POLITÉCNICA

REPORT:
ASSIGNMENT FOR UNIT 2: COMPUTER VISION

Deep Learning

Stefano Baggetto
Angel Igareta
Giorgio Segalla

April 3, 2020

1 Introduction

The aim of this report is to describe the work done using deep learning techniques for image classification on two different datasets and finally use the knowledge acquired to solve a real world problem: traffic sign detection and classification.

As first approach to solve image recognition problems in the assignment 2.1, we have been asked to use feedforward Neural Networks (ffNN). Despite this not being the best practice for solving an image classification problem, the work done and approach taken provided the understanding and the knowledge on how deep neural networks work. Moreover we got familiar to their architectures and parameters.

The second approach used in the assignment 2.2 was based on Convolutional Neural Networks (CNN). CNNs are one of the most used tools for images classifications nowadays. Carrying out this work, we understood the differences between ffNNs and CNNs, how CNNs work and we explored some popular architectures.

Finally, in the last assignment, the work was focused in traffic signs detection and classification, where we were provided with some pictures and we have to detect and classify the traffic signs that appears in those pictures.

1.1 Structure

After the introduction and the presentation of the datasets, the document will present the approaches adopted for the image classification using a ffNN and CNN along with the multiple tests done. Different network architectures have been designed in order to accomplish the classification test, reaching desirable performance in terms of time and accuracy.

After the description of the simpler classification problem, it will be presented in depth the adopted strategy for solving the traffic sign detection problem, the encountered obstacles and the action made to overcome those issues.

1.2 Datasets

First of all, let us introduce the datasets used. In the first two assignments two datasets were used: German Traffic Signs Detection Benchmark (GTSDB) and CIFAR-100 from Keras. In the last assignment, as we only had to work with traffic signs, we did not use CIFAR-100 but we did introduce another dataset in order to augment our train data, this was the German Traffic Sign Recognition Benchmark (GTSRB).

1.2.1 GTSDB

The German Traffic Signs Detection Benchmark is a detection assessment for researchers with focus in the field of computer vision, pattern recognition and image-based driver assistance. It was introduced on the IEEE International Joint Conference on Neural Networks 2013.

As it consist on a set of images of the whole road and not specific traffic signs, in order to use this dataset for the classification tasks of the first two assignments we cropped the signs from the ground truth bounding boxes and used them as train data. The features of this dataset are:

- Total full images: 900. Total signs for classification: 1213 with 43 classes.
- Train, validation, test, split: 50%, 20%, 30%
- Images format: PPM
- Image sizes: 16x16 to 128x128

1.2.2 CIFAR-100

The second dataset is called CIFAR-100, as it consists of 100 classes containing 600 images each. It is used popularly in order to pretrain CNN with a big amount of images. As compared with the previous dataset, the images here were cropped and they contain only a specific class.

This dataset was used in the first two assignments to test the developed neural networks with a bigger amount of images, in order to perceive a higher difference in the precision with our modifications, such as with data augmentation.

The features of the CIFAR-100 are:

- Total images: 60.000 with 100 classes,
- Train, validation, test, split: 66.67%, 16.67%, 16.67%
- Data source: Keras
- Image sizes: 32x32

1.3 GTSRB

The German Traffic Sign Recognition Benchmark is taken from the same source as the first one but this time is focused for classification challenges. It contains the same amount of classes but higher amount of data, so we decided to use it in order to augment our data for the traffic sign classifier in the last assignment as we would explain further in this report. Its features are:

- Total images: 50,000 with 43 classes.
- Train, validation, test, split: 80%, 10%, 10%
- Images format: PPM

2 ffNN Assignment

In the first assignment, our task was to classify the images from the first two datasets with a feedforward neural network. A ffNN is the first and simplest type or artificial neural network. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes, if there are any and to the output nodes.

2.1 Data preparation

In order to feed the images to the network, the input data must be transformed in a compatible format. The transformation done was to normalize the pictures, converting the pixels in values in the range [0, 1]. Besides, the classes have been one-hot-encoded in order to use categorical crossentropy as loss function.

2.2 Preprocessing

In order to get a higher accuracy in our models, we tested by preprocessing the images before feeding them to the neural networks. Some of the approaches followed for the first dataset were converting the images to grey-scale or the Gaussian blur transformation to reduce noise. Apart from those, for the second dataset we tried to apply Laplacian and Canny transformations. However, neither these actions did not improve the network performance.

2.3 Model Structure

Our ffNN structure for both datasets is based on 3 to 4 dense layers with decreasing number of nodes and dropout between each layer. In order to choose the number of nodes, dropout parameters, learning rates, optimizers and activation functions we made use of hyperparameter tuning. Here are some of the tested components:

- Dense layers sizes: As we previously commented, we experimented with different sizes per layer, with sizes from 150 to 1000.
- Dropout: It was used for reduce overtifitting in our neural network by preventing complex co-adpation on training data. The values we tested with were 0.1, 0.15, 0.2, 0.3.

- Activation Function: ReLU, eLU, Leaky ReLU and tanh.
- Optimizers: Adam, Adamax, Nadam, SGD.
- Learning Rate: $1e^{-03}$, $1e^{-04}$, $1e^{-05}$, $5e^{-05}$, $1e^{-06}$.

2.4 Hyperparameter Tuning

2.4.1 GTSDB

In the next table are presented the some of the combinations of hyperparameters we tested for the GTSDB dataset 1.

HyperParameters				**Result**	
Num_Units	**Dropouts**	**Optimizer**	**Learning_Rate**	**Accuracy**	**Test_Loss**
256	nadam	0.15	5e-05	88.1%	0.622
256	nadam	0.2	5e-05	86.14%	0.600

Table 1

2.4.2 CIFAR-100

Some of the combinations result for cifar-100 are represented in the table 2.

HyperParameters				**Results**	
Num_Units	**Optimiezer**	**Dropout**	**Learning_Rate**	**Accuracy**	**Test_Loss**
300	adam	0.1	1e-06	12.14%	3.921
300	adam	0.1	1e-05	22.57%	3.309
300	adam	0.1	5e-05	28.08%	3.055
300	adam	0.2	1e-06	11.53%	3.922
300	adam	0.2	1e-05	22.29%	3.313
300	adam	0.2	5e-05	28.74%	3.041
300	nadam	0.1	1e-06	11.81%	3.926
300	nadam	0.1	1e-05	22.04%	3.315
256	nadam	0.15	5e-05	28.06%	3.021

Table 2

2.5 Final configuration

2.5.1 GTSDB

The configuration which we obtained better results in balance with elapsed time for the first dataset was:

- Dense layers sizes: As it can be appreciated in the figure 1, the best result was to add three dense layers, the first one with 300 nodes,the second with half of the size and the third one the number of classes.
- Dropout: 0.3.
- Activation Function: ReLU.
- Optimizers: SGD.
- Learning Rate: $1e^{-03}$.
- Batch Size and epochs: 128 and 120. We stopped it at 120 in order to avoid overfitting.

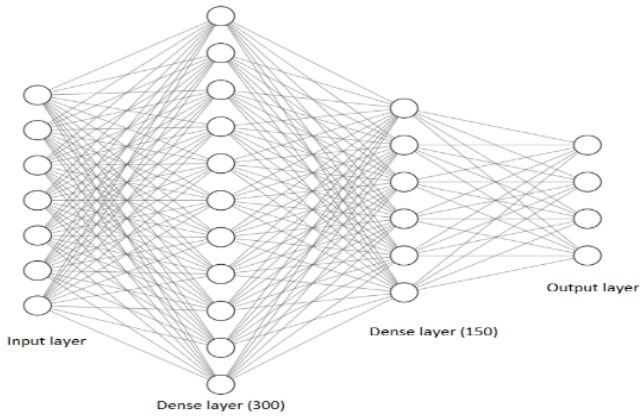
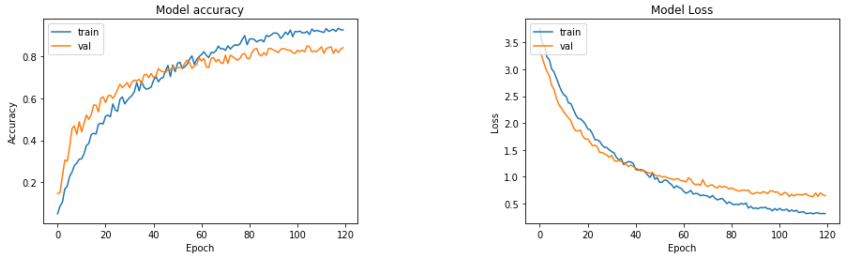


Figure 1: ffNN Model for GTSDB dataset

The final results obtained with this configuration where:

- Train Accuracy and Loss: 92.5% and 0.3143.
- Validation Accuracy and Loss: 84.47% and 0.8413.
- Test Accuracy and Loss: 89.47% and 0.453. Elapsed time: 0.2258s



(a) (b)

Figure 2

2.5.2 CIFAR-100

The configuration which we obtained better time-accuracy ratio for the second dataset was:

- Dense layers sizes: As it can be appreciated in the figure 3, the best result was to add four dense layers, with number of nodes: 1024, 768, 512 and the number of classes.
- Dropout: 0.2.
- Activation Function: ReLU.
- Optimizers: SGD.
- Learning Rate: $1e^{-03}$.
- Batch Size and epochs: 64 and 150.

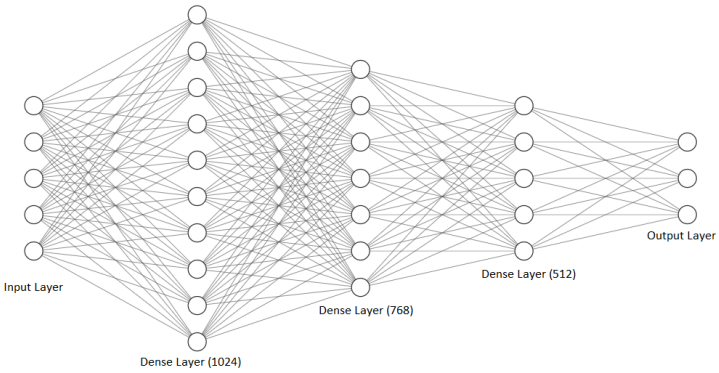
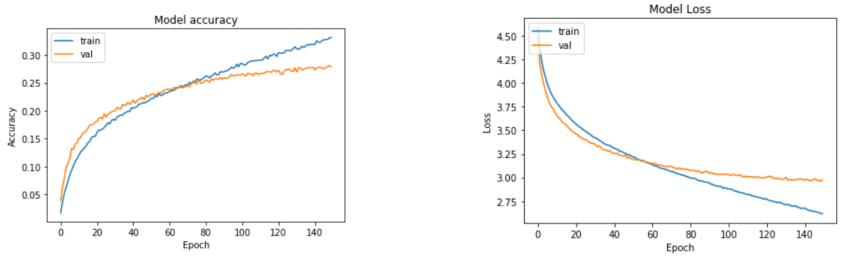


Figure 3: ffNN Model for CIFAR-100 dataset

The final results obtained with this configuration where:

- Train Accuracy and Loss: 33.16% and 2.6182.
- Validation Accuracy and Loss: 27.88% and 2.9767.
- Test Accuracy and Loss: 28.53% and 2.966. Elapsed time: 0.904s



(a) (b)

Figure 4

2.6 Conclusion

Carrying out this first task we have been able to put in practice the knowledge acquired during the previous classes. First we understood the problem and what we had to do in order to find a solution. We proceeded adapting the images to a format that our neural network could understand and we applied some transformations to see if we obtained some improvements. Once we found our desired design we wanted to achieve even better results therefore we applied parameter tuning. Finally we obtained 89,47% accuracy for the GTSDB dataset and 28,53% on the CIFAR100.

3 CNN Assignment

In this second assignment the goal was to do the same classification task as in the previous assignment but using Convolutional Neural Networks. Therefore we added convolutional and max pooling layers to the neural network in order to classify traffic signs in a better way than a simple feedforward network.

Note that the data preparation and preprocessing approaches were the same as in the previous assignment. Our focus here was in trying different models to see which gave us the best accuracy and performance and finally perform hyperparameter tuning.

3.1 Models used for GTSDB

In order to come up with the best model structure for the GTSDB dataset, we tested several ones.

3.1.1 Lenet-5

Yann LeCun, Leon Bottou, Yosuha Bengio and Patrick Haffner proposed a neural network architecture for handwritten and machine-printed character recognition in 1990's which they called *LeNet-5*. The architecture is straightforward and simple to understand [2]. It is represented in the figure 5.

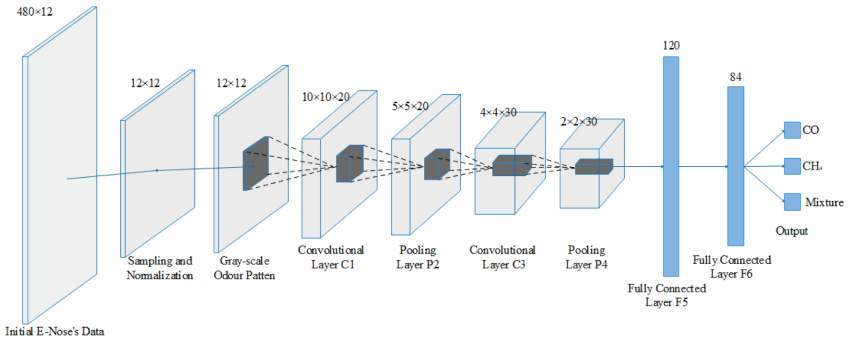
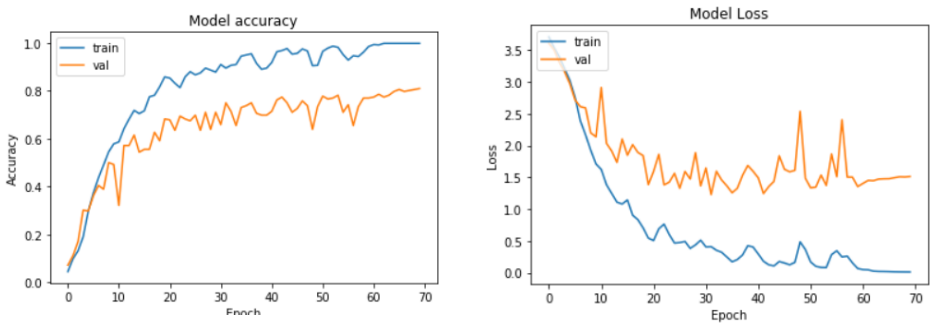


Figure 5: Lenet-5 Structure

In this structure we achieved a 85.59% of test accuracy and 1.159 of loss, with the accuracy and loss chart displayed in the figure 6.

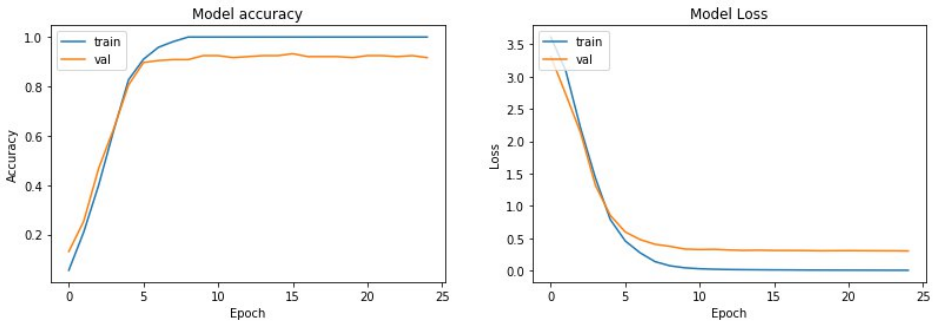


(a) (b)

Figure 6

3.1.2 Pretrained Resnet

After that we decided to test the model Resnet from Tensorflow already pretrained with imagenet weights. With it, we achieved a 95.01% of test accuracy and 0.2147 of loss. The corresponding charts displayed in the figure 7.



(a) (b)

Figure 7

3.1.3 Alexnet

AlexNet is the name of a convolutional neural network (CNN), designed by Alex Krizhevsky. It contains eight layers; the first five are convolutional layers, some of them followed by max-pooling layers, and the last three are fully connected layers.

In order to train it, we used the non-saturating ReLU activation function, which showed improved training performance over tanh and sigmoid. The used optimizer has been SGD with a learning rate of 0.001. The batch size used to feed the CNN has size 40. The architecture is shown below.

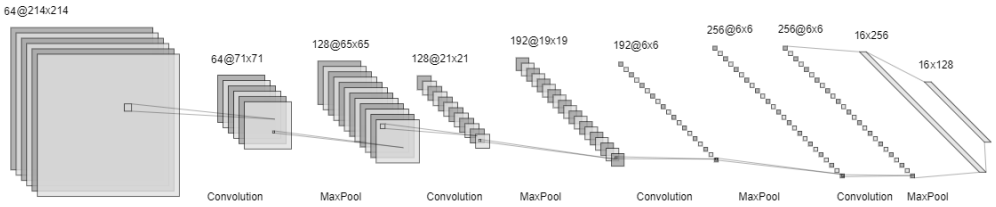
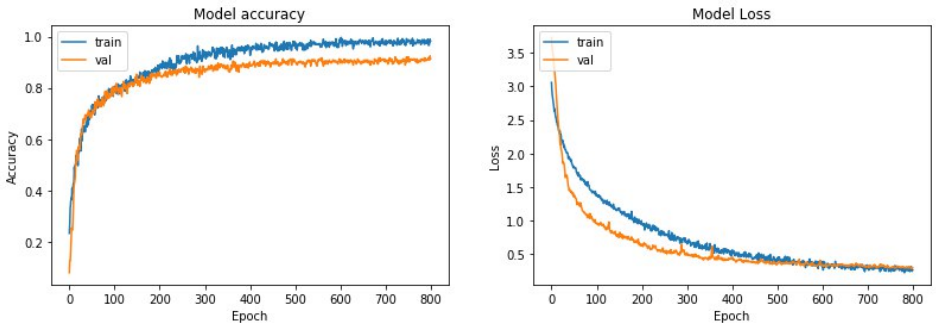


Figure 8: Alexnet Structure

The results obtained with this structure were the best so far, with a test accuracy of 95.56% and loss of 0.2014. The corresponding loss and accuracy charts can be appreciated in the figure 9.



(a) (b)

Figure 9

3.2 Models used for CIFAR-100

For solving the CIFAR-100 problem, differently from the previous approach, we decided to use our own neural network with a custom architecture. The graphical representation can be seen at the figure 10 and the layers are:

- Two Convolutional layers 128@32x32
- One MaxPool layer 128@15x15
- Two Convolutional layers 256@15x15 and 256@13x13
- One MaxPool layer 512@6x6
- Two convolutional layers 512@6x6
- One MaxPool layer 512@2x2
- Three dense layers of 1024, 1024 and 100 neurons

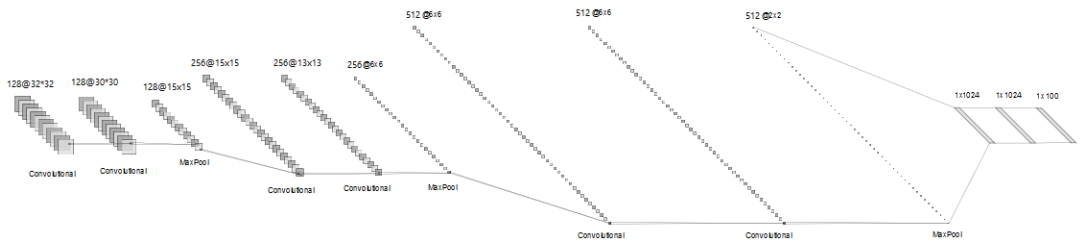
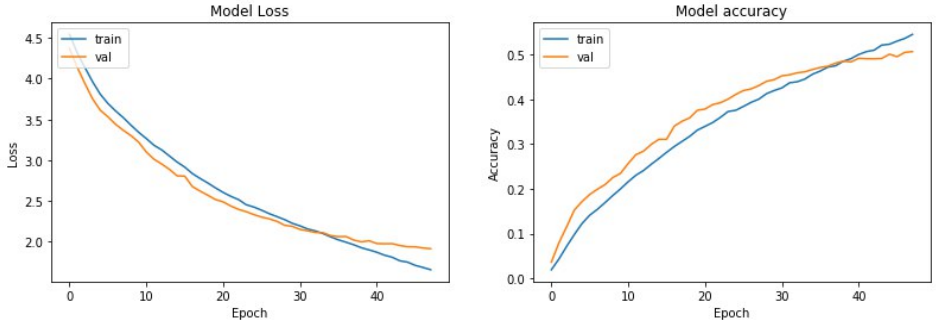


Figure 10: Custom CNN

The final configuration were we achieved better results for this structure are:

- Activation function: eLU
- Optimizer: SGD
- Learning rate: 0.01
- Batch size: 100

The results can be seen at the figure 11. We obtained a test accuracy of 50.08% and a loss of 1.8855.



(a) (b)

Figure 11

3.3 Hyperparameter Tuning

3.3.1 GTSDB

Finally, some of the tests done for this assignment related with the hyperparameters in the models to classify GTSDB images are represented in the table 3.

Hyperparameters								Results	
Model	**CNN_Layer**	**Neural_Layer**	**Pool**	**Batch_Normal**	**Dropout**	**Optimizer**	**LR**	**Acc**	**Loss**
model_1	4	2+1	Max	Yes	0.5	SGD	1e-3	94.18%	0.2232
Lenet	2	2+1	Average	No	0	SGD	1e-3	85.60%	1.1592
VGG16	13	1+1	Max	No	0	SGD	1e-3	95.01%	0.2147
Alexnet	4	2+1	Max	Yes	0	SGD	1e-3	95.57%	0.2014
Alexnet_Augmentation	4	2+1	Max	Yes	0	SGD	1e-3	50.96%	1.9680

Table 3

3.3.2 CIFAR-100

Some of the combinations result for cifar-100 are represented in the table 4.

Model	HyperParameters							Results	
Name	**CNN_Layer**	**Neural_Layer**	**Pool**	**Batch_Normal**	**Dropout**	**Optimizer**	**LR**	**Acc**	**Loss**
model_2_SGD	6	2	Max	No	Variable(0.1/0.5)	SGD	1e-03	50.85%	1.885
model_2_adam	6	2	Max	No	Variable(0.1/0.5)	adam	NA	51.96%	1.978

Table 4

3.4 Conclusion

Carrying out the second assignment we could understand the differences between ffNNs and CNNs. After understanding the new aspects of this kind of networks we proceeded designing some network architectures. In addition to that we applied some state of the art solutions for image classification problems with convolutional neural networks. During this phase we tweaked pre-existing models in order to fit as solution to ours, both in terms of execution time and performance. Once we found our desired design we wanted to achieve even better results therefore we applied again parameter tuning. Finally we obtained 95.56% accuracy for the GTSDB dataset and 50.08% on the CIFAR-100.

4 Detection Assignment

In this final assignment the task is not only to classify traffic signs but to locate them beforehand in the whole image, so a detection task is added. The followed approach was based on Faster R-CNN, in which we have to follow two stages. First we had to predict class-agnostic bounding box proposals using a Region Proposal Network (RPN) and then classify and fine-regress location of boxes. In order to classify we could use the information for the previous assignments as the object to categorize is the same: traffic signs.

The RPN was already provided so our focus was in discarding bounding boxes that were not traffic signs. However, apart from that we hypertuned the parameters for the RPN in order to generate shapes that were adequate and in line with our ground truth. As previously explained in the description of the detection dataset GTSDB (1.2.1) the traffic signs could appear in sizes from 16x16 to 128x128 and the shape was almost squared, so this data was useful in order to choose the best parameters for the RPN.

4.1 RPN Tuning

Fist of all, we needed a way of seeing how accurate was the RPN generating bounding boxes, so after their generation, we developed a script that counted the number of ground truth from **only the training boxes** where there was at least one bounding box intersecting with an IoU (Intersection over Union) higher than 0.7. We decided to use that threshold to be more severe in the hyperparameter tuning and generate boxes that then the CNN could classify better. After having that number we divided it by the total number of bounding boxes for training, resulting in the precision accuracy of the RPN.

Once we had a method to compare how did the hyperparameters affect the RPN, we did hypertuning with more than 100 tests, from where we could achieve a 67.13% as maximum with the IoU threshold of 0.7 and almost 90 if we changed it to 0.5. The most relevant parameters were anchor_ratios, anchor_sizes, number of bounding boxes before and after the NMS (Non Maxiumum Supression) and NMS threshold.

First, as the maximum width or height in the traffic signs were up to 128, we tried in the range down to 16. However, after trying 40 with these combinations the maximum precision was 56%. We discovered that by changing the sizes to be a bit bigger, the result improved a lot, reaching our maximum score. With respect of the ratios, as the signs in the images were almost squared, we tried generating boxes with small difference in ratio (lower than 0.1) and this did achieve better results. Finally, we could observe a threshold between detection speed and the number of bounding boxes generated, resulting in higher precision too. We took that into account to achieve the best speed-precision ratio. In the table 5 can be observed 10 of the 100 tests.

ANCHOR_RATIOS	ANCHOR_SIZES	TEST_PRE + NMS_TOPK	TEST_POST + NMS_TOPK	PROPOSAL + NMS_THRESH	PRECISION
(0.95, 1.0, 1.05)	(32, 62, 102, 182, 232)	30000	5000	0.82	67,13%
(0.8, 1.0, 1.2)	(32, 62, 102, 182, 232)	20000	3000	0.8	66,78%
(0.95, 1.0, 1.05)	(32, 62, 112, 212, 252)	15000	3000	0.8	66,54%
(0.93, 1.0, 1.07)	(32, 62, 102, 182, 232)	20000	2500	0.77	66,19%
(0.85, 1.0, 1.15)	(32, 62, 102, 182, 232)	10000	2000	0.8	64,78%
(0.95, 1.0, 1.05)	(32, 62, 112, 212, 252)	35000	4000	0,7	64,08%
(0.9, 1.0, 1.1)	(32, 62, 112, 212, 252)	50000	5000	0,7	63,96%
(0.8, 1.0, 1.2)	(32, 62, 112, 212, 252)	10000	1500	0.7	61,15%
(0.85, 1.0, 1.15)	(32, 48, 64, 96, 128)	10000	3500	0.7	55,98%
(0.8, 1.0, 1.2)	(16, 32, 48, 64, 128)	8000	1500	0.5	5,00%

Table 5: RPN Tuning

4.2 Bounding Boxes Reduction

In the second part of the assignment we focused on how to reduce the number of bounding boxes that were generated. The goal was to improve the mAP (mean Average Precision), which is the average of the individual average precision per class, which takes into account the IoU, precision and recall. Hence, by reducing the number of false positives we would obtain higher precision and higher mAP, so that was our first approach by using a position and shape filter.

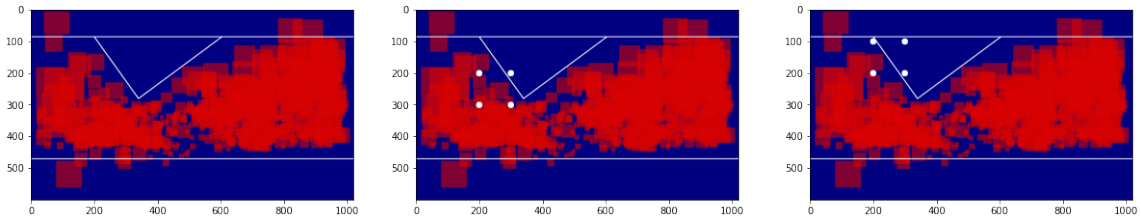
4.2.1 Position and shape filter

The first filter that helped improving the mAP and significantly reducing the evaluation time was taking into consideration the position of the bounding box and the shape of it.

Regarding the position, we created a heatmap from the ground truth of traffic signs, in order to see the distribution of signs in the images. After analyzing the result, we draw some areas were the bounding boxes rarely appear, represented in the figure 12a. As the bounding boxes contained 4 corners, we had to come out with a way to see if a box was valid or not. The filter consisted in two parts:

1. Discard boxes which have two corners with y-coordinate value lower than **85** or higher than **470**.
2. Discard boxes which have two corners in the triangle that appears in the middle of the picture. This was done by using linear equations with the coordinates of each corner [1].

An example of valid and not valid boxes can be appreciated in the boxes 12b and 12c.



(a) (b) (c)

Figure 12

Regarding the shape, despite of generating smaller bounding boxes, we limited them in width, height and ratio (to restrict the boxes to have a shape similar to square). These were some of the hyper-parameters we tried in the end, but in all of them not allowing a size lower than 13 or higher than 150.

The results of applying these two filters combined reduced the time in our tests by 2.7% in average with similar precision, so it was a big improvement in performance for this project.

4.2.2 Background CNN

The next step in our detector was to implement a CNN that was able to classify bounding boxes that contain background and traffic signs. In order to obtain the data for the CNN, we would use the train data from GTSDB and GTSRB. However, to obtain background data, we decided to use the generated bounding boxes. The approach was the following:

1. Loop over the generated bounding boxes for train_images.
2. To differentiate between background and traffic sign, we calculated the IoU of the bounding boxes with the ground truth, and if it was higher than a threshold T, we considered them traffic sign or otherwise background data.
3. For each image, take N boxes that refer to background. We did that by throwing a dice with a probability (n / number of bounding boxes) to have a representative distribution of the background data for all the images.
4. For each image, take all the possible images for traffic signs as the ratio is uneven and add them to the GTSDB and GTSRB train data. This will be useful because its a real example of the images that our detector is going to see.

5. Repeat until having 50.000 data for each class.

In order to choose the T threshold, we tried first with 0.2, but most of the images retrieved did not show the traffic sign correctly in order to appreciate its class. On top of that, despite the accuracy of the CNN trained with these images was high, the mAP decreased. Our second approach was to use a IoU threshold of 0.6, with which we obtained the best results.

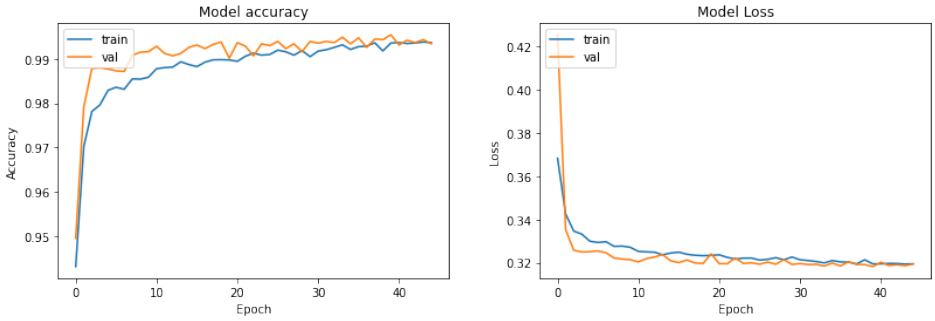
Regarding the model structure we decided to use MobileNetV2 available with Tensorflow [3], with an alpha component of 0.5. This value was chosen because this filter is applied before the traffic sign CNN, where we need higher precision, but here we need it to be fast as well, so by reducing the alpha component, we reduced the number of model parameters and hence the elapsed time. Lower values were discarded due to the low accuracy obtained.

The final results for this CNN were:

	LOSS	**PRECISION**	**LOSS + PREPROC**	**PRECISION + PREPROC**
TRAIN	0.3219	99.10%	0.3195	99.37
VAL	0.3230	99.01%	0.3194	99.34
TEST	0.3219	99.08%	0.3172	99.57

Table 6: Background CNN Results

As displayed in the table 6 the model accuracy was higher trained with data augmentation, reaching almost 99.60%. Besides, the background CNN incremented the mAP around 1.4% and the time was reduced by 3%, thanks to MobileNetV2@0.5 structure. On top of that the loss and accuracy charts (figure 13) were correct.



(a) (b)

Figure 13

4.2.3 Traffic Sign CNN

The second step was to improve the given traffic sign classifier, because, as previously explained in this same section, a higher precision contributed to a higher mAP. In order to do so, we decided to further improve the CNN of the previous assignment by training it with more traffic signs labeled data, through using the GTSRB (seen in section 1.3) database.

The train data used for the new traffic sign CNN was around 50800 images, much more than the previous 800 images from the GTSDB. This combined with data augmentation allowed us to have not only higher accuracy but a more solid CNN that could behave better with unseen images.

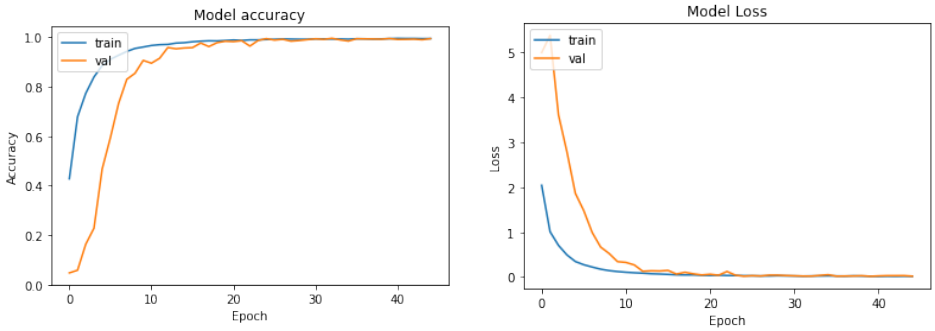
The model structure used for the Traffic Sign CNN was the same as in the Background CNN (section 4.2.2), with the difference in alpha value. The value chosen in order to have a lighter but still accurate CNN was **0.7** and the optimizer used was Adamax, after performing several tests. Besides, we used the pretrained weights of *ImageNet* for this model that made the training faster and the CNN more robust.

The results for this CNN using the previously commented approach were the following:

	LOSS	PRECISION
TRAIN	0.0129	99.59%
VAL	0.0731	98.33%
TEST	0.0254	99.49%

Table 7: Traffic Sign CNN Results

As it can be appreciated in the table 7 the results were very high and the model accuracy and loss chart were adequate (figure 14, resulting in an increment of mAP. On the other hand, the processing time increased due to the high amount of model parameters.



(a) (b)

Figure 14

4.2.4 Non Maximum Suppression

After displaying some of the bounding boxes that passed the previous filters, we realized there were multiple bounding boxes for the same ground truth, so we decided to apply Non Maximum Suppression algorithm, which consist on removing overlapping boxes taking into account an IoU threshold, number of input and output images and a score associated to each image.

The IoU threshold that provided us of better results was 0.3, meaning that if the intersection was higher than this value, we only kept one of the bounding boxes, the one with highest score in the Traffic Sign CNN prediction. Note that after some hyperparameter tuning we realized the NMS affected the result significantly. In fact a change from 0.7 to 0.3 reflected in an increment of 12% of mAP. Finally, we limited the maximum number of final bounding boxes per image to 6, as seen in the GTSDB dataset description.

4.3 Detector Results

4.3.1 Hyperparameter Tuning

As we had several configurations for the hyperparameters of the RPN to choose from (previously seen at table 5), here there are some of the hyperparameter tuning we did for one configuration, specifically the second in that table. We are not able to add all the tests in this report but the configurations with highest generation precision were tried and the most relevant parameters for the filters were hypertuned, displayed in the table 8.

BG CNN SCORE	TS CNN SCORE	MIN_SIZE	MAX_SIZE	IOU_THRESHOLD	mAP
0.999994	0.92	14	128	0.15	63.68%
0.999999	0.9	14	150	0.2	63.64%
0.999999	0.86	14	150	0.2	64%
0.999999	0.945	14	150	0.25	64.68%
0.99	0.93	14	150	0.2	63%
0.999999	0.955	14	200	0.2	64.35%

Table 8: Detector Results

4.3.2 Final results

The best results in respect of ratio precision-speed were obtained with the following parameters.

With respect of the RPN:

- Anchor ratios: (0.8, 1.0, 1.2)
- Anchor sizes: (32, 62, 112, 212, 252)
- Number of bboxes before NMS: 10000
- Number of bboxes after NMS: 1500
- NMS threshold: 0.7

Regarding the filters and models hyperparameters:

- Min size allowed: 16
- Max size allowed: 130
- Probability threshold Traffic Sign CNN: 0.999999
- Probability threshold Background CNN: 0.93
- IoU threshold: 0.3

With this configuration we obtained a final mAP of 66.28% in 2183 seconds.

4.4 Conclusion

Through this assignment we have comprehended the importance of the Region Proposal Generator in the two stages approach. By tuning the parameters of the RPN we achieved better and more realistic bounding boxes, despite probably changing the generator weights or model could have resulted in a faster detector. On top of that, the metrics of the train data such as size, shape and position heatmap were essential to achieve our final score and that can be interpolated to any Machine Learning problem, analyze the data is crucial. Finally, the CNN able to distinguish between background images and traffic sign allowed us to have a faster and more accurate model.

References

[1] HUSE360. Cómo saber si un punto está dentro de un triángulo, Dec 2019.

[2] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86* (12 1998), 2278 – 2324.

[3] SANDLER, M., HOWARD, A., ZHU, M., ZHMOGINOV, A., AND CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks, 2018.