**DEEP LEARNING**

# Computer Vision:

## *Object recognition & detection*

**Authors:**
Serena Alderisi
Elisa Mateos
Yassir Al Bahri

**Course:**
MSc. in Data Science, UPM

April 3, 2020

# Contents

# List of Figures



# List of Tables

# Abstract

In this paper, an analysis of the German Traffic Sign Benchmark and CIFAR 100 datasets are performed with applying different methodologies in the spectrum of Computer Vision. Starting from the most basic architectures and gradually improving the algorithm. From Feed-Forward Neural Network to Convolutional Neural Network. Lastly, using the knowledge acquired, an object detection and recognition algorithm is build and evaluated. This work has been development in TensorFlow.

*keywords*: Computer Vision, Deep Learning, German Traffic Sign Benchmark, CIFAR 100 Dataset, Feed-Forward Neural Network, Convolutional Neural Network, traffic sign detection.

# Chapter 1

# Introduction

The objective of this project is to classify, recognize and detect objects for two different datasets. The first dataset is the German Traffic Signs Recognition Benchmark. It consist of 43 classes corresponding to street signs in Germany. The number of images in each class vary from 210 to 2250. The images are in a PPM format of varying sizes from 15x15 pixels to 250x350 pixels. The shape is not necessarily squared. The images include some space between boundaries and the actual sign so that edge based approaches can be used. The annotation corresponding to an image is provided as the class of the sign present in the image and relative coordinates of the bounding box enclosing the street sign.

The second dataset is the CIFAR 100 Dataset. This dataset has been collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. It has 100 classes and each image consists of 3 RGB colour channels and 32x32 pixel dimensions for an overall size per input of 3x32x32 = 3072.

# Chapter 2

# Feed-Forward Neural Network

Feed-forward networks consist of a number of simple neuron-like processing units, organized in layers. Every unit in a layer is connected with all the units in the previous layer. Data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs

## 2.1 German Traffic Sign Benchmark

### 2.1.1 Data pre-processing

The data is divided in two groups, training and testing. 70.42% in training and 29.58% in testing. Normalization is applied to compress the values between 0 and 1. To achieve this, the training, testing and validation are divided by 255. Finally, using the *nputils.to_categorical* function, the class vector can be converted to a binary class matrix.

### 2.1.2 Network

The net consist of 6 layers as it can be seen in Figure 2.1. The input layer, 4 hidden layers and the output layer.

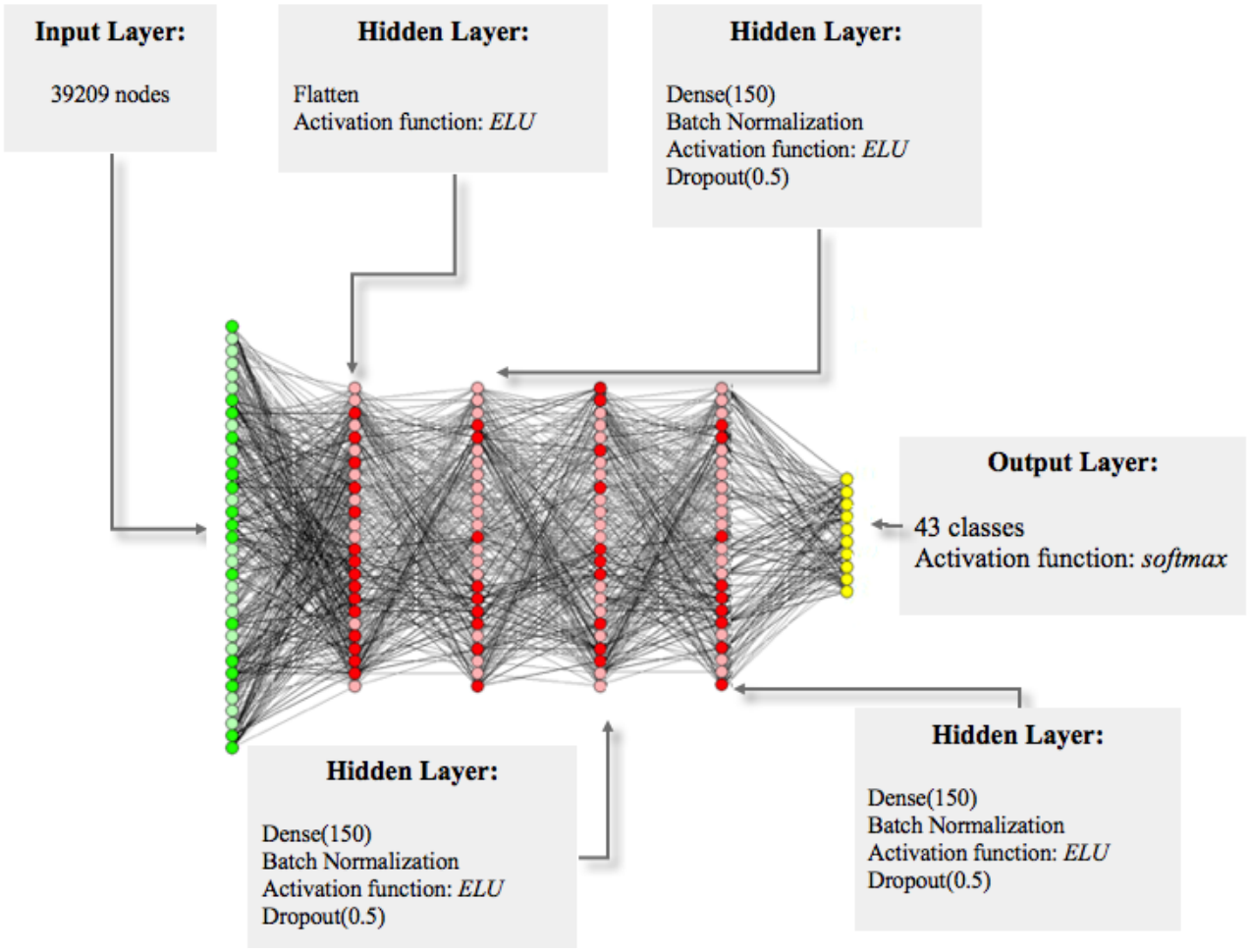



Figure 2.1: German Traffic Sign network architecture (ffNN).

- **First hidden layer:** In this layer, a flatten process is implemented to convert the data into a 1-dimensional array for the process of inputting it to the next layer. The activation function used is the Exponential Linear Unit (ELU).
- **Additional hidden layers:** The second, third and fourth hidden layers have the same structure. For fully connected layers the Dense class is used. The number of neurons can be specified in the layer as the first argument. The activation function is added using the activation argument. In addition, batch normalization and dropout are also used techniques. All of these layers consist of 150 nodes with ELU as the activation function.
- **Output layer:** The last layer of the network consists of 43 nodes equivalent to the number of classes. The Softmax activation function is used.

Finally, being this a classification problem, metrics are added and analyzed to improve the network. Due to the simplicity of the architecture, Stochastic gradient descent is considered the most adequate optimizer.

- **Total parameters:** 22632943
- **Trainable parameters**: 22632043

- **Non-trainable parameters:** 900

### 2.1.3 Results

The main results to be evaluated are the accuracy (Figure 2.2), the test loss and the time of processing. With 200 epochs and batch size of 25:

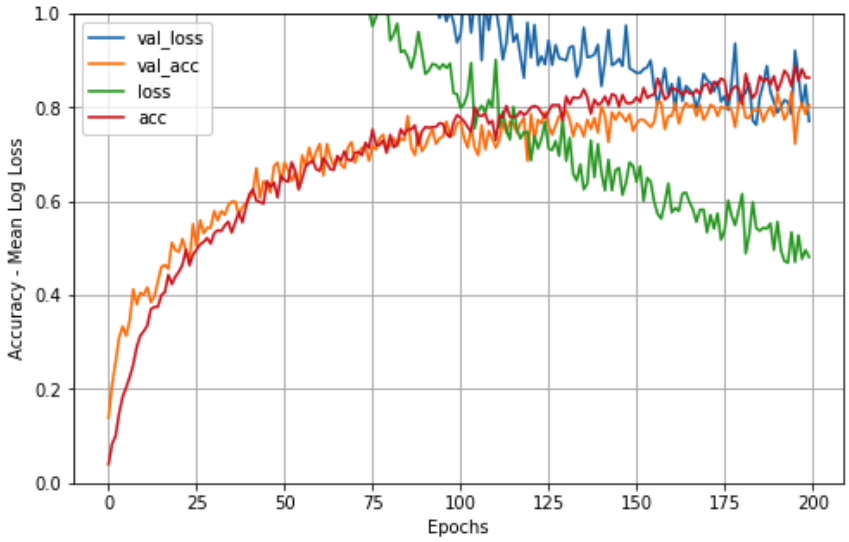



Figure 2.2: German Traffic Sign network accuracy and test loss (ffNN).

- **ffNN took:** 0.7341 seconds
- **Test loss:** 0.7679
- **Accuracy:** 0.8923

## 2.2 CIFAR 100 Dataset

The database has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. Normalization is applied.

### 2.2.1 Network

The net consist of 5 layers as it can be seen in Figure 2.3. The input layer, 3 hidden layers and the output layer.

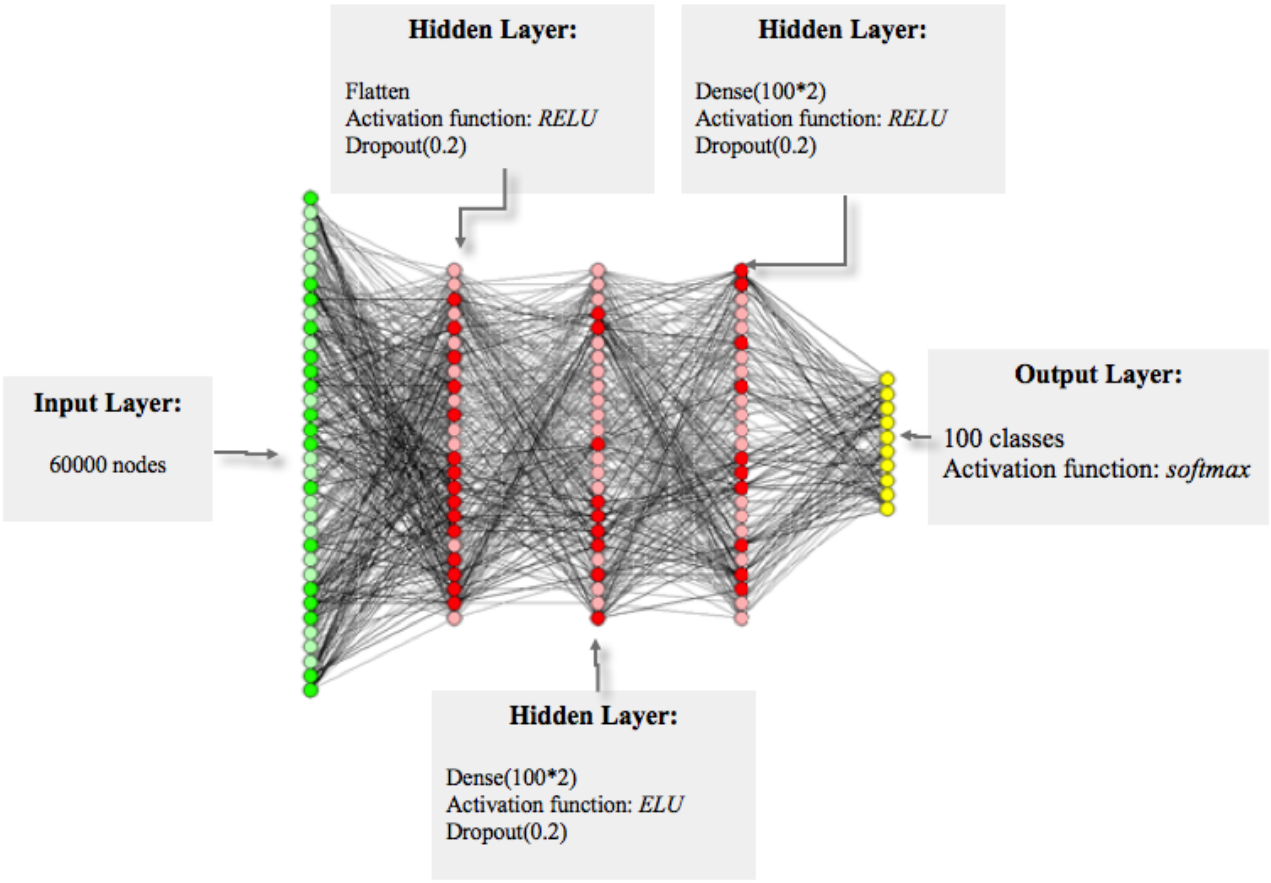



Figure 2.3: CIFAR 100 network architecture (ffNN).

- **First hidden layer:** In this layer, a flatten process is implemented to convert the data into a 1-dimensional array for the process of inputting it to the next layer. The activation function used is the Rectified Linear Unit (ReLU). Dropout is also implemented.
- **Second layer:** This layer has a dense class equivalent to two times the number of classes. The activation function is ELU. Dropout added as well.
- **Third layer:** Similar to the previous one but switching the activation function to a ReLU.
- **Output layer:** This layer includes 100 nodes, equivalent to the number of classes. For classification, the Softmax activation function is enabled.

The network can be summarized as follows:

- **Total parameters:** 674900
- **Trainable parameters**: 674900
- **Non-trainable parameters:** 0

### 2.2.2 Results

The main results to be evaluated are the accuracy (Figure 2.4), the test loss and the time of processing. With 100 epochs and batch size of 16:

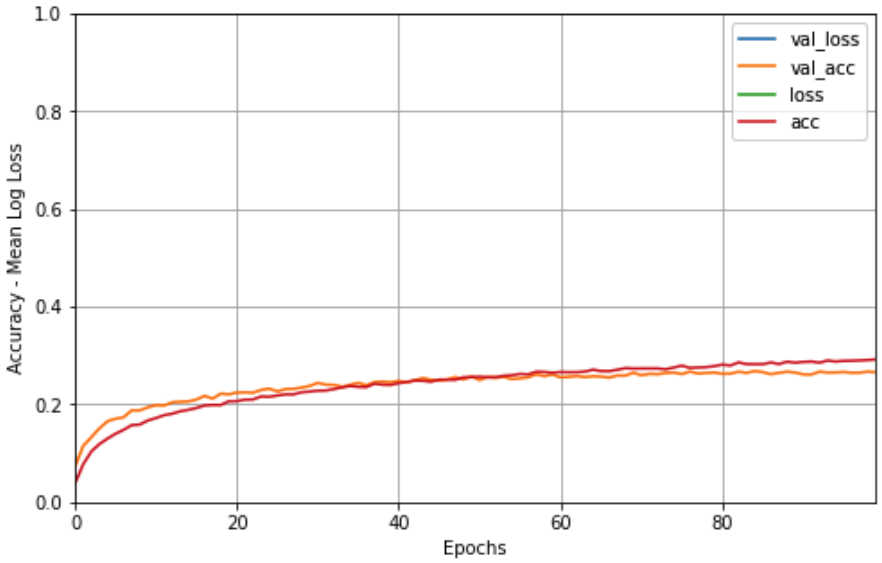



Figure 2.4: CIFAR 100 network accuracy and test loss (ffNN).

- **ffNN took:** 0.6716 seconds
- **Test loss:** 3.028
- **Accuracy:** 0.2734

As expected and as can be seen in Figure 2.4, the ffNN shows poor performance. This is expected to be improved in the following chapter.

# Chapter 3

# Convolutional Neural Network

This chapter is about object recognition using a convolutional neural network for the German Traffic Signs and the CIFAR 100 datasets. This has been carried out entirely on *Google Cloud* using a *Nvidia Tesla k80*. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

## 3.1 German Traffic Sign Benchmark

### 3.1.1 Data pre-processing

First of all, the size of the sign has been changed from (224,224) to (32,32). This is to reduce computational power usage and improve the processing speed. The portion of the data divided into training, testing and validation is the same of the proposal one.

### 3.1.2 Network

The net is composed by the input layer, 10 hidden layers and the output layer. Figure 3.1 shows in detail the structure of each hidden layer. With convolutional neural network it's not necessary to flatten the images because CNNs take into account also the geographical information. From the first hidden layer until the seventh there is an alternation of convolution and pooling layers that extract features from the source images to learn specific characteristics of the images. Then there is the flatten layer that flattens the output of convolutional neural network layers, in preparation for the fully connected layers that make the classification decision. Flattening transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier. Then, the results of the convolutional layers are fed through the last two hidden layers of our net to generate a prediction with the Softmax activation function.

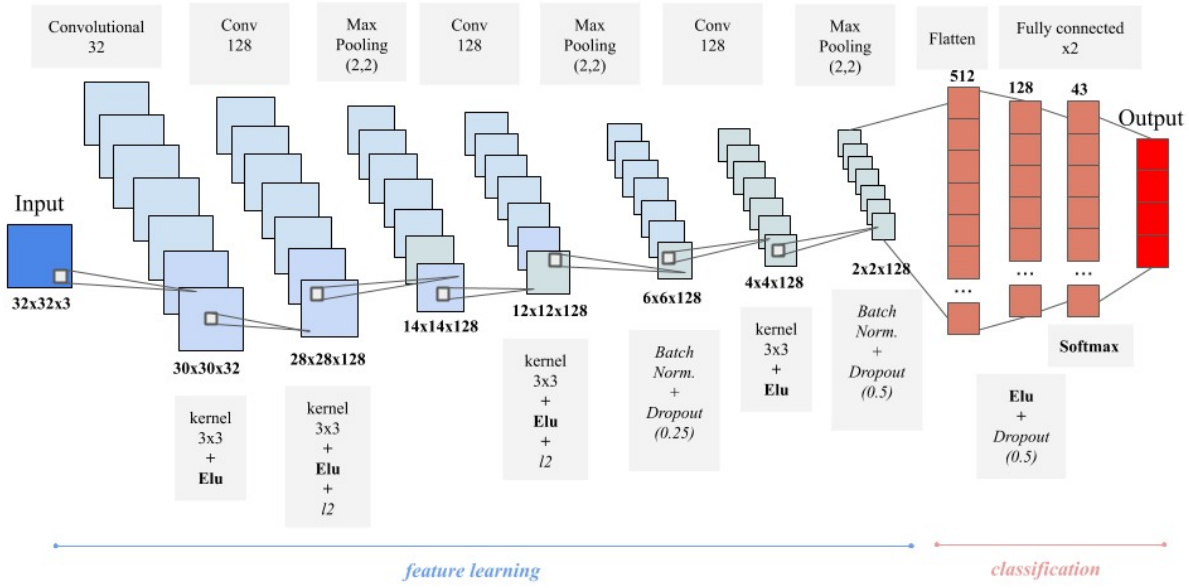



Figure 3.1: German Traffic Sign network architecture (CNN).

The usage of a CNN allows to reduce significantly the number of parameters compared to a ffNN, indeed the net generated:

- **Total parameters:** 405291
- **Trainable parameters**: 404779
- **Non-trainable parameters:** 512

### 3.1.3 Results

After reducing the size from (224,224) to (32,32), the first attempt was to add the L2 regularization to the convolutional layers to reduce overfitting of the proposed net. The other changes consisted in implementing "Adam" as the optimizer and the reduction of the epochs to a mere 50. Obtaining the first decent accuracy of 0.8925. The second improvement in terms of accuracy was a raise to 0.9529. This was due to the addiction of 3 pooling layers and some batch normalization. The decision of switching the ReLU activation functions for ELU functions was also made in this decision process.

So the final network holds an "Adam" optimizer and categorical cross entropy as the loss function. The batch size is 50 and the number of epoch is 100. The main results to be evaluated are the accuracy (Figure 2.2), the test loss and the time of processing.

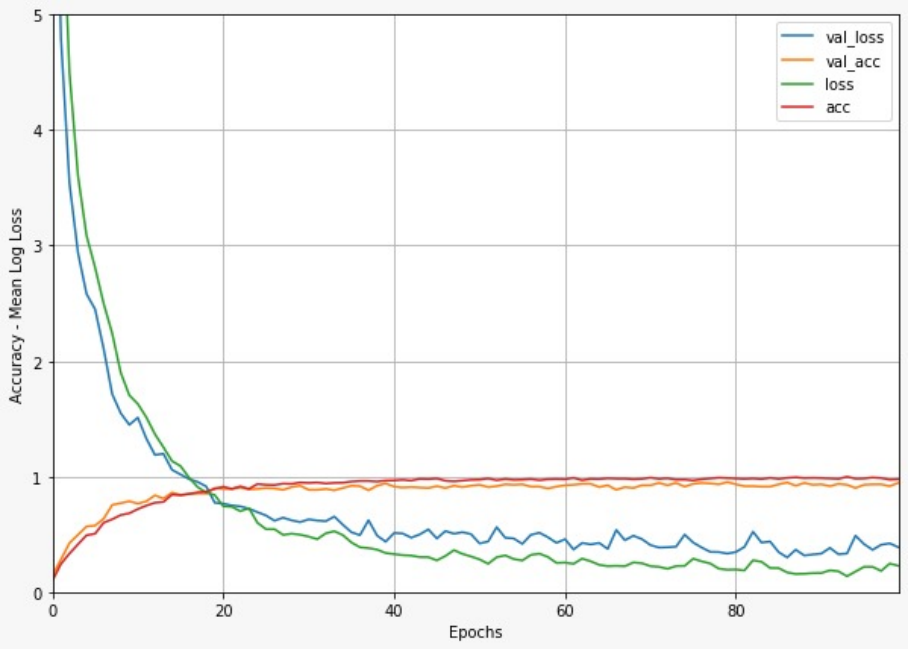



Figure 3.2: German Traffic Sign network accuracy and test loss (CNN).

- **CNN took:** 0.9244 seconds
- **Test loss:** 0.2996
- **Accuracy:** 0.9584

Figure 3.2 shows the development of the accuracy and the loss function in the training and in the validation dataset. The model has a good level of variance since the curves related to training and validation sets aren't distant from each other in the plot.

### 3.1.4 Data augmentation

Even if the model has already a good value of accuracy, we used data augmentation by a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better. Indeed it helped to achieve a better accuracy. With 100 epochs and a batch size of 50 the results are as follows.

- **CNN took:** 0.8691 seconds
- **Test loss:** 0.2216
- **Accuracy:** 0.9722

Figure 3.3 shows that even after data augmentation the variance is under control.

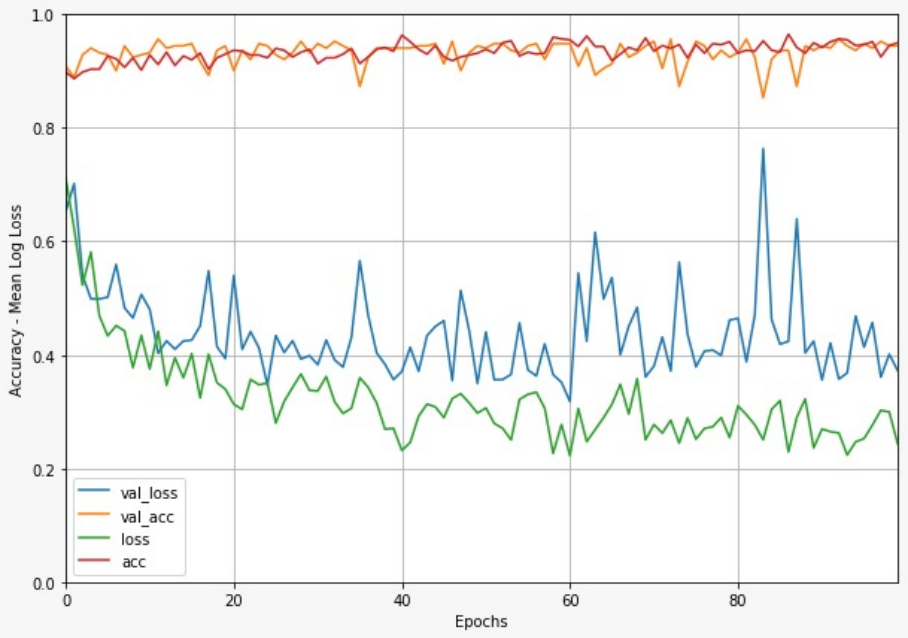



Figure 3.3: German Traffic Sign network accuracy and test loss after data augmentation (CNN).

## 3.2 CIFAR 100 Dataset (CNN)

### 3.2.1 Data pre-processing

Similarly to the previous dataset the size has been changed from (224,224) to (32,32).

### 3.2.2 Network

The net is composed by the input layer, 10 hidden layers and the output layer. The Figure 3.4 shows in detail the structure of each hidden layer. The Net has basically the same structure of the one used in the first dataset, except for the choice of the activation functions that in this net are ReLUs (except for the Softmax function).

Also in this case the usage of a CNN allowed to reduce the number of the parameters generated by the net compared to the ffNN, even if it is not as significant as in the previous dataset:

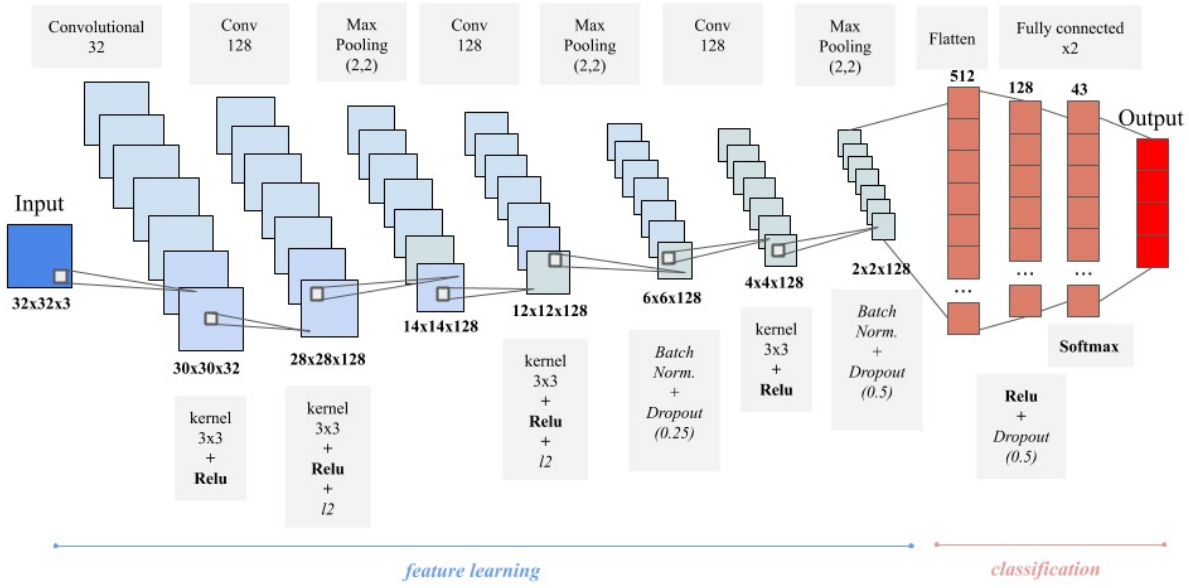



Figure 3.4: CIFAR 100 network architecture (CNN).

The network can be summarized as follows:

- **Total parameters:** 494564
- **Trainable parameters:** 494052
- **Non-trainable parameters:** 512

### 3.2.3 Results

The optimizer used is "Adam" and the loss is categorical cross entropy, being the number of epochs 25 and the batch size 16.

- **CNN took:** 1.061 seconds
- **Test loss:** 2.4692
- **Accuracy:** 0.4235

Figure 3.5 shows an interesting pattern of curves. The validation accuracy being higher than the training accuracy. This is probably due to the amount of measures implemented in the architecture to reduce overfitting, like dropout.

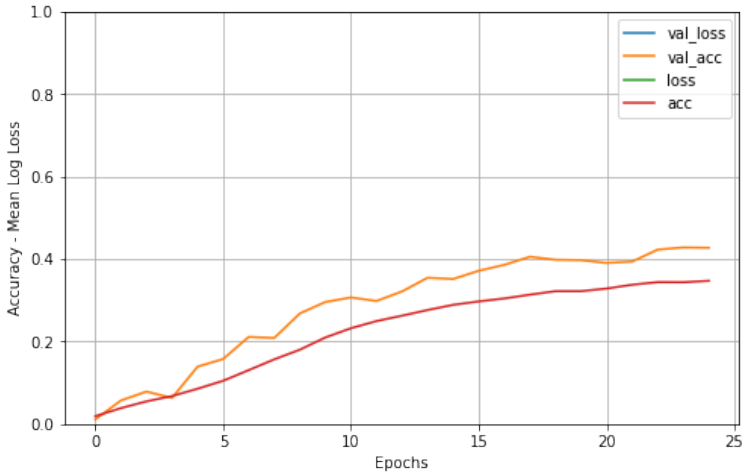



Figure 3.5: CIFAR 100 network accuracy (CNN).

In the first two assignments, we implemented different ffNN and CNN architectures and used them for classification of the same two datasets. The goal was to compare the performance of ffNN and CNN networks and find an architecture having the highest classification accuracy. Our results indicate that, CNNs performs better than FFNNs as expected for this kind of tasks. Moreover, by a proper choice of the architecture, it's possible to obtain better results for deeper networks. Indeed in the major part of the attempts adding more hidden layer helped us to increase the accuracy value, but the processing time also increased in a obvious correlative way. The biggest limitations of using a ffNN to do an object classification are related to the number of the parameters in the net and to the fact that flattening the data causes loss of geographical informations. Indeed, the usage of a CNN instead of a ffNN allowed us to achieve a better accuracy and to also to save parameters and reduce time. The differences between the results can be appreciated in the Tables 3.1, 3.2 and 3.3.

Table 3.1: Comparison of accuracy.

| **ACC.** | GTSB | CIFAR 100 |
|---|---|---|
| ffNN | 0.8923 | 0.2734 |
| CNN | 0.9723 | 0.4263 |

Table 3.2: Comparison of processing time.

| **Time (s)** | GTSB | CIFAR 100 |
|---|---|---|
| ffNN | 0.7341 | 0.6716 |
| CNN | 0.9244 | 1.0641 |

Table 3.3: Comparison of number of parameters.

| **Param.** | GTSB | CIFAR 100 |
|---|---|---|
| ffNN | 2263294 | 674900 |
| CNN | 405291 | 494564 |

# Chapter 4

# Object detection: Traffic Signs

This chapter will focus in applying all the knowledge acquired in the previous chapter to manage to build a network that is capable of recognizing and detecting traffic signs. This is achieved by using Region Proposal Network (RPN) and Region-based Convolutional Neural Network (R-CNN).

## 4.1 Data pre-processing

Similarly to the previous networks, the sign size is reduce to (32,32) to improve processing time and reduce computational power needs.

## 4.2 Architecture and strategy

In this following section, it is showcased the used strategy.

### 4.2.1 Filtering

Before running the detector, 2 filters are applied. To make use of these filters, the height and width are defined in (4.1) and (4.2).

$$Height = |bbox_2 - bbox_0| \tag{4.1}$$

$$Width = |bbox_1 - bbox_3| \tag{4.2}$$

1. **Square filter:** The purpose of this filter is to make sure that all the boxes are squared by applying Equation (4.3). After calculating the ratio, the filter exclude all values outside the range shown in Equation (4.4).

$$Ratio = \frac{Height}{Width} \tag{4.3}$$

$$0.75 \leq Ratio \leq 1.25 \tag{4.4}$$

2. **Box size filter:** This filter will prevent boxes that are considered too big to be included. Values greater than 105 in both height and width are excluded.

### 4.2.2 Detector

After the filtering a binary CNN is applied as a detector. This CNN purpose is to detect any sign. To achieve this, it only operates with 2 classes (Figure 4.1).

1. **Sign** (TRUE)
2. **Not a sign** (FALSE)

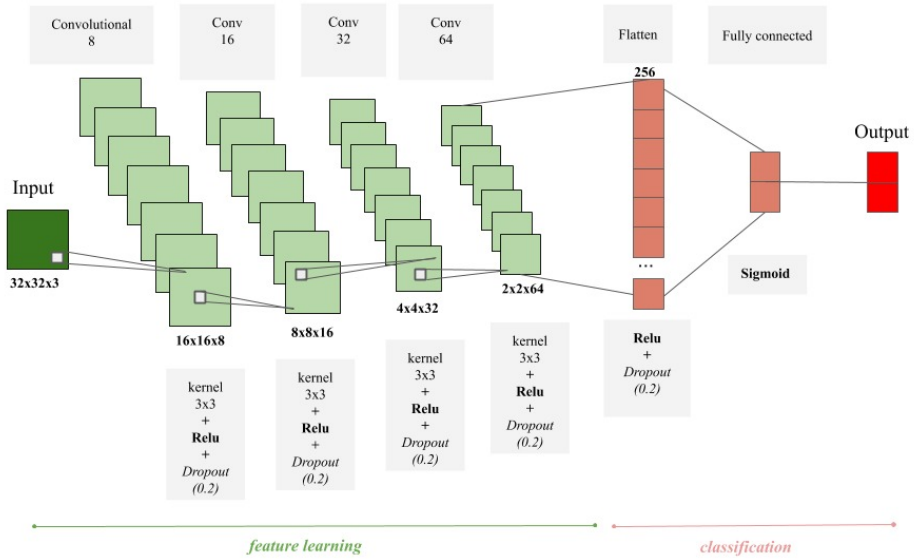



Figure 4.1: Binary detector (CNN).

This is attained by labeling all the signs as one class and "everything else" as the other class, making use mainly of the already predefined boxes in the *ground-truth* labels.

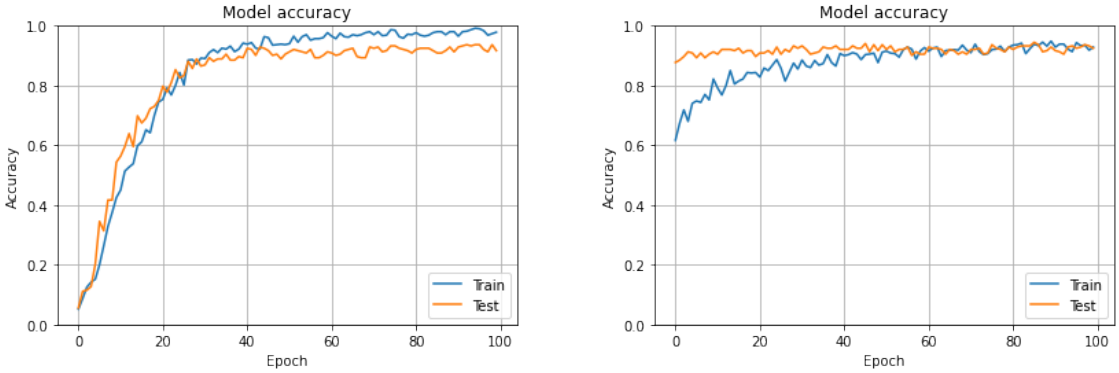
Using ADAM as the optimizer and the Sigmoid function instead of Softmax, the network can be summarized as follows:

- **Total parameters:** 25042
- **Trainable parameters:** 25042
- **Non-trainable parameters:** 0

With the accuracy and test lost after data generation being:

- **Test loss:** 0.0302
- **Accuracy:** 0.9927

In Figure 4.2 (a), a slight divergence between the training and testing curves appears as the epochs progress. This overfitting is corrected by augmenting the data for training as can be seen in Figure 4.4 (b)




(a) Model accuracy

(b) Model accuracy (Data augmentation)

Figure 4.2: Accuracy of detector previous and after data augmentation.

### 4.2.3 Signs classification

When the signs have been detected by the binary CNN, the classifier identifies every unique sign in the data set. To achieve this a new CNN architecture is established. It is important to notice that this algorithm only applies if the sign is detected by the detector.

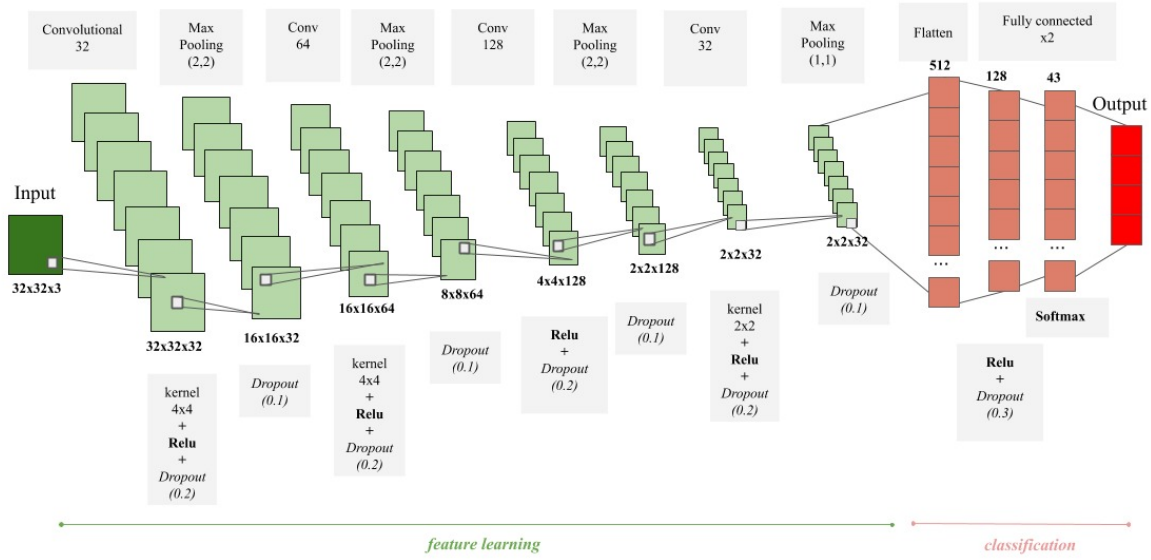


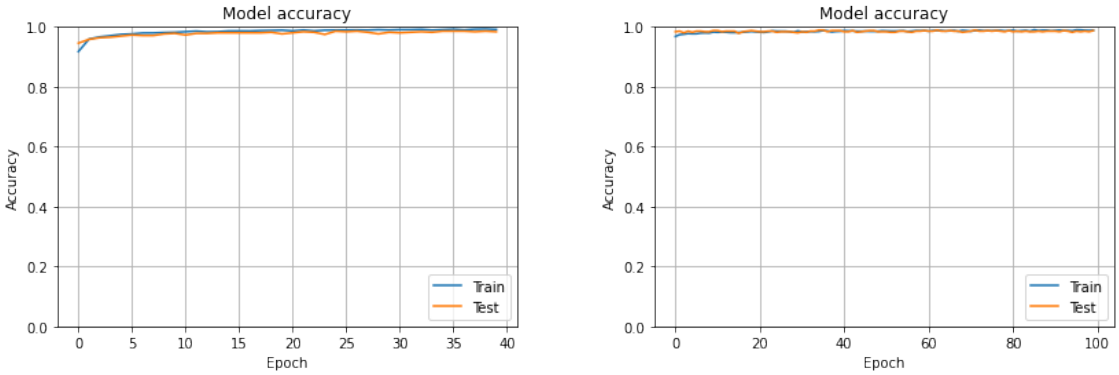

Figure 4.3: Sign classifier (CNN).

As it can be seen in Figure 4.3, the output consists of 43 classes, each one being a unique sign. Using ADAM as the optimizer, the network can be summarized as follows:

- **Total parameters:** 358187
- **Trainable parameters:** 358187
- **Non-trainable parameters:** 0

With the accuracy and test lost after data generation being:

- **Test loss:** 0.1512
- **Accuracy:** 0.9723




(a) Model accuracy

(b) Model accuracy (Data augmentation)

Figure 4.4: Accuracy of sign classifier previous and after data augmentation.

### 4.2.4 Additional changes

As can be seen in Tables 4.1 and 4.2, some parameters have been changed to reach the best outcome for this architecture. This was an iterative process, keeping the combination of settings that worked best.

Table 4.1: Anchor related parameters changes.

| | |
|---|---|
| ANCHOR_RATIOS | **[0.7, 1.0, 1.3]** |
| ANCHOR_STRIDE | **16** |
| ANCHOR_SIZES | **[32, 64, 128, 256, 512]** |

Table 4.2: NMS related parameters changes.

| | |
|---|---|
| TEST_PRE_NMS_TOPK | **5000** |
| TEST_POST_NMS_TOPK | **1500** |
| PROPOSAL_NMS_THRESH | **0.5** |

## 4.3 Results

To evaluate this network, the AP (Average precision) metric is used. This metric measures the accuracy of object detectors. Average precision computes the average precision value for recall value over 0 to 1. The mAP is the mean over all classes.

### 4.3.1 Progress

The strategy defined in Section 4.2 has been achieved following an iterative process with the purpose of attaining the best mAP score, as can be seen in Figure 4.5.

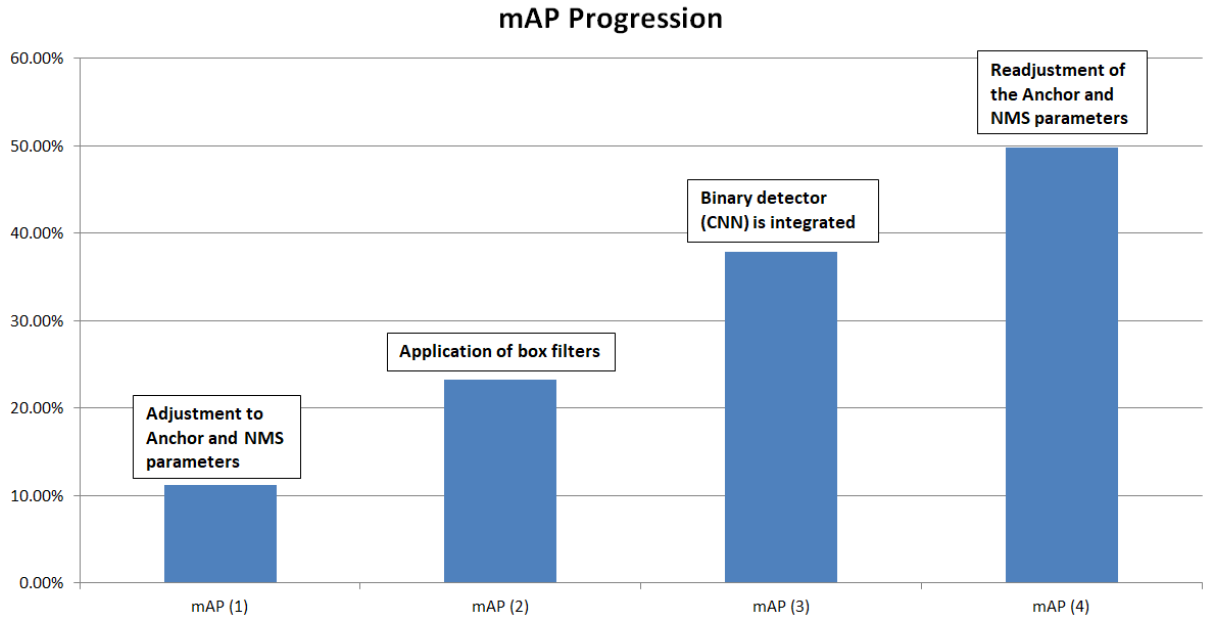



Figure 4.5: mAP progress during the process of improving the network's performance.

At the end, by building this architecture, the maximum mAP score achieved is of **49.87**%. The time of evaluation was of **153.17** seconds. One of the measures that could have been used to improve the mAP score reside in the data. It is noticeable that some of the signs get a very high score (up to a 1) while others get low scores (up to 0). One way of solving this issue is to start focusing in the signs that are getting low scores and find a way to target them specifically.

Notice how the total number of parameters is very small in both networks, detector (∼25000) and classifier (∼360000) compared to what would be expected from a problem this size. One of the main efforts of this assignment was trying to improve efficiency as much as possible. This being, reducing the computational power and time required while keeping a high score.

# Chapter 5

# Conclusion

In a more personal way, this set of assignments have not proven easy to tackle. A lot of additional investigation was needed specifically for the last of the assignments. The current situation in Spain (COVID-19 Crisis) has only put more strains in the efforts, due to impediments, that required more attention, suffered by members of this team. At the end we consider that we delivered proper results but improvement is still possible. In the future, when free time becomes more available, further analysis of the algorithm is for sure going to take place, just for the sake of learning and curiosity.

# Bibliography


[1] Luis Baumela, Iago Suárez, *Unit 2: Deep Learning for Computer Vision*, Master in Data Science, Deep Learning course, UPM, 2020.

[2] Karen Simonyan and Andrew Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, University of Oxford, 2015.

[3] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, University of Toronto, 2012.

[4] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba, *Interpreting Deep Visual Representations via Network Dissection*, 2018.

[5] Keras Blog, *Building powerful image classification models using very little data*, `https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html`.

[6] missinglink.ai, *Keras Conv2D: Working with CNN 2D Convolutions in Keras*, `https://missinglink.ai/guides/keras/keras-conv2d-working-cnn-2d-convolutions-keras/`.

[7] missinglink.ai, *Using the Keras flatten operation in CNN models with code examples*, `https://missinglink.ai/guides/keras/using-keras-flatten-operation-cnn-models-code-examples/`.

[8] Raghul Asokan, *Anchors and Object Detection*, `https://towardsdatascience.com/neural-networks-intuitions-5-anchors-and-object-detection-fc9b12120830`.

[9] Tomasz Kacmajor, *Traffic Sign Recognition using Convolutional Neural Network*, `https://hackernoon.com/traffic-sign-recognition-using-convolutional-neural-network-8a1f90e8fb24`.