Master's Degree in Automatic Control and Robotics

Segmentation, Labeling and Optical Character Recognition Applied on Receipt Images

Author: Claudi Ruiz Camps
Director: PhD. Cecilio Angulo Bahón
Call: April 2016

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Declaration of Authorship

I, Claudi Ruiz Camps, declare that this thesis titled, 'Segmentation, Labeling and Optical Character Recognition Applied on Receipt Images' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Progress is made by trial and failure; the failures are generally a hundred times more numerous than the successes ; yet they are usually left unchronicled."

William Ramsay

UNIVERSITAT POLITECNICA DE CATALUNYA

Abstract

ETSEIB

Automatic Control Department

Master's degree in Automatic Control and Robotics

Not always science and companies share the same objectives, however a company's need can be sometimes satisfied by applying science. Two of the common problems of a company that tries to work with data in the field of artificial intelligence are firstly how to get the data itself and secondly how to label it. This thesis presents a real case where a company has to acquire data, label it and then create the best model that fits the data. After many experiments this thesis shows how a small number of training inputs and an Arachnid model, that combines 45 specialized CNNs and a classifier that finds the pattern behind the output of those CNNs, can improve the test accuracy of LeNet-5 from 93.85% to 99,00% when classifying 10 different classes of optical characters with a concrete dataset. 1.000 single characters were extracted randomly from around 10.000 images of receipts and 800 of them were used for the training and 200 for the test. The approach of this thesis is focused on a real and concrete problem of a company, trying to find the best solution by using science and at the same time taking into account the company's need.

Acknowledgements

I would like to thank my girlfriend Miriam, my three really good friends Fran, Iker and Juan, and my brother Aram for being an inspiration to me. Furthermore, I would like to thank my parents Manolo and Marisa and my sister Ona for their endless support. I would also like to thank Max Welling and Tijmen Blankevoort for the opportunity to do my thesis at their company Scyfer B.V. Lastly, thanks to my supervisor Cecilio Angulo for being always there and giving me feedback.

Contents

List of Figures

List of Tables

Dedicated to the ones that have been always there ...

Chapter 1

Introduction

A company always needs to solve specific problems by searching the best solution in terms of time and/or quality. This thesis tries to simulate one of those real problems that a company in the field of the artificial intelligence has to face very often, to obtain a proper dataset to work with and then reach a concrete objective by using this dataset and different machine learning techniques. The key is to combine Science and a concrete need. In this case the need would be how to classify as best as possible single characters from receipt pictures, however there are some handicaps that will make difficult to satisfy properly this need. This research will show how the current state of the art in the classification part of the offline Optical Character Recognition (offline OCR) can be improved for a specific dataset by using Arachnid models. We know that it is difficult to parallelize the objectives of the science and the needs of the real world because companies want to solve their needs while maybe Science is more interested in other directions.

LeNet-5 is one of the best single character classifiers and it works really well under some conditions like small translations and rotations. This thesis will not improve this aspect but the accuracy of the LeNet-5 in this specific dataset. Based on my knowledge, this research proposes something new in the field of the offline OCR. This thesis mixes different concepts in order to obtain the best of all of them and then have the maximum accuracy when classifying 10 different classes within a small and challenging dataset.

The first part of the research will focus on the obtaining of the dataset, that means image preprocessing, character segmentation and labeling. The raw data of this project was proportionate by Scyfer B.V. and it consists in 10.000 images of receipts. The second part will consist on using different techniques of machine learning to classify those 10 classes and then discover the best technique for this concrete dataset. The new concept of this thesis is called Arachnid models and it is the combination of different classifiers, each one specialized in only two classes, that will give their opinion about a new sample

and then all of those opinions will hide a pattern that will be discovered by another classifier. We will see how to synthesize those opinions in a way that the classifier can better handle them, and that will be by reducing the dimensionality and then trying to find the best projection of the opinions. In other words, the information coming from the input images will pass through three lines of information compression. The first line will be composed of 45 specialized CNNs that will take care to give their opinion about the input images, the second line will be a dimensionality reduction that will take care of the compression of those opinions by searching the best projection. Finally the third line will be a classifier that will learn about those opinions and then will give the last word about the class of the input image.

Chapter 2

Image Preprocessing for the segmentation

The dataset is composed of 10.000 images of receipts in a really bad conditions like: different orientations, perspectives, brigness, ... and this makes really difficult to segment isolated characters from them. Some image preprocessings can improve very much the segmentation task. In this chapter we will only focus on this by showing the output of each image manipulation in order to find the best way that works as much as possible with all the images.

Different image preprocessings were applied in order to make easier the segmentation process and Tesseract was the software used to segment and label all the isolated characters inside the images of the receipts. Tesseract needs some conditions to give the best results: binarization, removed noise and good orientation [1]. In this thesis the binarization and noise were solved but the orientation, due to the random way that the pics were taken and that makes really hard or impossible to orientate all the character lines. However the goal of this thesis is to get data and then work with this data with different classifiers and techniques, so we will just try to obtain the maximum number of isolated characters to later randomly take a mini-batch. Notice that all the decisions made in the preprocessing techniques were based on the output of Tesseract, sometimes it is hard to see how those image prerpocessings made any change on the original image by just looking to their output but Tesseract is very sensitive to all those small changes and it can give very different results when segmenting and classifying.

In the figure 2.1 it is shown some examples of receipt images that have been used for this thesis. It is easy to see that the receipts were blurred, discolored, wrinkled, misaligned with respect to the pic,... in general the images have poor quality. In the next sections

FIGURE 2.1: Some examples of the receipt images.

we will see different manipulations by checking the inputs and outputs of each one and quantifying the results.

2.1 Removing the noise

The first image preprocessing was to apply a median filter in order to remove the noise but in a really smooth way because the ink of many of the characters was too weak. The method was taking the median of all the pixels under a kernel area and the central element were replaced by this median value [2] (the homogeneous kernel's size was 5x5 pixels (see figure 2.2).

The median value is just the one in the middle of all the sorted pixels that the kernel takes. The template size slider defines how much filtering takes place. Median filtering is used to remove "salt and pepper" noise but it doesn't remove gaussian noise. Other type of filters were also applied, and combinations between them, like the average filter,

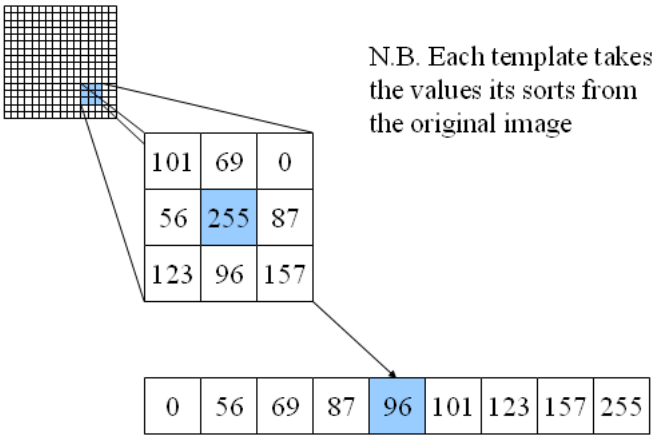

FIGURE 2.2: 5x5 median filter.

Gaussian filter and bilateral filter, but after applying those filters and then segmenting with Tesseract for some images it worked really well but not in general. So the final solution was only to apply the median filter in a very smooth way (by applying a small kernel in comparison with the size of the images) because it worked well for the whole dataset. Tesseract has troubles segmenting images with "holes" because it segments them like they were characters and that's why the median filter is necessary. Gaussian filter could also work well but due to very different conditions in all the images it doesn't become necessary.

FIGURE 2.3: On the left the original image and on the right the output after applying the median filter.

In the figure 2.3 we can see an example of the original image of a receipt on the left and on the right the same image after applying the median filter, notice that the output is in gray-scale because it took less computational time applying the filter to a grey-scale image than to a color image with three channels.

2.2 Adaptive Threshold

The second image preprocessing was to apply an adaptive threshold. This adaptive threshold value is the weighted sum of neighbourhood values where weights are a gaussian window [3]. In the beginning a fixed threshold was applied but it was not the best solution because all the images were very different between each other and with distinct lighting conditions in all areas, and also because of the weak ink of many characters. So the best results were obtained by applying and adaptive threshold 2.4.

2.3 Close Morphology

The last preprocessing was to apply a close morphology in order to remove those random black dots [4] that could mislead Tesseract for the segmentation, but again in a really smooth way.

$$AclosingB = (AdilationB)erosionB \tag{2.1}$$

Equation 2.1 shows how closing the image A (in our case A is a 2x2 matrix of ones) with B is the same than dilating A with B and then eroding the result with B again. In the dilation process the output pixel is the maximum value of all the pixels in the input pixel's neighborhood, and the erosion has the same definition but the output is the minimum instead of the maximum [5]. So for closing first were applied the dilation and then the erosion.

In the figure 2.4 we can see an example of the output after applying an adaptive threshold (on the left) and then closing (on the right). The closing part is important not only for the segmentation task, it is really useful for the labeling because Tesseract also classifies isolated characters after being segmented. In the next chapter we will see how Tesseract helped in the labeling task and then how important is the closing.

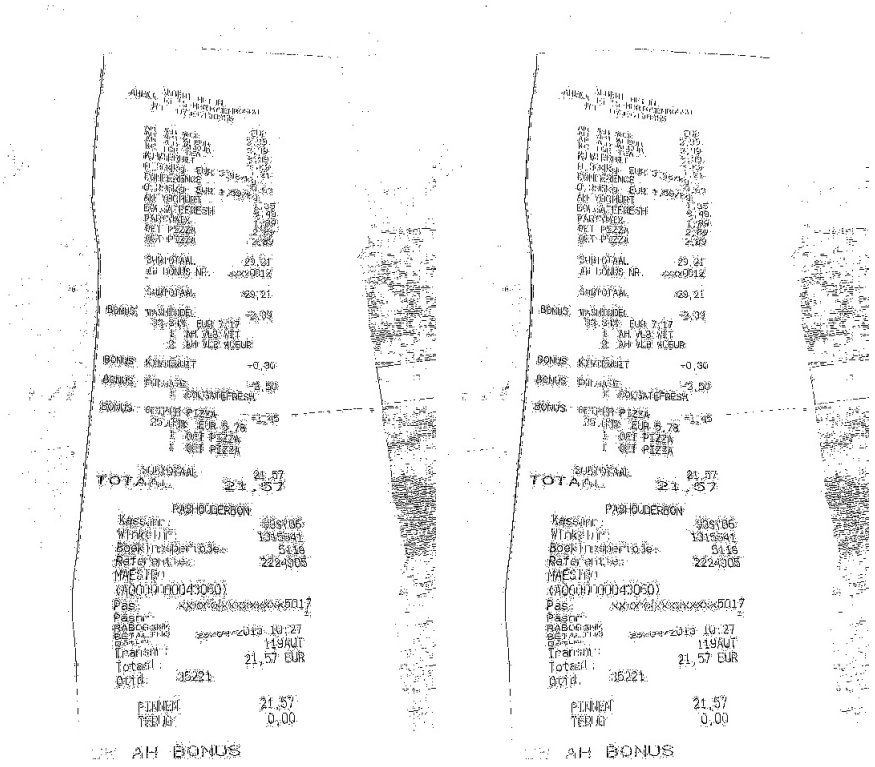

FIGURE 2.4: On the left the output of the adaptative thresholding and on the right after closing.

Chapter 3

Character Segmentation and Labeling

In this project Tesseract (is an open source OCR developed at HP Labs between 1985 and 1995 and now at Google) was used for the segmentation and labeling. For the segmentation Tesseract gives the box boundaries of each character of all the images in a text file, and them were cropped from the original images using those coordinates. This software does various image preprocessing operations internally before doing the actual OCR [1] however sometimes those are not enough to guarantee good reasults and that's why some image preprocessings were applied to the original images before using the Google software. Tesseract doesn't work perfectly and sometimes is mistaken taking boundaries from the background of the images. Once the characters are segmented tesseract is applied to classify all of them (in a single character mode) under the supervision of a human in order to have a correct labeled training and test dataset. At the end, 8.000 segmented images were obtained and then 1.000 images were randomly selected to train and test different classification models, concretely 800 were used for training and 200 for the test. Because of the random selection of training and test isolated characters we can't ensure the same quantity of images per each class so this means the algorithms that we will use for the classification will have to deal with this handicap.

FIGURE 3.1: Example of Tesseract working well.

In the figures 3.1 and 3.2 we have examples of how Tesseract works. The first row of isolated characters shows and example of each class on the dataset and the second row

FIGURE 3.2: Example of Tesseract not working well.

contains some other examples on how Tesseract doesn't work well due to bad image prerpocessing, bag image conditions or simply because of the Tesseract limitations.

Chapter 4

Convolutional Neural Networks for Isolated Character Recognition

One of the main objectives of this project is to classify the test dataset with 10 different classes as best as possible and in this case by using a Convolutional Neural Network (CNN). Most of the theory explained in this chapter were extracted from the famous paper of Yann Lecun in his paper [6]. For a CNN each input image comprises a set of pixel intensity values, and the desired output is a posterior probability distribution over the ten classes. The CNN is able to classify the digits under small translations, elastic deformations, scaling and rotations. One simple approach would be to treat the image as the input to a fully connected network, such a network could in principle yield a good solution to this problem and would learn the appropriate invariances by example. However, this approach ignores a key property of images, which is that nearby pixels are more strongly correlated than more distant pixels. A CNN exploits this property by extracting local features that depend only on small subregions of the image. Information from such features can then be merged in later stages of processing in order to detect higher-order features and ultimately to yield information about the image as a whole. Also, local features that are useful in one region of the image are likely to be useful in other regions of the image, for instance if the object of interest is translated or rotated.

A CNN is composed by different kind of layers: the input, convolutional and subsampling layers. The convolutional layer has different feature maps and each take inputs only from a small subregion of the image, and all of the units in a feature map are constrained to share the same weight values. All of these units in a feature map detect the same pattern but at different locations in the input image, and they have their own parameters and

biases that lead to extract efficient features. If the input image is shifted, the activations of the feature map will be shifted by the same amount but will otherwise be unchanged, so this makes a CNN invariant to small translations, elastic deformations, scaling and rotations of the input image. The outputs of the convolutional units form the inputs to the subsampling layer of the network. For each feature map in the convolutional layer, there is a plane of units in the subsampling layer and each unit takes inputs from a small receptive field in the corresponding feature map of the convolutional layer. The receptive fields are chosen to be contiguous and nonoverlapping and this implies that there are half the number of rows and columns in the subsampling layer compared with the convolutional layer. So the response of a unit in the subsampling layer will be slightly invariant to small shifts of the image in the corresponding regions of the input space. The whole CNN is composed by multiple pairs of convolutional and subsampling layers and as deeper the information goes across all the layers there is a larger degree of invariance to input transformations because of the abstraction of the information, and also the loss of information is compensated by the increment of feature maps. The final layer of the network would typically be a fully connected layer like a typical Neural Network. The whole network can be trained by error minimization using backpropagation to evaluate the gradient of the error function or other techniques.

4.1 Convolutional Operator

A convolutional layer is the result of applying the convolutional operator (see figure 4.1 and 4.2) with a defined kernel to all the neighborhoods of the previous layer.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = (1*i)+(2*h)+(3*g)+(4*f)+(5*e)+(6*d)+(7*c)+(8*b)+(9*a)$$

FIGURE 4.1: Convolutional operator.

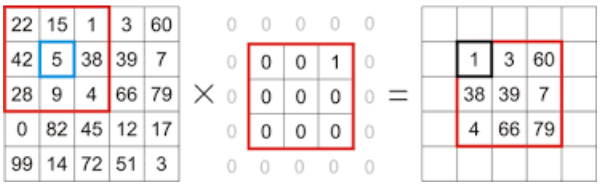

FIGURE 4.2: Example of the convolutional operator.

4.2 Single Image Preprocessing

A CNN doesn't need many image preprocessings due to its capacity to extract features from the image, however it is preferable to do at least three image preprocessings: to add a frame of four pixels of thickness to ensure that all the pixels that define the character have a neighborhood to be studied [6], mean extraction to every pixel [7] and with standard deviation equal to one. Then the output is an image with a 0-mean, 1 standard deviation and size of 32x32 (the original has 28x28).

4.3 Different Architectures for the CNN

4.3.1 LeNet-5

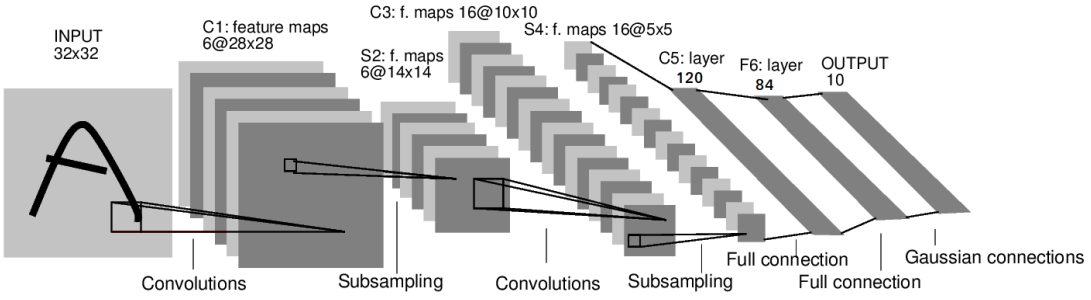

FIGURE 4.3: LeNet-5 architecture.

LeNet-5 was a Convolutional Nerual Network proposed by Lecun et al. in 1998 [6]. This LeNet-5 has 7 layers and all of them, except for the input layer, have trainable parameters. The input is a preprocessed image with a size of 32x32 pixels.

The first convolutional layer (C1) is the output of applying 6 different 5x5 kernels to the input image, so at the end this layer has 6 feature maps. Concretely each kernel interacts with each pixel and its neighborhood of the input image obtaining as an output a single value that belongs to the specific feature map. So C1 is a convolutional layer with 6 28x28 feature maps and 156 trainable parameters (kernel's size by number of kernels plus the number of biases $= (5 * 5) * 6 + 6)$ and 122.304 connections (kernel's size plus the bias and all of this multiplied by the feature map's size and the number of feature maps $= (5 * 5 + 1) * 28 * 28 * 6)$.

The second sub-sampling layer (S2) is the output after applying a 2x2 filter to each neighborhood of C1. S2 has 6 feature maps of size 14x14. The operation is to add the four inputs, multiply it by a trainable parameter and add the result to a trainable bias. Then S2 has 12 trainable parameters (number of filters by number of trainable parameters per

filter $= 6 * 2$) and 5880 connections (filter's size plus the bias and all of this multiplied by the feature map's size and the number of feature maps $= (2 * 2 + 1) * 14 * 14 * 6$).

The fourth layer is again a convolutional layer (C3), this layer takes as an input each neighborhood of S2 and applies a fixed and asymmetric combination of kernels to obtain 12 feature maps. According to [6] the generalisation is better if there's asymmetry in layers connections due to S2-C5 matrix.

1	0	0	0	1	1	1	0	0	1	1	1	1	0	1	1
1	1	0	0	0	1	1	1	0	0	1	1	1	1	0	1
1	1	1	0	0	0	1	1	1	0	0	1	0	1	1	1
0	1	1	1	0	0	1	1	1	1	0	0	1	0	1	1
0	0	1	1	1	0	0	1	1	1	1	0	1	1	0	1
0	0	0	1	1	1	0	0	1	1	1	1	0	1	1	1

FIGURE 4.4: Fixed and asymmetric S2-C5 matrix.

Figure 4.4 shows the fixed and asymmetric matrix combination between S2's maps (6 rows) and C3's maps (16 columns), for example the first C3's map is obtained by applying a different kernel to each 3 first S3's maps. This combination can also be done randomly and figure 4.5 shows an example.

1	0	1	1	0	0	0	1	0	1	1	1	1	0	1	0
1	1	1	1	1	0	0	1	1	0	1	1	1	1	0	0
0	1	0	1	0	1	1	0	1	1	0	1	0	1	0	1
0	0	1	1	1	1	0	0	1	1	1	0	0	0	1	1
1	0	1	1	0	1	1	1	0	1	1	0	1	1	1	0
1	1	0	0	1	0	0	1	1	1	1	1	1	1	0	1

FIGURE 4.5: Random S2-C5 matrix.

C3 has 1.516 trainable parameters (kernel's size by number of kernels plus the number of biases $= (5 * 5) * 60 + 16$) and 151.600 connections (kernel's size by number of kernels plus number of biases $= (5 * 5 * 60 + 16) * 10 * 10$).

The fifth layer (S4) it's a sub-sampling of C3 and it works in the same way than S2 but with 16 feature maps of size 5x5 as the output, it has 32 trainable parameters ($= 16 * 2$) and 2.000 connections ($= (2 * 2 + 1) * 5 * 5 * 16$).

The last convolution layer is C5 and it is full connected with S4 with kernels of size 5x5, that means C5 has 120 feature maps of size 1x1, 48.120 trainable parameters connections ($= 5 * 5 * 120 * 16 + 120$).

Finally, the last layer is F6 and it is fully connected with C5 with 84 neurons 10.164 trainable parameters ($= 120 * 84 + 84$).

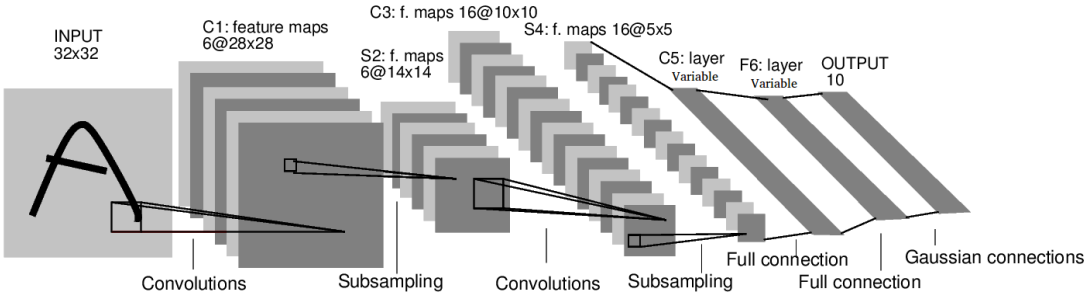

FIGURE 4.6: Optimized LeNet-5 architecture.

4.3.2 Optimized LeNet-5

The optimized LeNet-5 keeps the LeNet-5 architecture but with the best number of feature maps in the convolutional layer C5 and neurons in the full connected layer F6 for our specific dataset. The best number of feature maps in C5 should be around 120 (what Yann LeCun suggested in his paper [6]), for the case of the number of neurons in F6 the range is between 70% and 90% of the input layer [8]. Again using in one side a fixed feature map matrix between S2 and C3 and a random one.

4.4 Backpropagation

The backpropagation algorithm was originally introduced in the 1970s, but its importance wasn't fully appreciated until a famous 1986 paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams. That paper describes several neural networks where backpropagation works far faster than earlier approaches to learning, making it possible to use neural networks to solve problems which had previously been insoluble. Today, the backpropagation algorithm is the workhorse of learning in neural networks.

The main idea of the backpropagation algorithm is to compute the error given by a loss function that basically computes the difference between the target and the output of the Convolutional Neural Network, and then propagates this error through all the layers from the output to the first layer (the input layer doesn't have trainable parameters). This algorithm allows to update each parameter with an incremental value that reduces the difference between the output and the target at each learning iteration.

The Levenberg-Marquardt algorithm, which is an approximation to the Newtons method is said to be more efficient in comparison to other methods for convergence of the Backpropagation algorithm for training a moderate-sized feedforward neural network [9].

Starting with the mathematics shown in [6], at each learning iteration a particular parameter w_k is updated according to the following stochastic update rule:

$$w_k \leftarrow w_k - \epsilon_k \frac{\partial E^p}{\partial w_k} \tag{4.1}$$

where E^p is the instantaneous loss function for pattern p:

$$E^p = \frac{1}{2}(prediction^p - target^p)(prediction^p - target^p)^T \tag{4.2}$$

Notice that the sign in the equation 4.1 is negative because the gradient always points in the direction of the greatest rate of increase of the function but the backpropagation looks for just the opposite, it tries to find the minimum of the loss function. In Convolutional Neural Networks, due to the weight sharing, the partial derivative in the equation 4.1 is the sum of the partial derivatives with respect to the connections that share the same parameter w_k:

$$\frac{\partial E^p}{\partial w_k} = \sum_{(i,j)\in V_k} \frac{\partial E^p}{\partial u_{ij}} \tag{4.3}$$

where u_{ij} is the connection weight from unit j to unit i, V_k is the set of unit index pairs (i,j) such that the connection between i and j share the parameter w_k, i.e.:

$$u_{ij} = w_k \quad \forall (i,j) \in V_k \tag{4.4}$$

The step sizes ϵ_k are a function of the second derivative of the loss function along the axis w_k:

$$\epsilon_k = \frac{\eta}{\mu + h_{kk}} \tag{4.5}$$

where μ is a hand-picked constant and h_{kk} is an estimate of the second derivative of the loss function E with respect to w_k. The larger h_{kk}, the smaller the weight update. The parameter μ prevents the step size from becoming too large when the second derivative is small. The term h_{kk} is expressed as:

$$h_{kk} = \sum_{(i,j)\in V_k} \sum_{(k,l)\in V_k} \frac{\partial^2 E}{\partial u_{ij} \partial u_{kl}} \tag{4.6}$$

However three approximations are used in order to calculate h_{kk}:

1. The first approximation is to drop the off-diagonal terms of the Hessian with respect to the connection weights:

$$h_{kk} = \sum_{(i,j)\in V_k} \frac{\partial^2 E}{\partial u_{ij}^2} \tag{4.7}$$

the terms $\frac{\partial^2 E}{\partial u_{ij}^2}$ are the average over the training set of the local second derivatives:

$$\frac{\partial^2 E}{\partial u_{ij}^2} = \frac{1}{P} \sum_{p=1}^{P} \frac{\partial^2 E^p}{\partial u_{ij}^2} \tag{4.8}$$

Those local second derivatives with respect to connection weights can be computed from local second derivatives with respect to the total input of the downstream unit:

$$\frac{\partial^2 E^p}{\partial u_{ij}^2} = \frac{\partial^2 E^p}{\partial a_i^2} x_j^2 \tag{4.9}$$

where x_j is the state of unit j and $\frac{\partial^2 E^p}{\partial a_i^2}$ is the second derivative of the instantaneous loss function with respect to the total input to unit i (denoted a_i).

2. The second approximation is the Gaussian-Newton approximation and it ignores the non-linearity of the activation function *f()*, but not that of the loss function.

$$\frac{\partial^2 E^p}{\partial a_i^2} = f^{'}(a_i)^2 \sum_k u_{ki}^2 \frac{\partial^2 E^p}{\partial a_k^2} \tag{4.10}$$

The right-hand side is a sum of products of non-negative terms, therefore the left-hand side term is non-negative.

3. The third approximation is that the average in equation 4.8 is note run over the entire training set, but it on a small subset randomly selected of the training set instead.

Chapter 5

Arachnid Models

This chapter proposes a kind of ensemble learning in order to improve the test accuracy of the optimized CNN. The definition of ensemble learning is the following:

"Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the (classification, prediction, function approximation, etc.) performance of a model, or reduce the likelihood of an unfortunate selection of a poor one" [10].

The main idea is to define 45 CNNs each one specilized in one concrete task, more concretely each CNN will just be trained with two specific classes. For example the first CNN will be only focused in class 0 and 1, the second CNN in class 0 and 2, the third in class 0 and 3 and so on until we have the 45 possible combinations to cover the 10 classes. The architecture of those 45 CNNs will be the best one in the experiments, the one that best classifies the test dataset.

Once all the 45 CNNs are trained we will use the output samples of all of them, when using as an input all the training images per CNN, to train another classifier like a Neural Network or k-NN. For the test part it would be just the forward of all the test input samples through all the CNNs and the classifier. To see more in detail the training and test part please check 5.4 and 5.5 respectively. In the experimental part we will see how this technique improves radically the test accuracy.

As a real example, imagine that a manager has three kind of employees: a biologist, a physicist and a mathematician. The manager doesn't have the specialized knowledge of each employee but knows when one of them is right or wrong just based on the experience (previous cases). So when the manager has a new problem to solve he just ask by making exactly the same question to each employee about how this employee would solve or face

that new problem. Once the manager has all the answers he/she will make a decision based on the past experiences. For example the manager knows that the mathematician knows a lot about maths but not that much about biology or physics, so the employee will give an answer based on his/her knowledge. When the topic is about mathematics this employee will give the best answer in comparison with the other two and will feel really self-confident when answering and the manager will notice that. And maybe the mathematician will also be pretty confident by answering about a topic in physics but less confident with a topic in biology. So there is a pattern behind the answer of all the employees that the manager can detect after analysing many training examples.

In the following sections we will define the different parts of the Arachnid models: Legs, Body and head. Each part has a different task and all the information goes from the input to the output going through the legs, body and head. As a summary: each leg is a specialized CNN trying to be the best at its task, the body takes care of reducing the dimensionality of the output of all the specialized CNNs to then make easier the task of the classifier, which is the head.

5.1 Legs (45 CNNs)

As it was explained above, each leg of the Arachnid model is a CNN specialized in two classes (the first CNN would be specialized in class 0 and 1, the second in class 0 and 2,...). The architecture of all the CNNs will be the best one in the experiments. Each CNN once is trained will give its opinion about the class of the training/test input so it will have 10 available outputs even being just specialized in only two classes. Those outputs will go to the classifier, first filtered by the body (the dimensionality reduction), and this one will study the pattern behind all those outputs.

5.2 Body (Dimensionality Reduction)

The Body part of the Arachnid model takes care of the dimensionality reduction. The two techniques used in this thesis are the Principal Components Analysis and the Linear Discriminant Analysis. Both techniques look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of data. PCA on the other hand does not take into account any difference in class, and factor analysis builds the feature combinations based on differences rather than similarities. Discriminant analysis is also different from factor analysis in that it is not an interdependence technique: a distinction between independent variables and

dependent variables (also called criterion variables) must be made. In the experiments we will see how both techniques work with our dataset. It is not easy to decide a priory which one of the two algorithms is better for our case because we have labeled data so it would make sense only to use the LDA to reduce the dimensionality but that's not entirely true. The reason is the distribution of the data, LDA assumes a Gaussian distribution for all the classes and even more, they share the same covariance matrix and this may not be the case because of two reasons: maybeb we don't have enough data or the data just have other distributions per class than the Gaussian distribution, and even they are Gaussians maybe they don't share the same covariance matrix.

5.2.1 Principal Components Analysis

Principal component analysis (PCA) is a technique that is widely used for applications such as dimensionality reduction, lossy data compression, feature extraction, and data visualization. PCA can be defined as the orthogonal projection of the data onto a lower dimensional linear space, known as the principal subspace, such that the variance of the projected data is maximized [11]. In this project a PCA algorithm was used in order to reduce the dimensionality of the features and then also reduce the computational time cost due to the hardware limitations. Now we follow the mathematical explanation of a PCA and then understand it in a deeper way.

Given a data matrix with p variables and n samples, the data are first centered on the means of each variable. This will insure that the cloud of data is centered on the origin of our principal components, but does not affect the spatial relationships of the data nor the variances along our variables. The first principal components (Y_1) is given by the linear combination of the variables $X_1, X_2, ..., X_p$

$$Y_1 = a_{11}X_1 + a_{12}X_2 + ... + a_{1p}X_p \tag{5.1}$$

or in a compact way:

$$Y_1 = a_1^T X \tag{5.2}$$

The first principal component is calculated such that it accounts for the greatest possible variance in the data set. Of course, one could make the variance of Y1 as large as possible by choosing large values for the weights a11, a12, ... a1p. To avoid this, weights are calculated with the constraint that their sum of squares is 1.

$$a_{11}^2 + a_{12}^2 + ... + a_{1p}^2 = 1 \tag{5.3}$$

The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.

$$Y_2 = a_{21}X_1 + a_{22}X_2 + ... + a_{2p}X_p \tag{5.4}$$

This continues until a total of p principal components have been calculated, equal to the original number of variables. At this point, the sum of the variances of all of the principal components will be equal to the sum of the variances of all of the variables, that is, all of the original information has been explained or accounted for. Collectively, all of these transformations of the original variables to the principal components is:

$$Y = AX \tag{5.5}$$

The rows of matrix A are called the eigenvectors of matrix Sx, the variancecovariance matrix of the original data. The elements of an eigenvector are the weights a_{ij}, and are also known as loadings. The elements in the diagonal of matrix Sy, the variancecovariance matrix of the principal components, are known as the eigenvalues. Eigenvalues are the variance explained by each principal component, and to repeat, are constrained to decrease monotonically from the first principal component to the last.

To summarize, principal component analysis involves evaluating the mean and the covariance matrix of the data set and then finding the eigenvectors of the covariance matrix corresponding to the largest eigenvalues.

5.2.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a classical statistical algorithm for supervised dimensionality reduction and classification and it has been used widely in many applications involving high-dimensional data. LDA computes an optimal projection by minimizing the within-class distance and maximizing the between-class distance at the same time, thus achieving maximum class discrimination under the condition that the probability distribution of each class is a Gaussian and they share the same covariance matrix. The optimal transformation in LDA can be readily computed by applying an eigendecomposition on the so-called scatter matrices.

Now we can express the idea of the LDA explained above but in a mathematical way. The LDA algorithm aims to find a linear transformation $W \in \mathbb{R}^{d \times m}$ that maps x_i in the d-dimensional space to m-dimensional space, in which the between class scatter is maximized while the within-class scatter is minimized:

$$argmaxtr_W((W^T S_w W)^{-1}(W^T S_b W)) \tag{5.6}$$

where S_b and S_w are the between-class scatter matrix and within-class scatter respectively, which are defined as:

$$S_b = \sum_{k=1}^{c} n_k(\mu_k - \mu)(\mu_k - \mu)^T \tag{5.7}$$

$$S_w = \sum_{k=1}^{c} \sum_{i \in C_k} n_k(x_i - \mu_k)(x_i - \mu_k)^T \tag{5.8}$$

where C_k is the index set of the k-th class, μ_k and n_k are mean vector and size of k-th class respectively in the input data space, $\sum_{k=1}^{c} n_k \mu_k$ is the overall mean vector of the original data. We can express equation 5.6 in an equivalent way:

$$argmaxtr_W((W^T S_t W)^{-1}(W^T S_b W)) \tag{5.9}$$

where $S_t = \sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T$ and $S_t = S_w + S_b$. When the total scatter matrix S_t is non-singular, the solution of the equation 5.9 consists of the top eigenvectors of the matrix $S_t^{-1} S_b$ corresponding to nonzero values. When the total class scatter matrix S_t does not have full rank, the solution of 5.9 consists of the top eigenvectors of the matrix $S_t^{\dagger} S_b$ corresponding to nonzero eigenvalues, where $S^{\dagger}$ denotes the pseudo-inverse of S_t.

5.3 Head (Classifiers)

The head is basically the classifier, the algorithm that looks for the pattern behind all the output samples of the CNNs. The k-NN and a Neural Network are the two classifiers that will take care of this task, the first one is faster and effective and the second one more complex but also effective.

5.3.1 k-Nearest Neighbors

K Nearest Neighbor (k-NN) is a very simple algorithm and at the same time it works really well. This algorithm does not make any assumptions on the underlying data distribution, and this is useful because in the real world most of the practical data does not obey the typical theoretical assumptions made. k-NN does not use the training data points to do any generalization so it keeps all the training data for the testing phase leading to expensive computation if the data set is large (makes decision based on the entire training data set or in a subset of them). Despite its simplicity, k-NN has been successful in a large number of classification problems, including handwritten digits, satellite image scenes,... It is often successful where each class has many possible prototypes, and the decision boundary is very irregular.

k-NN assumes that the data is in a feature space. The data can be scalars or possibly even multidimensional vectors. Since the points are in feature space, they have a notion of distance so different distance metrics are used, the most common is the Euclidean distance but in this project we will also use: city block distance, minkowski, Chebychev istance... see subsection Distance Metrics.

Each of the training data consists of a set of vectors and class label associated with each vector and the number of classes can be arbitrary. The number k says how many neighbors (where neighbors is defined based on the distance metric) influence the classification.

k-NN can be used either for density estimation or classification but in this project this algorithm is used just for classification. So part of the data is given for the training and the rest for testing. Depending on the value assigned to K the algorithm has different behavior, meaning k controls the degree of smoothing, so that small k produces many small regions of each class, whereas large k leads to fewer larger regions. An interesting property of the 1-NN is that in the limit when the number of training samples goes to infinite, the error rate is never more than twice the minimum achievable error of an optimal classifier [11]. To understand this we assume that the query point coincides with one of the training points, so that the bias is zero. This is true asymptotically if the dimension of the feature space is fixed and the training data fills up the space in a dense fashion. Then the error of the Bayes rules is just the vairance of a Bernouilli random variate (the target at the query point), while the error of 1-NN rule is twice the variance of a Bernouilli random variate, one contribution each for the training and query targets [12].

5.3.1.1 Nearest Neighbor Rule (k = 1)

This is the simplest scenario and it works reasoning not well only when the number of data points is not very large. As it was already defined, if x_1 is the point to be test then 1-NN finds the closest point to x_1 that is x_2. Now nearest neighbor rule asks to assign the label of x_2 to x_1. If the number of data points is very large there is a very high chance that the label of the two points are the same.

5.3.1.2 Nearest Neighbor Rule (k = K)

In this case the algorithm basically tries to find the k nearest neighbors and do a majority voting. Lets say k = 5 and there are 3 instances of class *1* and 2 instances of class *2*. In this case, k-NN says that new point has to be labeled as class 1 as it forms the majority and the same argument when there are multiple classes.

One of the straight forward extension is not to give 1 vote to all the neighbors. A very common thing to do is weighted k-NN where each point has a weight which is typically calculated using its distance. For example under inverse distance weighting, each point has a weight equal to the inverse of its distance to the point to be classified. This means that neighboring points have a higher vote than the farther points.

5.3.1.3 Distance Metrics

In this subsection different distance metrics are defined in order to be tested in the experimental k-NN and then find the best one. We define a and b as points of interest and n as their dimensionality.

- The **City block distance** is:

$$\sum_{i=1}^{n} |a_i - b_i| \tag{5.10}$$

- The **Chebychev distance** is:

$$max_i(|a_i - b_i|) \tag{5.11}$$

- The **Correlation distance** is:

$$1 - \frac{(a - \hat{a})(b - \hat{b})^T}{\sqrt{\sum_{i=1}^{n}(a - \hat{a})(a - \hat{a})^T}\sqrt{\sum_{i=1}^{n}(b - \hat{b})(b - \hat{b})^T}} \tag{5.12}$$

- The **Cosine distance** is:

$$1 - \frac{ab^T}{\sqrt{(aa^T)(bb^T)}} \tag{5.13}$$

- The **Euclidean distance** is:

$$\sqrt{(a-b)(a-b)^T} \tag{5.14}$$

5.3.2 Neural Networks

The artificial neural network technique (NN) was originally designed for pattern recognition purposes (Turing, 1948; Farley and Clark, 1954; Rosenblatt, 1958). Nevertheless it can be used as a regression as well. On one side it is the big advantage of an artificial neural network that it does not ask for physical pre-information to create the possibility of modelling. Furthermore it projects the systems function into a weight matrix without any physical relevance. But this black box character is on the other side a crucial problem: if the complexity of the system increases, a NN needs much more projection capacity and becomes complex itself.

5.3.2.1 Perceptron

Before going deep on what a Neural Network is able to do we will see a type of artificial neuron called a perceptron. Perceptrons were developed in the 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts. Today, it's more common to use other models of artificial neurons that have sigmoid neurons but first it is better to understand how a perceptron works.

A perceptron takes several binary inputs, $x_1, x_2, ...$, and produces a single binary output:

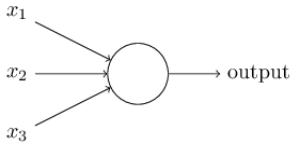

FIGURE 5.1: A perceptron with its three inputs and one output

In the figure 5.1 it is shown an example of a perceptron with three inputs and one output. Rosenblatt proposed a simple rule to compute the output by introducing the concept of weights $(w_1, w_2, ...)$. They are real numbers expressing the importance of the respective inputs to the output. Imagine that the perceptron has two outputs, 0 or 1,

determined by whether the weighted sum $\sum_j w_j x_j$ is less than or greater than some defined threshold, or bias (with opposite sign), value (see equation 5.15). Just like the weights, the bias is a real number which is a parameter of the neuron that measures how it is to get the perceptron output to 1. In other words, the bias is a measure of how easy it is to get the perceptron to fire. For a perceptron with a really big bias, it's extremely easy for the perceptron to output a 1. But if the bias is very negative, then it's difficult for the perceptron to output a 1.

$$output = \begin{cases} 0 & \text{if} \sum_j w_j x_j + b \leq 0 \\ 1 & \text{if} \sum_j w_j x_j + b > 0 \end{cases} \tag{5.15}$$

A way to understand how the perceptron works is that it is a device that makes decisions by weighing up evidence, and by varying the weights and the threshold it can give different models of decision-making.

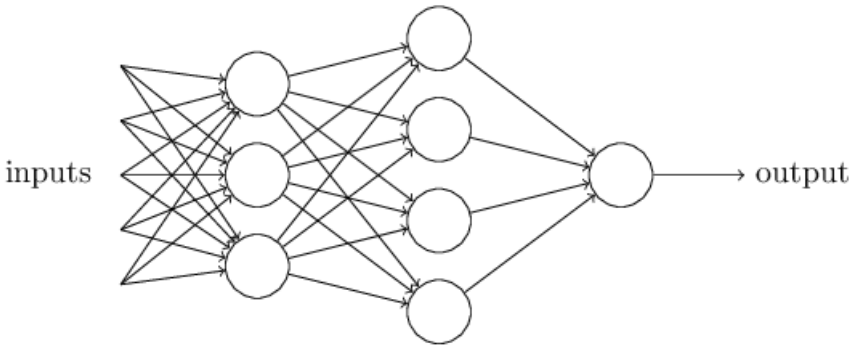

FIGURE 5.2: Network with multiple layers and perceptrons

In the figure 5.2 is shown a network with three layers of perceptrons. The first one is making three very simple decisions, by weigthing the input evidence. In the second layer each of those perceptrons is making a decision by weigthing up the results from the first layer of decision-making. In this way a perceptron in the second layer can make a decision at a more complex and more abstract level than perceptrons in the first layer. And even more complex decisions can be made by the perceptron in the third layer. In this way, a many-layer network of perceptrons can engage in sophisticated decision making.

5.3.2.2 Sigmoid Neurons

The sigmoid neuron can be described in the same way than the perceptron, so it also has some inputs $x_1, x_2, ...$ but instead of beeing binary inputs they can be values between

0 and 1 for instance. Also just like a perceptron, the sigmoid neuron has weights for each input, $w_1, w_2, ...$, and an overall bias, b. And like the inputs, the outputs can be values between 0 and 1 coming from the sigmoid activation, see equation 5.16 where z is $\sum_j w_j x_j + b$.

$$\sigma = \frac{1}{1 + e^{-z}} \tag{5.16}$$

In order to see some improvements in the output of a network of perceptrons or sigmoid neurons we need to do some small changes at each weight and then measure the consequences in the output. These small changes applied in a network of perceptrons could change completely the output, for instance switching from 1 to 0. That is why sigmoid neurons are more interesting, because they can give outputs between 0 and 1 giving us the possibility to apply small changes in the weights to improve a bit the output.

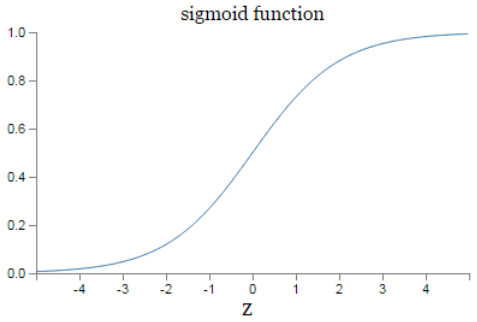

FIGURE 5.3: Sigmoid function used by a sigmoid neuron

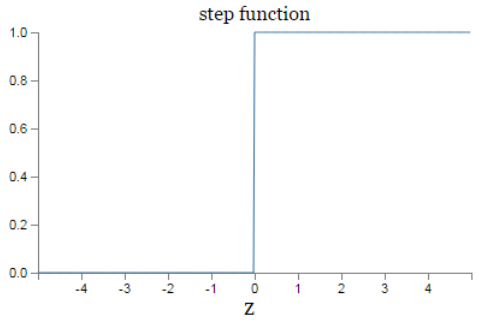

FIGURE 5.4: Step function used by a perceptron

As we said before, one big difference between perceptrons and sigmoid neurons is that sigmoid neurons don't just output 0 or 1. They can have as output any real number between 0 and 1, so this could mean a certain probability that the right output is 1 or 0 and just defining a threshold as a boundary between the two output possibilities we can impose which class the input belongs to. There are other activation functions for neurons but in this project we will only use the sigmoid one.

5.3.2.3 Backpropagation

During the training all the weights and biases of the Neural Network are updated by using a training function in order to minimize the loss function. In this thesis all the experiments were done by using the Stochastic Diagonal Levenberg-Marquardt already explained in section 4.4.

5.3.2.4 The Architecture of Neural Networks

As mentioned earlier, the leftmost layer in this network is called the input layer, and the neurons within the layer are called input neurons. The rightmost or output layer contains the output neurons, or, as in this case, a single output neuron. The middle layers are called a hidden layer, since the neurons in these layers are neither inputs nor outputs.

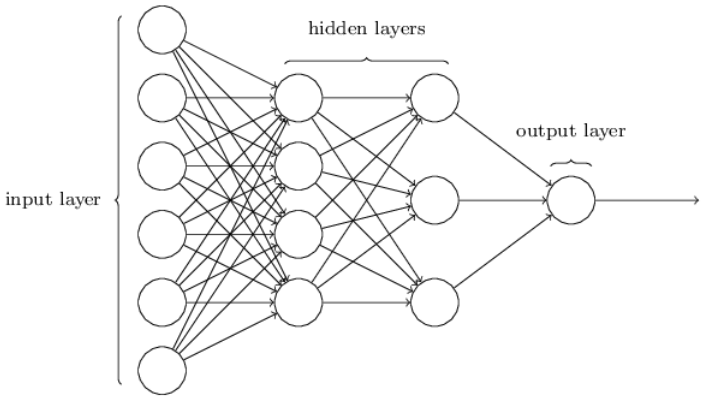

FIGURE 5.5: Example of a Neural Network structure with one input layer, two hidden layers and one output layer

5.3.2.5 Neural Networks as Black Boxes

The basic problem of every NN is its black box character, that prevents any analyses of the reliability and the plausibility of modelled output. Accordingly it happens, that every new computational approach ends up with a different new weight matrix. Surprisingly, the results of all the approaches can be nearly or totally the same. This phenomenon is caused by the start matrix, that has to contain small random values. This is a typical effect forced by what is called an underdetermined network. That means that the model capacity of the chosen network architecture is too high in comparison with the number of patterns. It is necessary to understand, what is going on inside an underdetermined network. One and the same numerical input-output relation can be expressed by separated groups of neurons. But the optimization of the network

during the training phase forces a fast approach by a training of all neurons. While separated groups of neurons would have been able to project the input-output relations, this projection is spread over all neurons now. If we sum up the weights of the homologues neurons and deletes the redundant ones afterwards, the modelled result would be exactly the same. The new reduced NN would be much smaller instead.

5.4 Training part

The training part of an Arachnid model is composed of 3 steps: training the specialized CNNs, obtaining all the output samples of the CNNs and finally training the classifier with those output samples.

More specifically, the steps to train the Arachnid model are the following ones:

1. Training the CNNs: each CNN is specialized in two different classes, i.e., is trained with a subset of the training dataset that contains only these two classes. Although we know that each CNN is only specialized in two classes they are trained imposing they have 10 outputs because this will allow, later at the next step, to give their opinion about a class they are not specialized in (see figure 5.6).

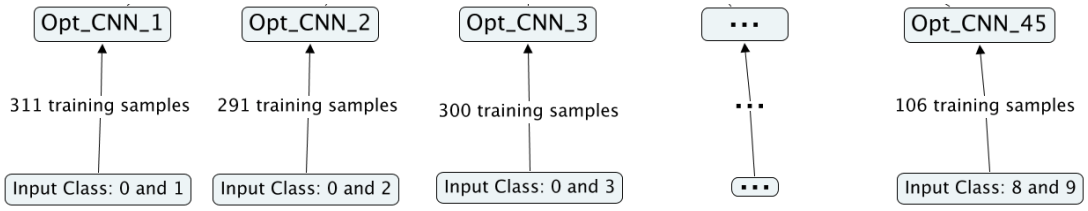

FIGURE 5.6: Training the CNNs: each CNN is specialized in two different classes.

2. Obtaining all the output samples of the CNNs: Once all the CNNs are trained we just forward all the training samples through all the CNNs. For instance the CNN that is only specialized in class 0 and 1 will also give its opinion about all the other classes and the same with the rest of CNNs. At the end we will have 10 outputs per CNN and per sample, this means 450 outputs that will go to the body in order to be reduced and then to the classifier (see figure 5.7).

3. Training the classifier: In all of these 450 outputs per sample it will be a combination of opinions between the 9 specialized CNNs and all the rest 36 CNNs. The task of the classifier is to find a pattern behind all the opinions per sample and relate this with its target to then in the test part be able to predict the target with new opinions (see figure 5.8).

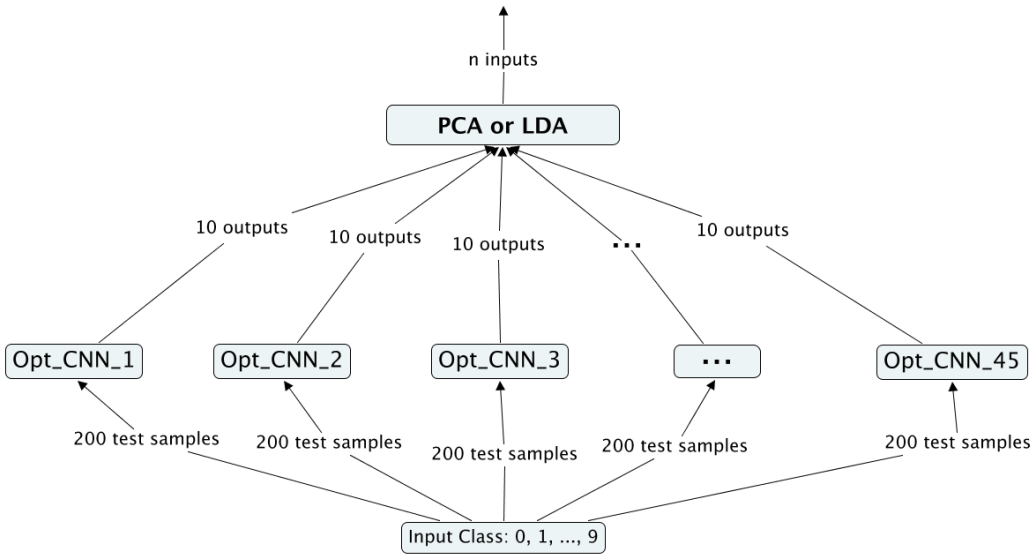

FIGURE 5.7: Obtaining all the output samples of the CNNs: doing the forward of all the training samples and reducing the dimensionality to n

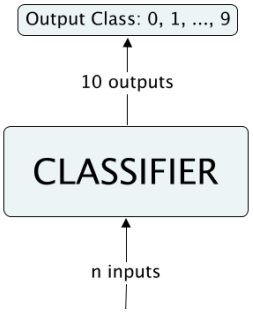

FIGURE 5.8: Training the classifier with all the output samples of the CNNs reduced to n dimensions

5.5 Test part

The test part is composed of only one step, simply by doing the forward of all the test input samples through all the legs, body and head of the Arachnid model (see figure 5.9).

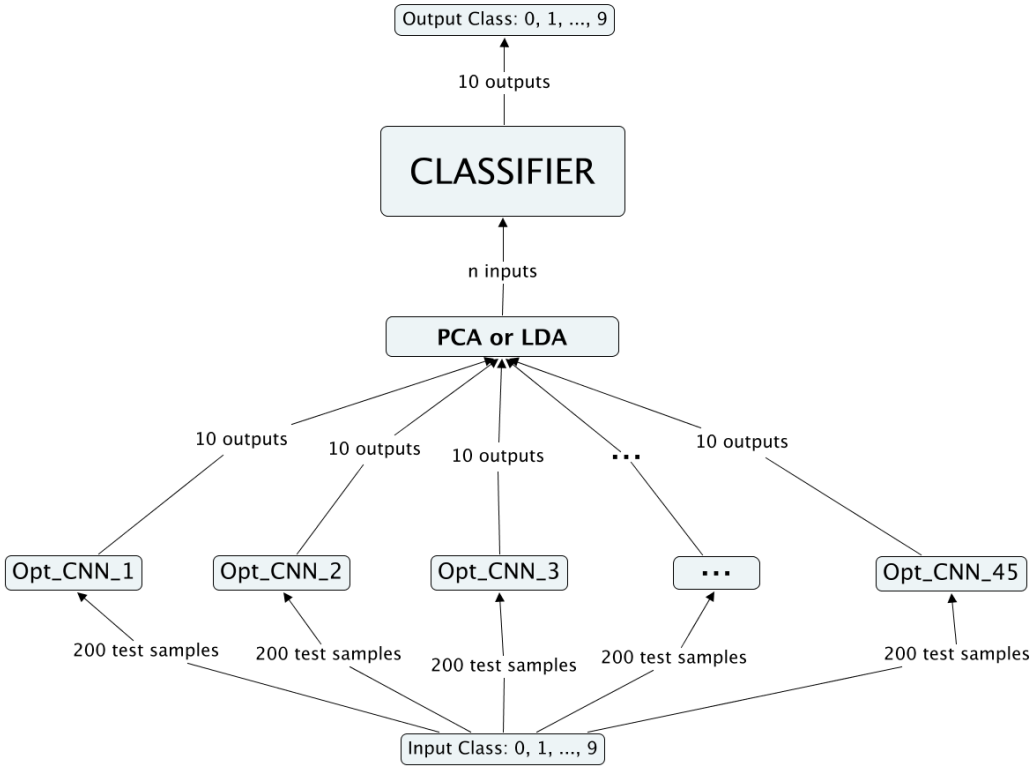

FIGURE 5.9: Data flow when testing the Arachnid Models

Chapter 6

Experiments and Results

In this section all the results are showed in order to find the model that best classifies the test dataset. Remember that the training dataset is composed of 800 images and the test dataset of 200 and the number of samples of each class is shown in the table 6.1. The code used to train and test all the CNNs was obtained from the file exchange of Matlab and modified to be adapted to the need of this research. The k-NN, NN, PCA and LDA were used from the Matlab toolbox of machine learning. In the first experiment a LeNet-5 with a fixed and asymmetric S2-C5 matrix was trained and tested, in the second experiment the same architecture was trained but with a random S2-C5 matrix. In the third experiment, an Optimized LeNet-5 with a fixed and asymmetric S2-C5 matrix was also trained and tested in order to find the best number of feature maps in the last convolutional layer and the best number of neurons in the full connected layer. The fourth experiment was the same than the previous one but again with a random S2-C5 matrix. Finally we will see how the Arachnid models can improve the results considerably. The Arachnid models will be the 45 CNNs, different combinations between the two algorithms to reduce the dimensionality (PCA and LDA) and between the two classifiers (k-NN and a Neural Network) and their different parameters. Notice that all the experiments related with Neural Networks (Convolutional Neural Networks as well) have been repeated 10 times in order to minimize as much as possible the random effect of the weight initialization. All the experiments were performed by using a laptop Lenovo Y50 with a 4th Gen Intel Core i7-4720HQ Processor (2.60GHz 1600MHz 6MB and 8 cores).

	Total Number	Training	Test
Zero	256	216	40
One	134	95	39
Two	102	75	27
Three	97	84	13
Four	68	57	11
Five	100	74	26
Six	71	52	19
Seven	48	41	7
Eight	55	48	7
Nine	69	58	11
Total	1000	800	200

TABLE 6.1: Number of samples of each class.

40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.3	35.2	0.2	0.8	0.2	0.4	0.0	0.8	0.0	0.1
0.0	0.1	26.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.3	0.2	11.8	0.0	0.7	0.0	0.0	0.0	0.0
0.0	0.1	0.0	0.0	10.9	0.0	0.0	0.0	0.0	0.0
0.6	0.1	0.0	0.0	0.0	25.3	0.0	0.0	0.0	0.0
0.8	0.0	0.0	0.0	0.1	3.8	13.1	0.0	1.1	0.1
0.0	0.4	0.0	0.1	0.1	0.0	0.0	6.0	0.0	0.4
0.0	0.0	0.0	0.6	0.0	0.0	0.0	0.0	6.4	0.0
0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	10.1

TABLE 6.2: Test confusion matrix of LeNet-5

6.1 LeNet-5

As it was explained above, the first experiment consists on training LeNet-5 with a fixed and asymmetric S2-C5 matrix and then the same with a random S2-C5 matrix for the second experiment.

6.1.1 LeNet-5 with a fixed and asymmetric S2-C5 matrix

After 1259.7 ± 0.1 seconds training the architecture proposed by Yann LeCun on his paper [6] (LeNet-5) it reached 92.85 ± 0.01 % of success on the test.

In the table 6.2 we can see the confusion matrix for the test of the LeNet-5 with a fixed and asymmetric S2-C5 matrix, and it shows how this architecture had some troubles to classify perfectly all the numbers instead of the *0*. The worst classified class was the *6* with a classification success of 68.9 ± 0.1 % (see table 6.3).

	Total Number	Test Classification Success
Zero	40	100,0%
One	39	90.3%
Two	27	99.6%
Three	13	90.8%
Four	11	99.1%
Five	26	97.3%
Six	19	68.9%
Seven	7	85.7%
Eight	7	91.4%
Nine	11	98.1%

TABLE 6.3: Test classification success for each class of the LeNet-5 with a fixed and asymmetric S2-C5 matrix.

40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	35.0	0.2	1.1	0.2	0.6	0.0	0.9	0.0	0.0
0.0	0.0	27.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.4	0.0	12.1	0.0	0.5	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	11.0	0.0	0.0	0.0	0.0	0.0
0.2	0.0	0.0	0.0	0.0	25.8	0.0	0.0	0.0	0.0
0.7	0.0	0.0	0.0	0.0	4.1	13.3	0.0	0.9	0.0
0.0	0.3	0.0	0.0	0.0	0.0	0.0	6.2	0.0	0.5
0.0	0.0	0.0	0.7	0.0	0.4	0.0	0.0	5.9	0.0
0.2	0.0	0.3	0.0	0.1	0.0	0.1	0.0	0.0	10.3

TABLE 6.4: Test confusion matrix of LeNet-5 with a random S2-C5 matrix.

6.1.2 LeNet-5 with a random S2-C5 matrix

After 1230.0 ± 0.1 seconds training the architecture proposed by Yann LeCun, but this time with a random S2-C5 matrix, it reached 93.30 ± 0.01 % of success on the test.

In the table 6.4 we can see the confusion matrix for the test of the LeNet-5 with a random S2-C5 matrix, and it shows how this architecture had no troubles to classify perfectly the classes *0*, *2* and *4*. The worst classified class was the *6* again, with a classification success of 70.0 ± 0.1 % (see table 6.5).

6.2 Optimized LeNet-5

6.2.1 Optimized LeNet-5 with a fixed and asymmetric S2-C5 matrix

Taking into account a fixed and asymmetric S2-C5 matrix, the best number of neurons in the full connected layer was 89 and the best number of feature maps in the last

	Total Number	Test Classification Success (± 0.1 %)
Zero	40	100.0%
One	39	89.7%
Two	27	100.0%
Three	13	93.1%
Four	11	100.0%
Five	26	99.2%
Six	19	70.0%
Seven	7	88.6%
Eight	7	84.3%
Nine	11	93.6%

TABLE 6.5: Test classification success for each class of the LeNet-5 with a random S2-C5 matrix.

39.8	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0
1.0	36.0	0.0	0.5	0.2	0.5	0.0	0.7	0.0	0.1
0.0	0.0	27.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.5	0.0	12.0	0.0	0.5	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	11.0	0.0	0.0	0.0	0.0	0.0
0.6	0.0	0.0	0.0	0.0	25.4	0.0	0.0	0.0	0.0
0.4	0.0	0.0	0.0	0.3	3.3	13.7	0.0	1.3	0.0
0.0	0.5	0.0	0.0	0.0	0.0	0.0	6.0	0.0	0.5
0.0	0.0	0.0	0.6	0.0	0.2	0.0	0.0	6.2	0.0
0.3	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	10.6

TABLE 6.6: Test confusion matrix of the Optimized LeNet-5 with a fixed and asymmetric S2-C5 matrix.

convolutional layer (C5) was 106. This architecture gave a 93,85 ± 0.01 % of test accuracy after 1132.2 ± 0.1 seconds training.

In the table 6.6 we can see the confusion matrix for the test of the Optimized LeNet-5 with a fixed and asymmetric S2-C5 matrix, and it shows how this architecture had no troubles to classify perfectly the numbers *2* and *4*. The worst classified class was the *6* with a classification success of 72.1 ± 0.1 % (see table 6.7).

6.2.2 Optimized LeNet-5 with a random S2-C5 matrix

Using the same architecture than before but just applying a random S2-C5 matrix we had a test accuracy of 93,20 ± 0.01 % and 1134.5 ± 0.1 seconds training.

In the table 6.8 we can see the confusion matrix for the test of the Optimized LeNet-5 with a random S2-C5 matrix, and it shows how it can only classify the number *2* perfectly. The worst classified class was the *6* with a classification success of 69.0 ± 0.1 % (see table 6.9).

	Total Number	Test Classification Success
Zero	40	99.5%
One	39	92.3%
Two	27	100.0%
Three	13	92.3%
Four	11	100.0%
Five	26	97.7%
Six	19	72.1%
Seven	7	85.7%
Eight	7	88.6%
Nine	11	96.4%

TABLE 6.7: Test classification success for each class of the Optimized LeNet-5 with a fixed and asymmetric S2-C5 matrix.

39.7	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0
1.1	35.3	0.1	0.9	0.1	0.6	0.0	0.8	0.0	0.1
0.0	0.0	27.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.2	0.0	12.5	0.0	0.3	0.0	0.0	0.0	0.0
0.0	0.2	0.0	0.0	10.8	0.0	0.0	0.0	0.0	0.0
0.4	0.0	0.0	0.0	0.0	25.6	0.0	0.0	0.0	0.0
0.5	0.0	0.0	0.0	0.1	4.2	13.1	0.0	1.0	0.1
0.0	0.3	0.0	0.1	0.0	0.0	0.0	6.0	0.0	0.6
0.0	0.0	0.0	0.8	0.0	0.1	0.1	0.0	6.0	0.0
0.3	0.0	0.2	0.0	0.0	0.0	0.0	0.1	0.0	10.4

TABLE 6.8: Test confusion matrix of the Optimized LeNet-5 with a random S2-C5 matrix.

6.2.3 Summary of the results

The four proposed architectures work really well with the given dataset despite the few samples of each class to train, the different quantity of samples per class and the quality of the training and test dataset (due to the low quality of the images of the receipts). The best architecture is the one with 106 feature maps in the last convolutional layer (C5), 89 neurons in the full connected layer and a fixed S2-C5 matrix. However, like the rest of the proposed architectures, the best one also had troubles when classifying well the number *6*. In the next section we will see how the Arachnid models can solve this issue easily.

6.3 Arachnid Models

In this part of the experiments we will use different Arachnid models in order to find the best combination of classifiers and dimensionality reduction techniques to improve

	Total Number	Test Classification Success
Zero	40	99.3%
One	39	90.5%
Two	27	100.0%
Three	13	96.2%
Four	11	98.2%
Five	26	98.5%
Six	19	69.0%
Seven	7	85.7%
Eight	7	85.7%
Nine	11	94.5%

TABLE 6.9: Test classification success for each class of the Optimized LeNet-5 with a random S2-C5 matrix.

the test accuracy. The main idea is to train different CNNs with different subsets of the training dataset, to train another classifier with the training output samples of all the CNNs and then test the whole model with the test dataset. More specifically, each CNN would be specialized in a different pair of classes, for instance the CNN_1 would be specialized in classes *0* and *1*, CNN_2 in classes *0* and *2* and so on (45 CNNs in total). They would be only focused on their own problem, simply to classify between two classes and then another classifier would take as a training input all the training outputs samples of the 45 CNNs. Until now we were talking about the training, so for the test all the CNNs will take all the test images, even if they are not specialized on their class (for instance CNN_1 is specialized in class *0* and *1* but it will take as a test input also images from the class *2,3*, ..., *9*). This gives the chance to evaluate each test image with all the CNNs, then to get their opinions of what class it belongs to and the classifier would find a pattern behind all the opinions in order to classify that test image even better than the specialized CNN for that kind of class. All the information goes from the input to the output by obtaining some features and at the same time losing information, so the classifier has more abstract information to work with than all the CNNs but this is compensated with less information to analyse. However two dimensionality reduction techniques were used in order to make easier the task of the classifier, those techniques are the Principal Components Analysis (PCA) and the Linear Discriminant Analysis (LDA). The PCA is focused on finding the maximum separability of the data without taking into account the label and the LDA it does the same but using the additional information like the labels, and then tries to maximize the separation between classes. The main problem of our dataset is the quantity of samples, it has only 800 training samples and 200 for the test, so that means we can't ensure a Gaussian distribution for all the classes at each feature dimension and even less that all the distributions have the same covariance. That's why the PCA algorithm has something to do because it doesn't

consider the kind of distribution of the data at all but the LDA does, so we will see if imposing Gaussian distributions for all the dimensions can be compensated by taking into account the labels.

6.3.1 k-NN

The k-NN classifier is a really fast algorithm that doesn't need a previous training, it just takes the test samples and classifies them simply by considering the class of the k closest points of the training dataset. This algorithm allows to play with different parameters like: the number of neighborhoods, the distance weight and the distance. During the experiments different results were obtained depending on the selected parameters, for more details see table 6.12.

40	0	0	0	0	0	0	0	0	0
0	38	0	1	0	0	0	0	0	0
0	0	27	0	0	0	0	0	0	0
0	0	0	13	0	0	0	0	0	0
0	0	0	0	11	0	0	0	0	0
0	0	0	0	0	26	0	0	0	0
0	0	0	0	0	0	19	0	0	0
0	0	0	0	0	0	0	7	0	0
0	0	0	0	0	0	0	0	7	0
0	1	0	0	0	0	0	0	0	10

TABLE 6.10: Test confusion matrix of the Arachnid model with a 2-NN as a classifier with an equal distance weight, using the correlation distance and the reduced dimensionality of 60 given by the Linear Discriminant Analysis.

The best combination for the k-NN was using $k = 2$, an equal distance weight and the correlation distance, everything under a dimensionality reduction of 60 by using the LDA algorithm. The fact that LDA gives better results than the PCA when using $k - NN$ gives us some idea about the data distribution, we could think that it fits really well with the assumptions of the LDA that say that the probability distribution of each class are assumed to be a Gaussian and they share the same covariance matrix. It is interesting to realize that 60 was the less reduced dimensionality found for each experiment and gave the best result, so projecting the data to those 60 dimensions would give the maximum explanation of the patterns to be discovered by $2 - NN$. However despite the PCA algorithm doesn't consider the label of each sample it gave really good improvements, concretelly 97.5 $\pm$ 0.1 %, using only 6 projected dimensions in the case of $1 - NN$, inverse distance weight and euclidean distance.

The best test accuracy of all the experiments was 99.0 $\pm$ 0.1 %. The table 6.10 shows the test confusion and we can see how this Arachnid model only misclassificies one sample

	Total Number	Test Classification Success
Zero	40	100.0%
One	39	97.4%
Two	27	100.0%
Three	13	100.0%
Four	11	100.0%
Five	26	100.0%
Six	19	100.0%
Seven	7	100.0%
Eight	7	100.0%
Nine	11	90.9%

TABLE 6.11: Test classification success for each class of the Arachnid model with a 2-NN as a classifier, an equal distance weight and using the correlation distance.

of class *1* and another one of class *3*, being confused by class 9 and class *1* respectively. The worst classified class in terms of percentage is class *9* with an accuracy of 90.9 ± 0.1 % (see table 6.11), we could say that the reason is the few training samples of this class but classes *7* and *8* have even more training samples.

The correlation distance measures the dependence between two samples and when the distance is zero means that the two samples are independent to each other. In our case the samples of the same cluster will have a big correlation distance and the opposite with the samples from different clusters. To compensate the fact that k-NN finds the closest k training samples to make a decision for the classification and the inverse meaning of correlation distance, we just define a new distance as $1 - correlationdistance$ (see equation 5.12).

6.3.2 Neural Network

The Arachnid model with a Neural Network also improved the test accuracy to 99.0 ± 0.1 % (see table 6.13) but with a training time of 33.1 ± 0.1 seconds. The training time shouldn't be an issue in most of the applications and concretely in our case 30 seconds are not a problem. In this case the NN was confused with a sample of class *1* and another one from class when *4* predicting class *3* and class *0* respectively (see the test confusion matrix 6.14).

6.3.3 Summary of the results

The NN also gives a performance of the test accuracy of 99.0 ± 0.1 % but with a training time of 33.1 ± 0.1 seconds. In this last experiments the algorithm to reduce

	Dim. Reduction		N. of Neighborhoods	Dist. Weight		Distance				
Test Accuracy	PCA	LDA	k	Equal	Inverse	CityBlock	Chebychev	Correlation	Cosine	Euclidean
97.5	19		1	X		X				
98.0	8		1	X			X			
97.5	12		5	X				X		
97.5	12		5	X					X	
98.0	6		4	X						X
97.5	7		3		X	X				
98.0	8		1		X		X			
98.5	12		4		X			X		
98.5	29		3		X				X	
97.5	6		1		X					X
98.0		51	2	X		X				
97.5		26	1	X			X			
99.0		60	2	X				X		
98.5		57	1	X					X	
98.5		44	2	X						X
97.5		48	1		X	X				
97.5		26	1		X		X			
98.0		35	9		X			X		
98.0		35	1		X				X	
98.0		20	7		X					X

TABLE 6.12: All the experiments related with k-NN as a classifier for the Arachnid model

	Dim. Reduction			Training Function	
Test Accuracy	PCA	LDA	Neurons	Levenberg Marquardt	Training Time
99.0	65		52	X	33.1 sec
97.5		34	27	X	5.8 sec

TABLE 6.13: All the experiments related with a Neural Network as a classifier for the Arachnid model

40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	38.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	27.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	13.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	26.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	19.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	7.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	7.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11.0

TABLE 6.14: Test confusion matrix of the Arachnid model with a Neural Network as a classifier with 52 neurons in the full connected layer and dimensionality reduction of 65 given by a Principal Component Analysis.

	Total Number	Test Classification Success
Zero	40	100.0%
One	39	97.4%
Two	27	100.0%
Three	13	100.0%
Four	11	90.9%
Five	26	100.0%
Six	19	100.0%
Seven	7	100.0%
Eight	7	100.0%
Nine	11	100.0%

TABLE 6.15: Test classification success for each class of the Arachnid model with a Neural Network as a classifier with 52 neurons in the full connected layer and dimensionality reduction of 65 given by a Principal Component Analysis.

the dimensionality that wins is the Principal Components Analysis.

6.4 The best model

After doing all the required experiments in order to find the best classifier we have two winners, both with 99.0 $\pm$ 0.1 % of test accuracy, and two misclassified images for each one:

1. The 2-NN with an equal distance weight, correlation distance and reduced dimensionality of 60 using the LDA algorithm. The two misclassified images were:

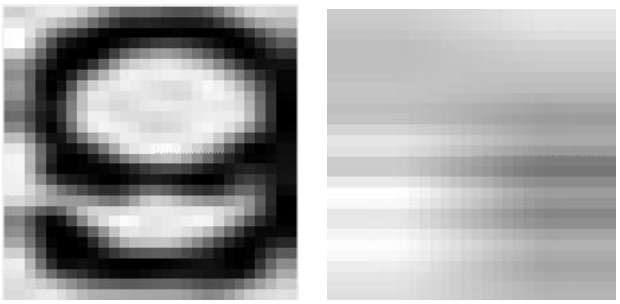

FIGURE 6.1: The two misclassified images of 2-NN-equal-correlation-LDA-60.

2. The Neural Network with 52 neurons in the full-connected layer, Levenberg-Marquardt training function and reduced dimensionality of 65 using the PCA algorithm. The two misclassified images were:

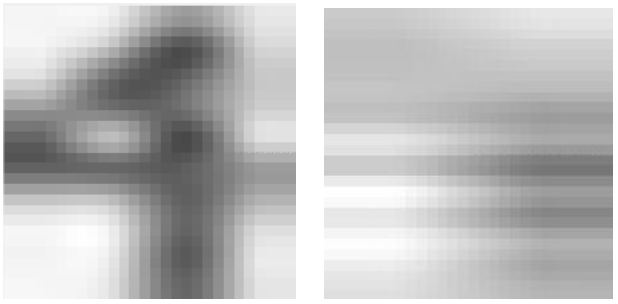

FIGURE 6.2: The two misclassified images of NN-52-LM-PCA-65.

Both models were mistaken when classifying the image of the right in figures 6.1 and 6.2, but that was not their fault. That sample shouldn't be in the dataset because it was not a number, it is just a consequence of a human error when selecting and labeling. However is curious how both models misclassify that test sample as a *3* (it was labeled as a *1*) and the reason is the three horizontal bars. The number *3* is the unique number with three horizontal bars as a feature, or at least this is one of its strongest features. So in the case that we want to say what number that sample is we could say that the nearest class in terms of features is the number *3*. 2-NN was mistaken when classifying the sample of the left in the figure 6.1 because one of the 2 nearest training samples to this one was mislabeled as a *1*, so again that was a human error when labeling. In the case of the NN the image of the left in the figure 6.2 was predicted as a *1* when the target was a *4* and the reason is the similarity of that sample with the class *1*. If we know for sure that the data is well labeled then we could consider to use the k-NN because it is really fast and accurate. However if we are not completely sure about the labeling but we have enough data to define a good model definite by the weights of the

neurons of a NN, then teh NN could guarantee better results than the k-NN because it would consider those mislabeled training samples as noise.

Chapter 7

Conclusions

The study was set up to do image preprocessing, segment, label and find the best way to classify the extracted single characters under some difficult conditions. The raw data was provided by Scyfer B.V. and it was composed of 10.000 images of receipts. Those images were in very bad conditions like: different orientations, perspectives, brightness,... and also the randomly selected dataset to train and test was composed of only 1.000 single characters belonging to 10 different classes (numbers from 0 to 9). It is well known that the Convolutional Neural Networks work really well classifying single characters and concretely LeNet-5 is one of the best. But, is there anyway to improve the test accuracy of the proposed LeNet-5 by Yann Lecun when applying an specific dataset? this was the aim objective of this thesis.

However, before searching the best algorithm for an specific dataset we need to obtain this dataset. Then, the first part of the study was to obtain 1.000 single characters from the raw dataset, 800 samples for the training and 200 for the test. Tesseract was used to segment and do the first labeling but it is a very sensitive software that it worked well only when some image preprocessings were applied to the raw data. The first image preprocessing was to apply different filters in order to remove the noise in the raw data, however it was not an easy task because all the images presented different light conditions as it was explained above. The best solution was just to apply a median filter smoothly. Tesseract has troubles when segmenting images with "holes" because it segments them like they were characters and that's why median filter was necessary. The second image preprocessing was an adaptive threshold that was able to adapt different thresholds to all the areas of each raw image. It was useful due to different lighting conditions between different raw images and in different areas in each of them, and also because of the weak ink of many characters. The last image preprocessing was the close morphology in order to remove the maximum black dots of the binary images after the adaptive threshold

because them mislead Tesseract during the segmentation. Finally the 10.000 images of the receipts were ready to apply Tesseract on them and 8.000 potential segmented characters were obtained but only 1.000 were selected randomly. Those 1.000 samples were numbers between 0 and 9 and the 80% of the dataset was used for the training and the rest 20% for the test. The last thing to do before having the final dataset was to label all the samples also using Tesseract but then supervised by a human because Tesseract didn't work perfectly. Despite of using the software and human supervision when the labelling one of the samples was not even a number between *0* and *9* and another one was mislabeled, that was discovered when the experiments.

Once the dataset was ready, different classification techniques were used in order to find the best test accuracy. The first technique was the LeNet-5 giving a 92.8 $\pm$ 0.01 % of test accuracy. This result was improved to 93.85 $\pm$ 0.01 % using an Optimized LeNet-5 composed of 106 feature maps in the last convolutional layer, 89 neurons in the full connected layer and a fixed and asymmetric S2-C5 matrix. The next improvement reached the 99.0 $\pm$ 0.01 % by using two different Arachnid models. One of the this best Arachnid models was composed of 45 specialized CNNs (the first one specialized only with zeros and ones, the second one with zeros and twos and so on until all the classes were covered), a LDA reducing the dimensionality of the output samples of the 45 CNNs from 450 to 60 and finally a 2-Nearest Neighbors (using an equal distance weight and a correlation distance) classifying all those output samples. The other best Arachnid model was also composed of 45 specialized CNNs, but with a PCA reducing the dimensionality of the output samples of the 45 CNNs from 450 to 52 and finally a NN with 52 neurons classifying all the output samples. Two human errors were detected when during the analysis of the results. The first one was a test sample that was not even a number between *0* and *9* but it presented three horizontal bars really similar to the characteristic features of the number *3* and then the two best models classified that sample as a *3*. The second human error was a mislabeled training sample where the real class was *9* but it was labeled as a *1*. This last error was the reason that made the Arachnid model with a 2-NN as a classifier fail because it took as a nearest neighbor that mislabeled sample when was classifying a test sample of class *9*. This same test sample was well classified by the Arachnid model with a NN as a classifier because it considered this mislabeled sample as a noise, and then giving more robustness when classifying. However, the Arachnid model with a NN as a classifier failed when classifying a test sample with target *4* because it confused it with a *1* due to its similarity with the class *1*. Those were human errors when the labeling in the firsts steps of the thesis, and showed how working with new datasets is not easy. This kind of problems are the ones that a company in the field of the Artificial Intelligence has to face very often.

The proposed Arachnid models show how ensemble learning can improve very much the classification success. The main idea is to specialize each CNN with a specific dataset containing only two classes. Then all of them give their opinion about all the training samples and, after a dimensionality reduction, a classifier can learn about the output of all of the 45 CNNs. There is a loss of information from the input sample to the final input of the last classifier but it is compensated with an abstraction of this information that helps the last classifier to be more accurate.

Despite that the proposed Arachnid models improved the test accuracy, in future research the accuracy could be enhanced by trying to find the best CNN's structure for each of the 45 CNNs. That would make them even more specialized because in this study the structure used for all the CNNs was the Optimized LeNet-5 mentioned before. Something that could be also studied is the best number of classes per CNN, maybe there is another better combination than 2 classes per CNN.

Bibliography

[1] Improving the quality of the output. . URL https://code.google.com/p/tesseract-ocr/wiki/ImproveQuality.

[2] Image blurring. . URL http://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html#gsc.tab=0.

[3] Adaptive thresholding. . URL http://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html#gsc.tab=0.

[4] Morphological transformations. . URL http://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html#gsc.tab=0.

[5] Dilation and erosion. . URL http://nl.mathworks.com/help/images/morphology-fundamentals-dilation-and-erosion.html.

[6] Yoshua Bengio Yann LeCun, Lon Bottou and Patrick Haffner. Gradient-based learning applied to document recognition. *IEEE*, 1998. URL http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf.

[7] I. Kanterand S. Solla Yann LeCun. Eigenvalues of covariance matrices: application to neural-network learning. *Physical review letters*, 66(18):2396–2399, 1991.

[8] Z. Boger and H. Guterman. Knowledge extraction from artificial neural network models. *IEEE systems, Man, and Cybernetics Conference*, 1997.

[9] M.T. Hagan and M. Menhaj. Training feed-forward networks with the marquardt algorithm. *IEEE Trans. Neural Networks*, 1994.

[10] Ensemble learning. . URL http://www.scholarpedia.org/article/Ensemble_learning.

[11] Christopher M. Bishop. Pattern recognition and machine learning. 2006.

[12] Robert Tibshirani Trevor Hastie and Jerome Friedman. The elements of statistical learning. 2009.