfsd svm_gauss.

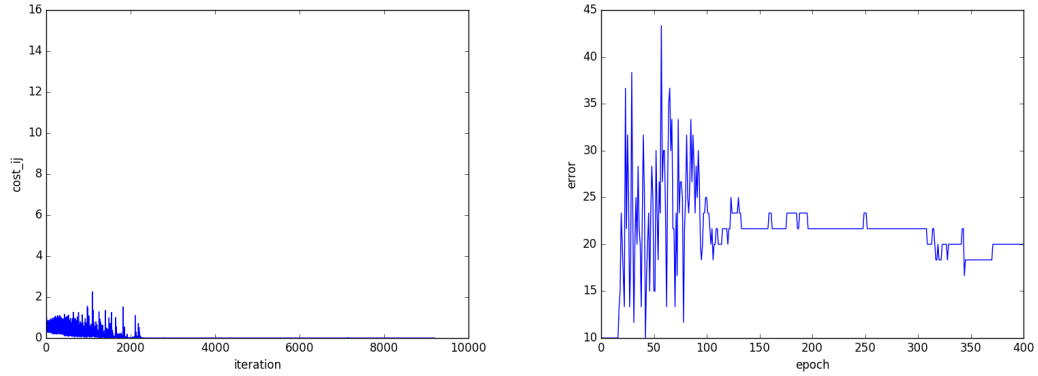


Figure 2.33: Cost at training and error at validating - mfsd svm_general.

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

2.4.3 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).

2.5 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 classifier utilized is the logistic regression.

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 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.

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.

With the purpose of having a more general results, all the training, validation and testing (with the classification) processes have been repeated three times, those times are differentiated among them in the seed if the pseudo-randomized initialization of the weights.

2.6 Conclusions

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

- From [13]: 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.

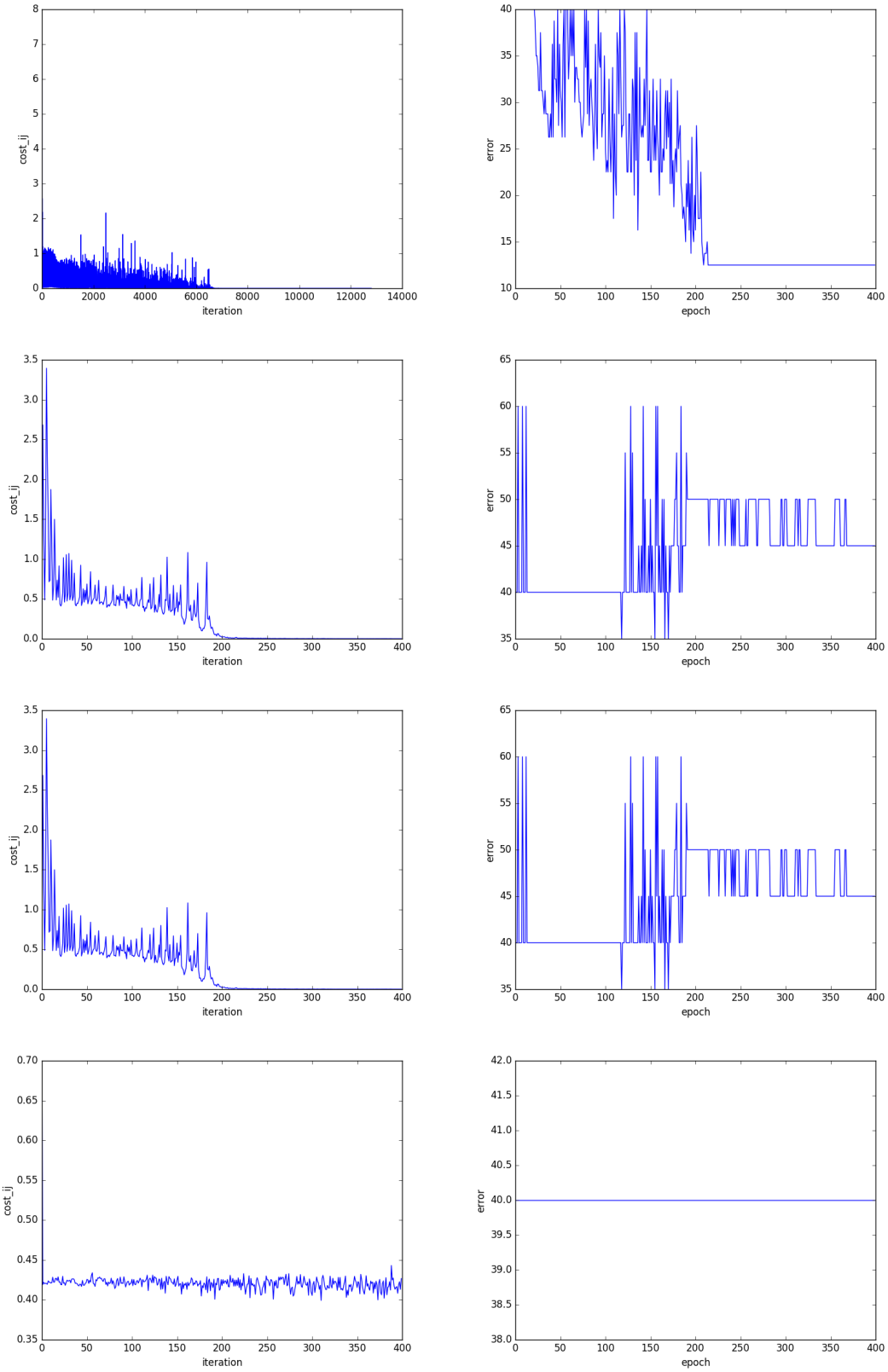


Figure 2.34: cost and error of the tree experiments with frav.

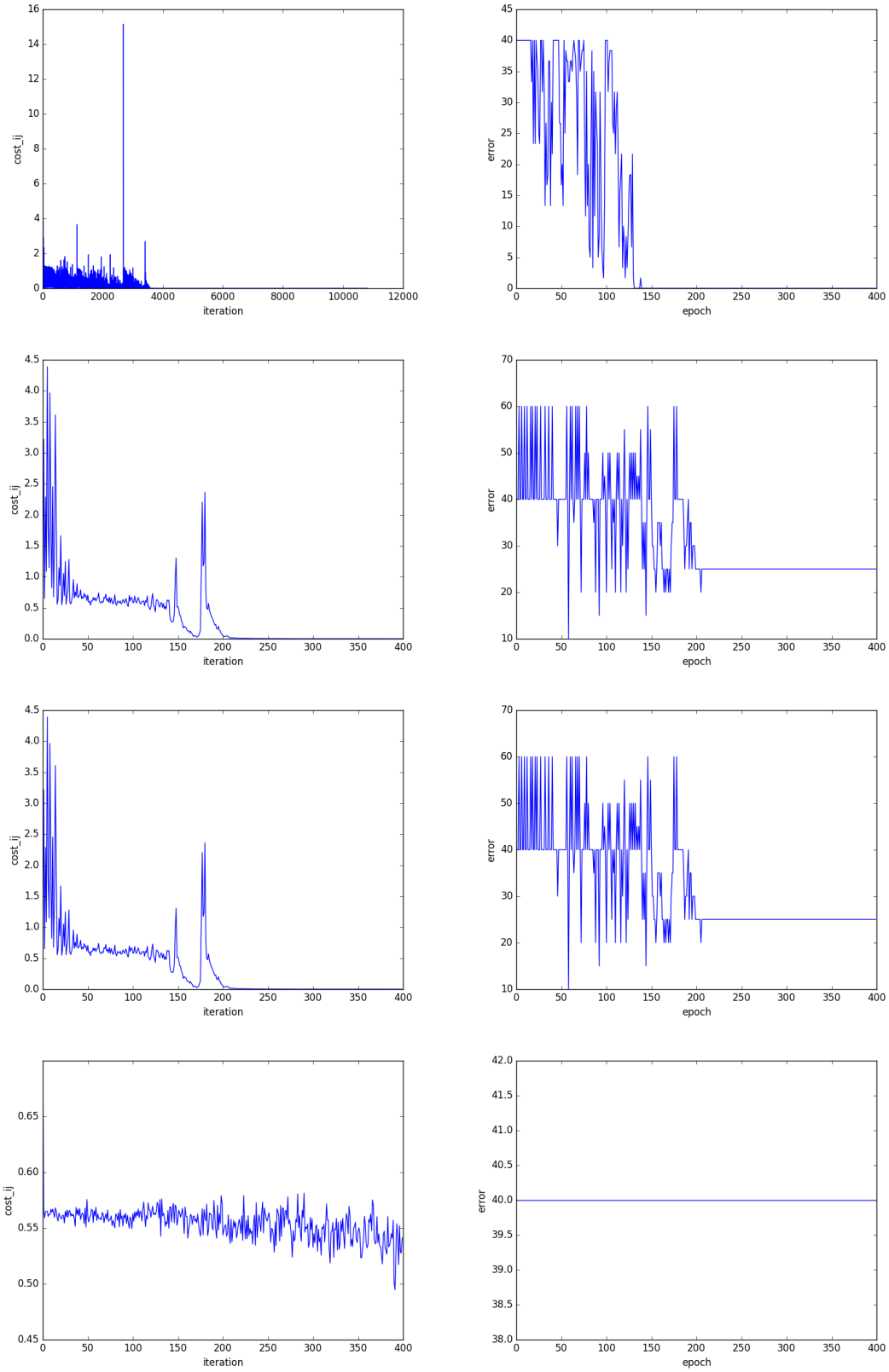


Figure 2.35: cost and error of the tree experiments with FRAV (rgb + nir) image level images.

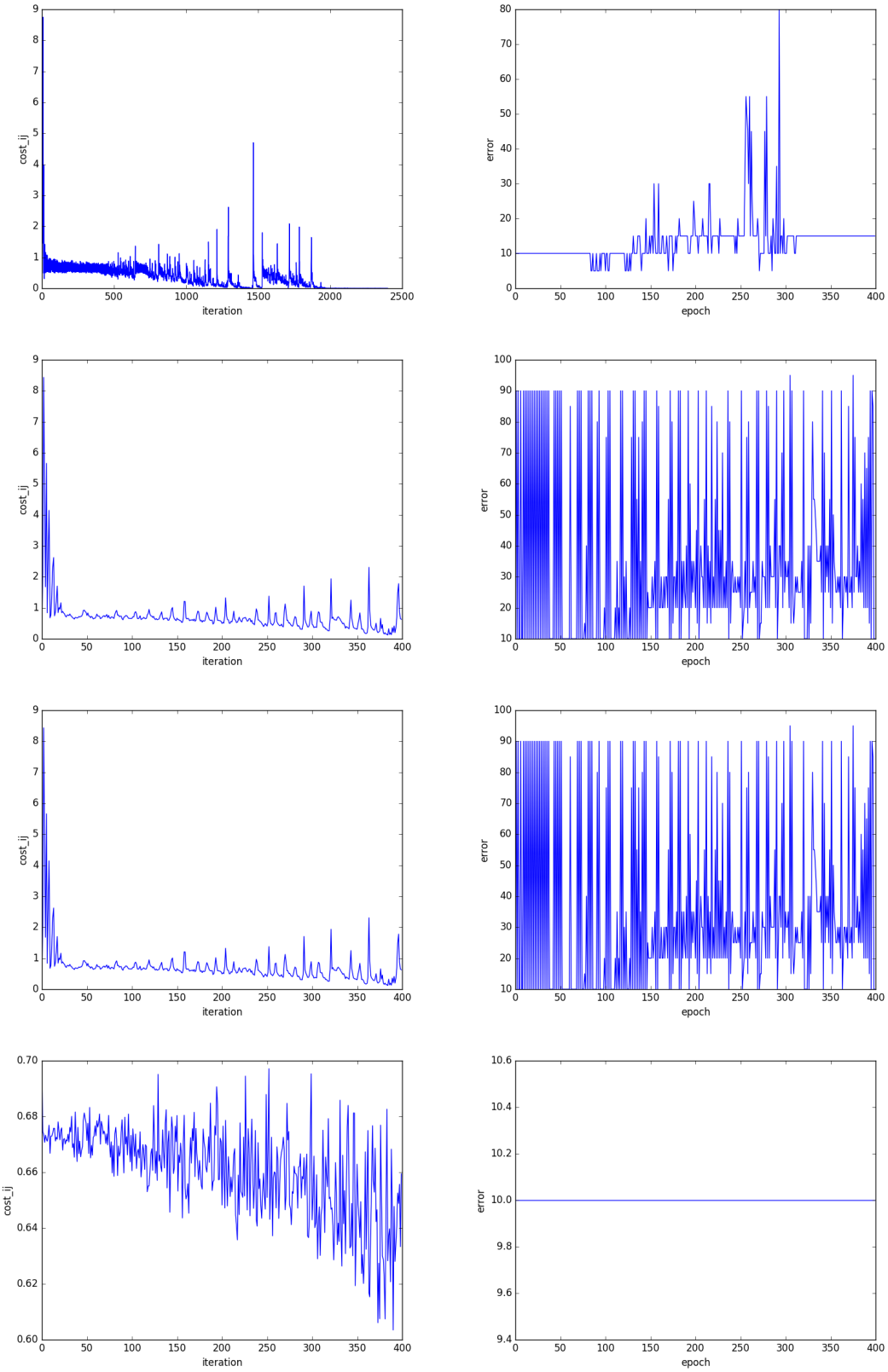


Figure 2.36: cost and error of the tree experiments with CASIA images.

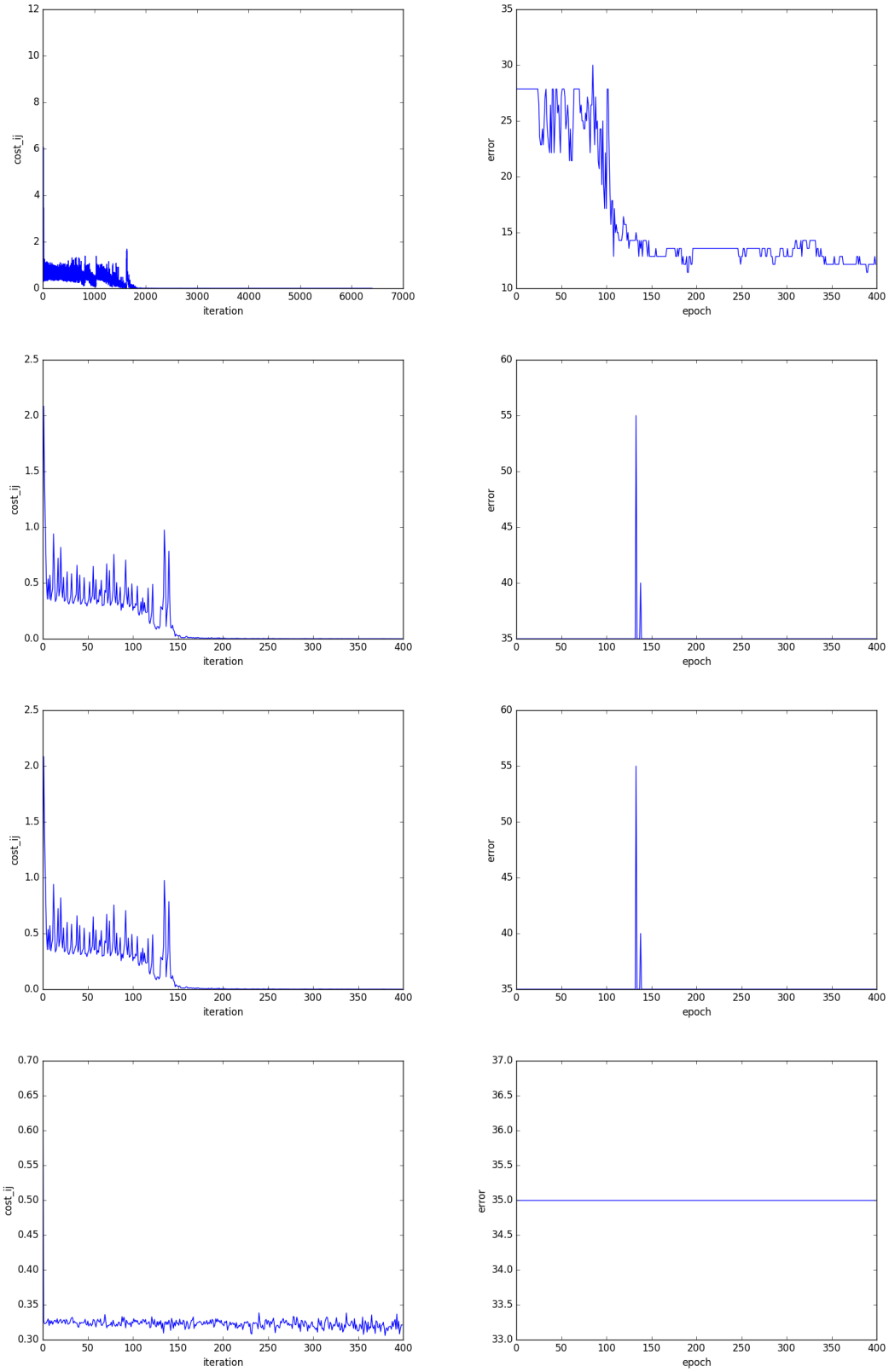


Figure 2.37: cost and error of the tree experiments with CASIA videos.

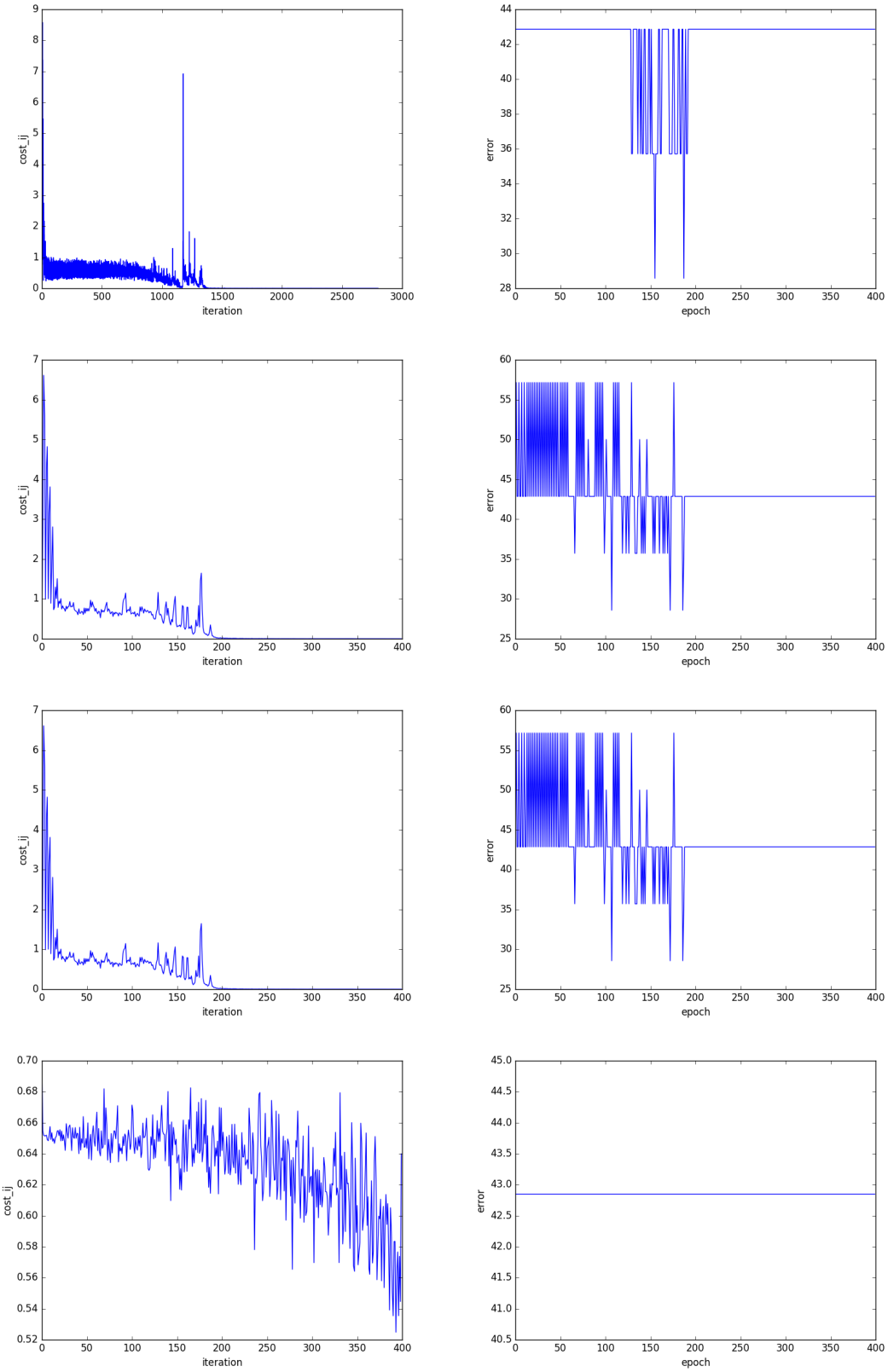


Figure 2.38: cost and error of the tree experiments with MFSD images.

Bibliography

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

[2] "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.

[3] 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.

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

[5] 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.

[6] 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.

[7] D. Stutz, "Understanding convolutional neural networks," 2014.

[8] 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.

[9] 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.

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

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

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

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

[14] 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.

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

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

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

[18] 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.

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

[20] 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.

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

[22] 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.

[23] 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.

List of Figures

List of Tables