Deep Learning: Assignment of Unit 3

Computer Vision Final Report: Traffic sign detection problem

Submitted To:
Luis Baumela
Roberto Valle

Submitted By :
Michail Gongolidis
Alejandro Gonzalez Gonzalez

1 Introduction

The objective of this activity is to follow the construction process of a convolutional neural network for the Traffic sign classification and detection problem. This problem was divided in sub problems throughout the Deep Learning Unit 3 class. The sub problems are listed below.

1. Design a feed forward Neural Network to classify signs.
2. Design a convolutional Neural Network to classify signs.
3. Retrain an already trained Network with transfer learning.
4. Design a network that detects and recognizes signs in images.

The present report explains the process it was followed to solve the aforementioned sub problems and explains the intuition and the logic behind the proposed solutions. Furthermore, the difficulties and problems faced during the process will be mentioned. Lastly, comments on the class structure and recommendations for next year are stated.

The German Traffic Sign Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. It is our dataset under study and it consist of 43 classes (Unique traffic sign images). In the first part of the assignment the training images dataset had a size of 852 images and the test images dataset had a size of 361 images. The total size of the dataset was modified during the implementation of the solutions. The process of the data exploration and data augmentation will be explained in the corresponding section in order to highlight the understanding of the problem throughout time.

The ultimate goal of the project was to be able to detect traffic signs in images, and once detected to classify them among one of the 43 different traffic signs classes.The selected working environment was Google Colab, which offered a GPU Tesla T4, with a good computing capability and 15 GB of RAM memory.

2 Theoretical background

Convolutional Networks (ConvNets) are a type of efficient neural networks that achieve impressive performances in perceptual tasks such as object recognition. Their architecture is loosely inspired by the visual cortex. In 2012 AlexNet, a type of ConvNet, won by a large margin the ILSVRC 2012 competition, starting the huge wave of interest in deep learning that continues today. In 2019, the state of the art architecture for object detection is ResNet, which is a type of ConvNet.

In the vision, a receptive field of a single sensory neuron is the specific region of the retina in which something will affect the firing of that neuron i.e., that will active the neuron. Every sensory neuron cell has similar receptive fields, and their fields are overlying.

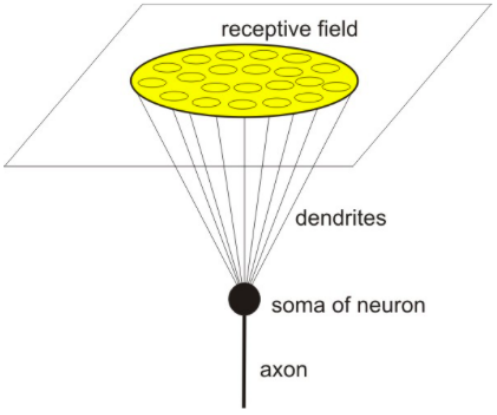


Figure 1: Receptive field of the retina

Further, the concept of hierarchy plays a significant role in the brain. Information is stored in sequences of patterns, in sequential order. The neocortex, which is the outermost layer of the brain, stores information hierarchically. It is stored in cortical columns, or uniformly organized groupings of neurons in the neocortex.

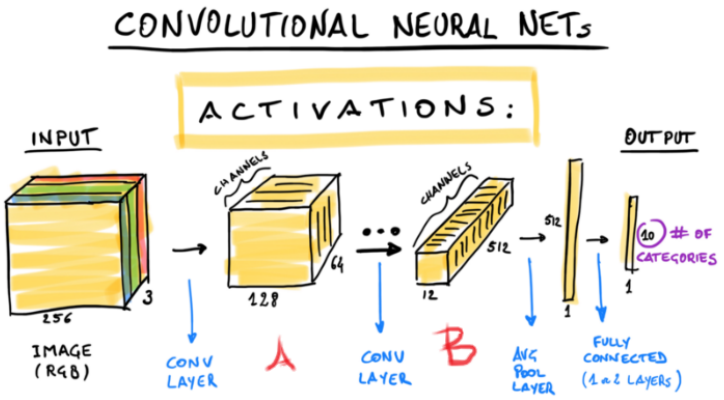


Figure 2: General structure of the convolutional networks

Convolutional Neural Networks are inspired by the brain[1]. Research in the 1950s and 1960s on the brain of mammals suggested a new model for how mammals visually perceive the world. They showed that cat and monkey visual cortexes include neurons that exclusively respond to neurons in their direct environment.

This is the reasoning behind why Convolutional NNs have a different architecture than regular Neural Networks. Regular Neural Networks transform an input by putting it through a series of hidden layers. Every layer is made up of a set of neurons, where each layer is fully connected to all neurons in the layer before. Finally, there is a last fully-connected layer(the output layer) that represent the predictions. On the other hand, the layers of CNNs are organised in 3 dimensions: width, height and depth [2]. Further, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension.

The operation done between the layers of CNNs is called convolution. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel and computes a convolved feature map that expresses how the shape of one is modified by the other.In the process a kernel slides over the input image (convolution operation) to produce a feature map. The convolution of another kernel, over the same image gives a different feature map so after every convolutional layer the resulting feature map has a depth = to the number of different kernels used. The kernel tries to mimic the way the receptive field of the retina works, by connecting neurons with a specific neighbourhood of pixels of the image. It is important to note that the convolution operation captures the local dependencies in the original image.

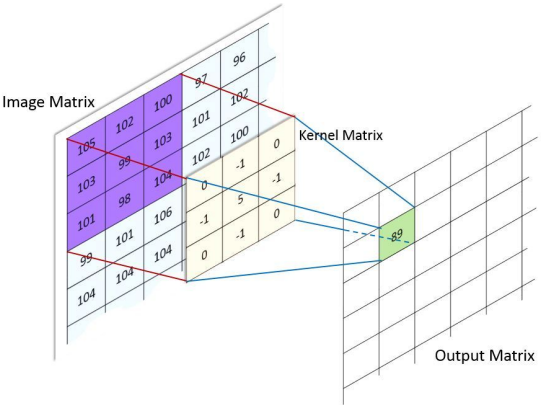


Figure 3: Convolution of a single kernel

Another important operation is the pooling step. Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc. In the case of Max Pooling, an spatial neighborhood is defined (for example, a 2×2 window) and the largest element from the

[1] http://blog.christianperone.com/2017/11/the-effective-receptive-field-on-cnns/
[2] https://bit.ly/2YB9o0H

rectified feature map within that window is selected. Instead of taking the largest element it is possible to take the average of the pixels (Average Pooling) or to sum all elements in that window. However, the Max Pooling has proven to perform the best.

In the Figure **??**, the image is passed to the network in two parts.**Part A** is where the network is learning the features of the objects in the images, that is, where the network will perform a series of convolutions and pooling operations during which the features are detected. Convolutions try to imitate the work of the brain's receptive field. For instance, if the input image would contain a zebra, this is the part where the network would recognize its stripes, two ears, and four legs.

In **part B** is where the classification takes place. The fully connected layers will serve as a classifier on top of these extracted features. They will assign a probability for the object on the image being what the algorithm predicts it is.

3 Design a feed forward Neural Network to classify signs

In this first part of the assignment it was required to put into practice the knowledge to optimize the parameters and the architecture of a ffNN for solving a traffic sign image classification problem. The purpose of this part was to get the student familiar with the Google Colab or Google Cloud working environment as well as to let the student face the difficulties of training a ffNN that classifies images in order to highlight the importance of the use of convolutions in Deep Learning.

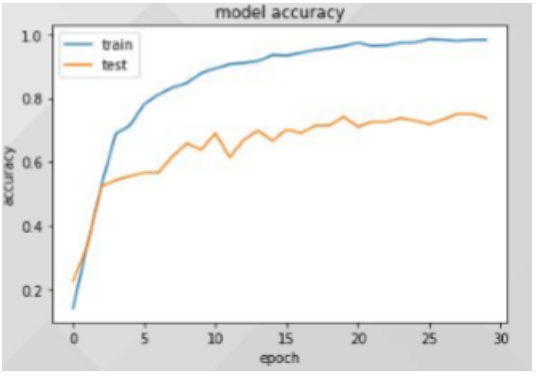
After a lot of trial and error a final network was obtained. It took a long time to train and had a high accuracy for the training dataset but really low accuracy for the test dataset. The model was clearly overfitting. Hence, l2 regularization was included for the hidden layers with the keras parameter "activity_regularizer". The chosen optimizer was SGD with momentum and decaying learning rate. In order to decide when to stop training, and what was the optimal number of epochs, keras EarlyStopping was used to minimize the validation loss. It is important to state that the model took 64 seconds for each epoch to train.

Even though it was possible to improve the results of a ffNN that classifies images, it was guessed during this part of the assignment that other techniques could improve the accuracy and the loss of such the model without taking so much time.

Layer (type)	Output Shape	Param #
dense_60 (Dense)	(None, 224, 224, 8)	32
activation_60 (Activation)	(None, 224, 224, 8)	0
dropout_42 (Dropout)	(None, 224, 224, 8)	0
dense_61 (Dense)	(None, 224, 224, 16)	144
activation_61 (Activation)	(None, 224, 224, 16)	0
dropout_43 (Dropout)	(None, 224, 224, 16)	0
dense_62 (Dense)	(None, 224, 224, 24)	408
flatten_21 (Flatten)	(None, 1204224)	0
activation_62 (Activation)	(None, 1204224)	0
dropout_44 (Dropout)	(None, 1204224)	0
dense_63 (Dense)	(None, 43)	51781675
activation_63 (Activation)	(None, 43)	0

Total params: 51,782,259
Trainable params: 51,782,259
Non-trainable params: 0

(a)



(b)

Figure 4: ffNN structure on the left, evaluation on the right

4 Design a convolutional Neural Network to classify signs

4.1 Initial process

The goal of the second part of the assignment was to improve the previous network by using convolutional layers. For the given notebook after 100 epochs the training accuracy was 99%, while the validation and test accuracy were around 79%.

The process of thinking during this part was divided in two sections:

1. Decrease bias by enlarging the network and adding more parameters and trying several architectures
2. Decrease the variance by tuning the dropout dense layers and trying data augmentation training

For the architecture of the models it was decided to add more convolutional layers. The followed approach was to start with a layer counting with 8 filters, and increasingly adding convolutional layers with more weight kernels, resulting in more feature maps for each level of depth. In order to avoid the exponential growth of the hyperparameters when flattening the feature maps after every CNN layer a Maxpooling2D was added. Finally, after every dense layer a dropout layer was added in order to avoid overfitting. For the lower layers the dropout probability was set higher than for higher layers.

The final network architecture had 5 convolutional layers and 4 dense layers with a VGG-like structure as seen in the Figure **??**.

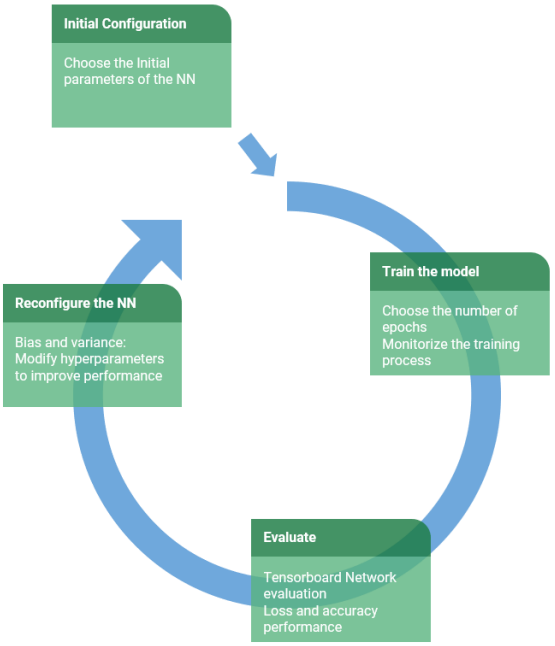


Figure 5: Improving the network

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 224, 8)	224
activation_1 (Activation)	(None, 224, 224, 8)	0
max_pooling2d_1 (MaxPooling2	(None, 56, 56, 8)	0
conv2d_2 (Conv2D)	(None, 56, 56, 16)	1168
activation_2 (Activation)	(None, 56, 56, 16)	0
max_pooling2d_2 (MaxPooling2	(None, 28, 28, 16)	0
conv2d_3 (Conv2D)	(None, 28, 28, 24)	3480
activation_3 (Activation)	(None, 28, 28, 24)	0
conv2d_4 (Conv2D)	(None, 28, 28, 32)	6944
activation_4 (Activation)	(None, 28, 28, 32)	0
conv2d_5 (Conv2D)	(None, 28, 28, 64)	18496
activation_5 (Activation)	(None, 28, 28, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 14, 14, 64)	0

(a)

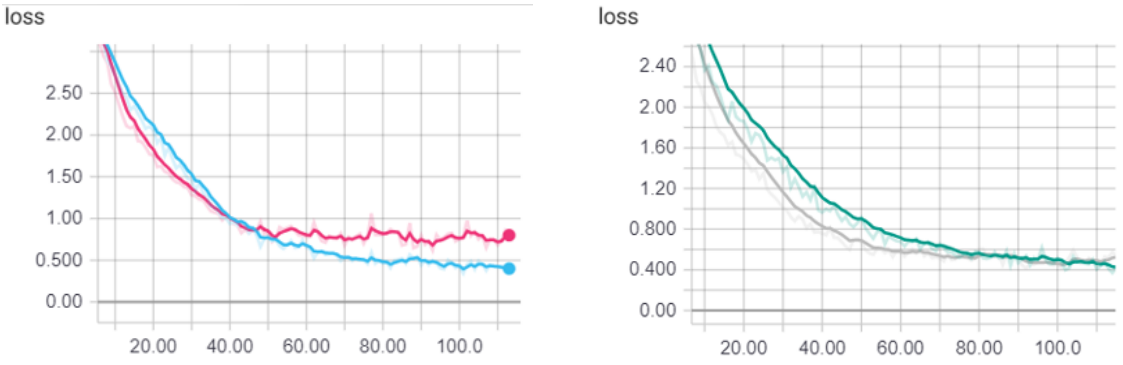
flatten_1 (Flatten)	(None, 12544)	0
dense_1 (Dense)	(None, 1024)	12846080
activation_6 (Activation)	(None, 1024)	0
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
activation_7 (Activation)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 256)	131328
activation_8 (Activation)	(None, 256)	0
dropout_3 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 43)	11051
activation_9 (Activation)	(None, 43)	0

Total params: 13,543,571
Trainable params: 13,543,571

(b)

Figure 6: Conv. layers on the left, dense layers on the right

The next step was to train our model. Therefore, the model was trained as in the first part of the assignment. Besides, data augmentation was introduced during the training process. In order to achieve that the keras ImagedataGenerator was used with the following parameters, $shearrange = 0.2$, $rotationrange = 0.2$ and $channelshiftrange = 0.35$. In that way, ew images with the stated modifications were generated from the original images. Again, an early stopping of 35 epochs of patience was used.



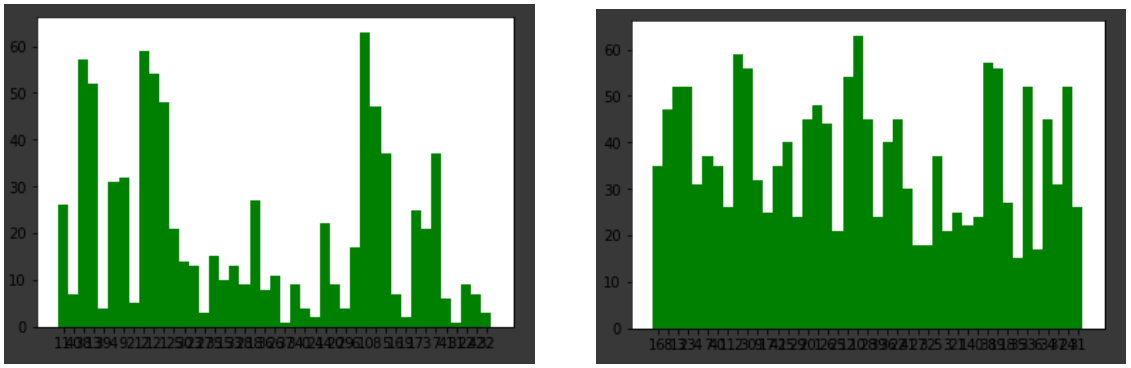
(a) (b)

Figure 7: Model loss before augmentation on the left,after on the right

It is possible to see that the variance was greatly reduced after applying data augmentation. As a final result, 0.94 test accuracy and 0.25 test loss were obtained for CNN model that was predicting in 0.8 seconds. However, during the Easter break some improvements were made on this model to increase accuracy and speed.

4.2 Improvements: Initial data augmentation

After studying the process of improving a deep neural network architecture some data exploration was done. It was found that the classes of the dataset were very unbalanced. For example, some classes had more than 60 instances and others had less than 10. In order to fix this problem a function was created to augment every image on the dataset x number of times depending on the class the image belonged to and the number of instances the corresponding class had. This function was used with the images of classes that had less than 20 instances, and the less instances each class had, the more augmented images were created. In order to do so the images were rotated, zoomed and changed the height and width but also the images that were allowed to be flipped were done. For the record, the signs that indicated mandatory left or right turn and forbidden left or right turn were not flipped.



(a) (b)

Figure 8: Initial dataset on the left, augmented on the right

The final dataset had 1588 instances and was split to 80% as training dataset and 20% as validation dataset. Also the size of the images was reduced by half.

The new dataset was tested on an improved architecture that had the same logic as the one before but less parameters to increase speed even more.

It is worth highlighting that the data augmentation used in the previous model was also used on this part of the training. That means that after adding the augmented images in the initial dataset, they were augmented again in the training part. As seen in the Figure **??**, the final model had $test\_loss = 0.2$, $Accuracy = 0.964$, better scores than the previous ones and $Prediction\_time = 0.07$, which was ten times faster than the previous one. A similar model was used in the final project, for the multi-class classification stage.

```
Epoch 00286: early stopping
Training time:964.0309464931488
CNN took 0.07396769523620605 seconds
Test loss: 0.20234047516407416 - Accuracy: 0.96398891966759
```

Figure 9: Final CNN model evaluation

5 Transfer learning

The goal of this section was to improve the previous results by using transfer learning from pre-trained models. This meant that it was needed to retrain an already existing model that had been previously trained on the ImageNet dataset. The idea behind this process called "transfer learning" is to update the weights of some of the final layers of the model so the model can classify images of a different dataset. With transfer learning it is possible to overcome the isolated learning paradigm and utilize the knowledge acquired for one task to solve related ones.

When applying transfer learning to an specific problem the process follows the next steps.

1. **Select Source Model:** A pre-trained source model is chosen from available models. In this case ResNet50, InceptionV3 and VGG were selected for testing.
2. **Reuse Model:** The pre-trained model can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.
3. **Tune Model:** Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

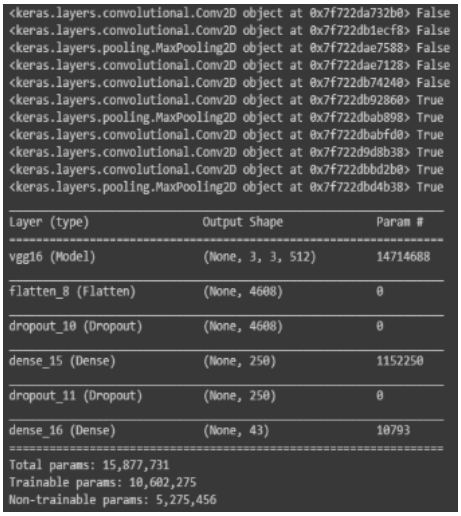
5.1 VGG network

This architecture is founded in the VGG group in Oxford. It supposes an improvement over AlexNet (first deep networks to push ImageNet Classification accuracy to top levels) by replacing large kernel-sized filters(11 and 5 in the first and second convolutional layer, respectively) with multiple 3X3 kernel-sized filters one after another. With a given receptive field (the effective area size of input image on which output depends), multiple stacked smaller size kernel is better than the one with a larger size kernel because multiple non-linear layers increase the depth of the network which enables it to learn more complex features with a lower cost.

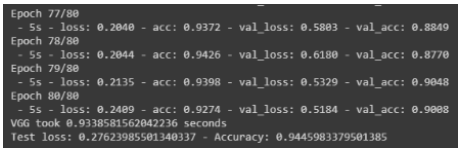
The VGG convolutional layers are followed by 3 fully connected layers. The width of the network starts at a small value of 64 and increases by a factor of 2 after every sub-sampling/pooling layer. It achieved accuracy of 92.3% top-5 on ImageNet[3].

In the assignment, VGG's top fully connected layers were removed and replaced by 2 fully connected layers that fitted the dataset classes. Dropout layers were added to enhance the generalization of the network. To achieve the goal of transfer learning all the network's layers except the last 6 were frozen. In the training process data augmentation was introduced as explained in Part 2. The obtained results are explained in the Figure 10.

[3] https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/



(a)



(b)

Figure 10: VGG adapted structure and the results.The frozen layers can be seen on the left image at the top and the trainable parameters at the bottom.

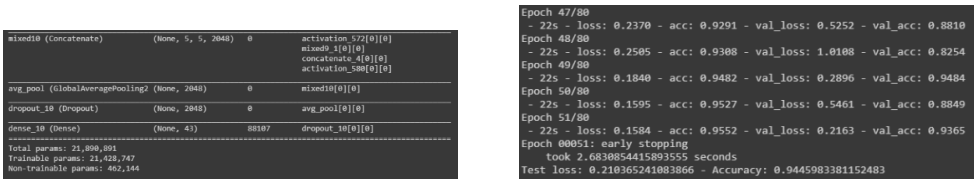
5.2 Inception network

While VGG achieves a great accuracy on ImageNet dataset, its deployment on most the average GPUs is a problem due to the huge computational requirements, both in terms of memory and time. Hence, it becomes inefficient because of the large width of convolutional layers.

Also, important parts in any image may have extremely large variation in size. Because of this huge variation in the location of the information, choosing the right kernel size for the convolution operation becomes a key task. A larger kernel is preferred for information that is distributed more globally. On the other hand, a smaller kernel is preferred for information that is distributed more locally.

In order to address the problems stated above the authors of the GoogleNet / Inception proposed a model which had multiple kernel sizes on the same levels, more specifically 3x3 and 5x5 kernel sizes. To reduce the number of computations, the authors limited the number of input channels by adding an extra 1x1 convolution before the 3x3 and 5x5 convolutions. Besides, they replaced the fully-connected layers at the end with a simple global average pooling which averages out the channel values across the 2D feature map, after the last convolutional layer. As a result, it was able to achieve 93.3%, that is top-5 accuracy on ImageNet being much faster than VGG.

For the purpose of the assignment an InceptionV3 was tested. In order to achieve valuable results in the transfer learning assignment very few layers (only 462.144 non trainable parameters out of 21.890.891 in total) had to be frozen. The final network in the Figure 11 was rather slow due to that reason.



(a) (b)

Figure 11: Inception adapted structure and the results.

5.3 ResNet

The problem with increased depth is that the signal required to change the weights, which comes from the end of the network by comparing ground-truth and prediction becomes very small at the earlier layers. That means that earlier layers are almost unable to learn. This problem is called **vanishing gradient**. The second problem with training deeper networks is to perform the optimization with a lot of parameters.Therefore by adding more layers the number of parameters increases causing a training error to rise. This is called **degradation** problem.

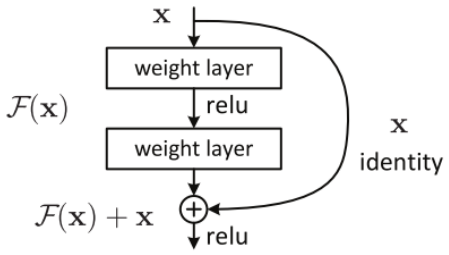


Figure 12: Core idea of Resnet model

The core idea of ResNet is to introduce a so-called "identity shortcut connection" that skips one or more layers, as shown in the Figure 12.

The model introduces a direct path between the input and output to the module implying an identity mapping so the next layer just need to learn the features on top of an already available input. Since the next layer is learning only the residual, the whole module is called residual module.

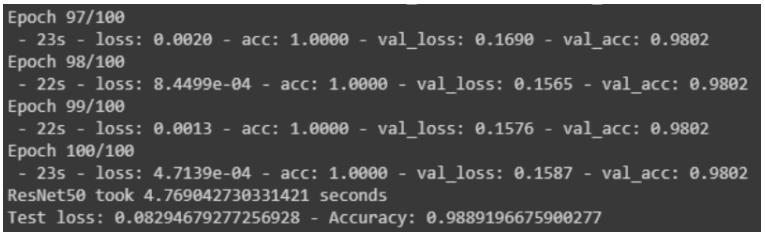


Figure 13: Results of ResNet50

A ResNet50 was tested by training the whole model's structure. The results can be seen in the Figure 13. The network achieved the greater accuracy we have seen so far but it was rather slow mostly because no layer was frozen [4].

[4] `https://bit.ly/2mWCUw1`

6 Traffic signs detector and classifier system

This chapter corresponds to the final assignment of the "Computer Vision" topic. Throughout, the development of this final phase some of the techniques studied during the previous weeks were used.

This section is structured as follows. First, the problematic of the computer vision issue is explained. Moreover, the different challenges to face are presented. Afterwards, the chosen approach to address the problem is stated. The selected approach drives to the design of a system that is explained in different sections matching with the different phases of the strategy followed. Finally, in the conclusion the results obtained are presented. Besides, the opinion of the authors is presented with respect to which techniques resulted more effective in the development of the project.

6.1 Problem and challenges

Some computer vision problems might imply to classify an input image where an object is clearly given to be classified. However, in most of the real applications the input is an image where many objects are present. Thus, to the classification task is summed the object detection problematic. In the case under study, the input are images of roads where many objects are present such as, cars, traffic lights, traffic signs, etc. In this terms, the goal of the project is divided in two different targets. First, given a set of input images, detect which objects within the image are traffic signs, and second, given the detected traffic signs classify them among the different traffic signs classes.

At a first glance, many challenges are related to the development of the different parts of the system. For instance, what is the correct approach to detect objects in an image?, how are the object recognized within an image?, how the dataset of the traffic sign detector should be built?, how to maximize the mAP?, how to propose only once each signal in the image?, etc.

The previously stated problematic and challenges are faced with the use of different techniques that will be explained in the following section.

6.2 Traffic signs detection and classification approach

The chosen approach to implement a system capable of detecting and classifying is divided in several stages. First, a Region Proposal Network (RPN) is built, and given an input image, it proposes a set of boxes containing a detected object in the image. Then, a binary classifier model is built to correctly detect which of these boxes actually contain traffic signs. For this purpose, it is necessary to construct a dataset to train the binary model. Afterwards, a multi-class model is designed to correctly classify traffic signs as belonging to one of the 43 traffic sign classes.

With the shake of following a clear structure and architecture for the system, the code was divided into 6 different notebooks. Thus, for each notebook the different datasets and weights generated were saved as *.pkl* files and the models and weights were saved as *.json* and *.h5* respectively in the working google drive folder. Then, the required files for each notebook were loaded from this folder in the beginning of each notebook. This strategy permitted to save RAM memory from the google Colab session. The 6 different notebooks are listed and briefly explained below.

1. **Data loading:** Read and store initial dataset.
2. **Region proposal network:** Build, train and store the RPN model and the box proposals.
3. **Binary dataset:** Build the dataset in order to train the binary model.
4. **Binary model:** Build, train and store the binary model.
5. **Classifier model:**Build, train and store the classifier model.
6. **Complete integration:** Integrate all the previous models and use them to detect and classify traffic signs in images.

It is worth highlighting that for every Colab notebook the working directory of the google drive was mounted in the Colab session to be able to load te working directory and hence, the generated files.

6.3 Data loading

In this first section the signs dataset *FullIJCNN2013* were downloaded. Successively, the dataset was divided into a training set and a test set. Then, by using the function *readImages* the corresponding data was stored in the respective variables presented below.

- Training data: *train_images, train_files, train_signs, train_bboxes, train_labels.*
- Test data: *test_images, test_files, test_signs, test_bboxes, test_labels.*

Afterwards, the training data was shuffled. Besides, the files *train_signs* and *train_labels* which contain the signs images information and the signs labels respectively were divided into a training set and a validation set. The normalization of the images was performed by dividing all the pixels by 255 in order to have every pixel value in the range between 0 and 1. Finally, the labels of the signs were one-hot encoded by using the keras function *utils.to_categorical()*.

6.4 Region proposal network

In this module a short description of the use of the Region proposal network build by the professor is made along with the decided hyperparameters and our extensions on the model.

6.4.1 RPN Hyperparameters

The present model had 3 parameters to tune :

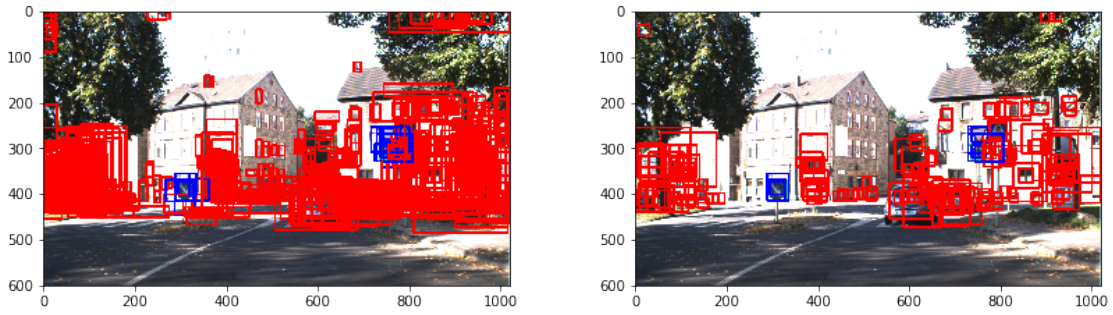
1. TEST_PRE_NMS_TOPK was set to 8.000 because we wanted a lot of regions produced before the non maximum suppression
2. TEST_POST_NMS_TOPK was set to 800 to limit the final images after the nms is applied
3. PROPOSAL_NMS_THRESH was set to 0.6. The larger the threshold is, the less confident proposals are less likely to be suppressed. This leads to larger number of false-positives and hence drop precision.

6.4.2 Boxes filters

In the process of analyzing the proposed boxes by the RPN model with respect to the ground truth boxes, it was found that most of the proposed boxes were very dissimilar to the ground truth boxes. The goal of the RPN was to propose boxes that may contain traffic signs. However, the boxes proposed by the RPN model included boxes with very dissimilar shapes, ranging from rectangles with one side much longer than the other to boxes very similar to squares. The RPN model also proposed boxes with a wide range of areas. By analyzing the ground truth boxes it was found that the larger box had an area close to 12000 pixels and that most of the boxes had a sides ratio above 0.8. In that way, the boxes of interest were the ones with a similar shape and area to the boxes containing the ground truth traffic signs. Then, it was considered convenient to drop boxes that were not likely to contain traffic signs. This lead to later construct a lighter binary dataset and to optimize the computing time of the final application. Thus, two filters were created.

- Ratio filter: The designed ratio filter took as input a list of boxes and returned a new list containing the filtered boxes. The filter calculates the ratio of the sides of each box, and if it was lower than a threshold, then this box was not appended to the list. Otherwise, it was

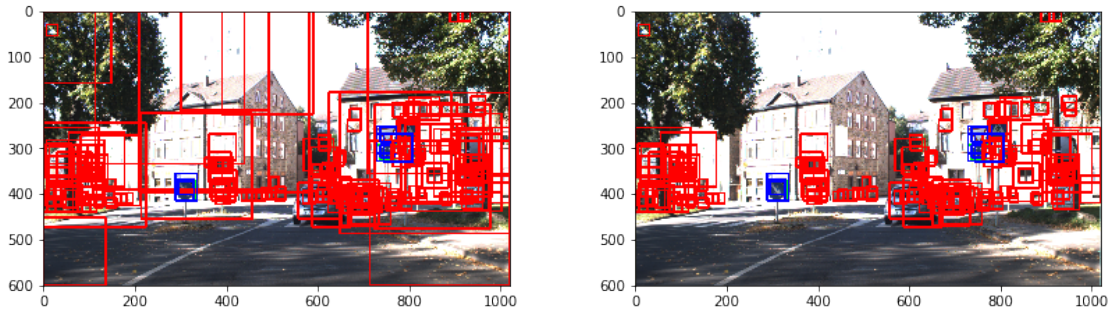
included in the new list. With the aim to set a suitable ratio threshold, the sides ratios of the ground truth boxes were calculated. Then, it was found that most of the ground truth boxes fulfill a 0.8 ratio, and hence this value was used to filter boxes by ratio throughout the implementation. The Figure 14 presents the boxes proposed for the same image before and after applying the ratio filter.



(a) (b)

Figure 14: Before and after applying ratio filter

- Area filter: Similarly to the ratio filter, the area filter took as input a list of boxes and returned a list containing the filtered boxes. The function calculated the area of each box. Then, boxes with an area greater than an area threshold were dropped. In order to establish a convenient value for the threshold, the areas of all the ground truth boxes were calculated. The biggest area of a ground truth box in the training dataset was close to 12000 pixels. In that way, the selected threshold used along the implementation was 15000 to have a margin of confidence. The Figure 15 presents the boxes proposed for the same image before and after applying the area filter.



(a) (b)

Figure 15: Before and after applying size filter

6.5 Binary dataset

The aim of this stage was to build a dataset to later train a binary classifier in order to distinguish traffic signals from other objects. In the process of building a model there is always the need to count with a labeled training dataset.

The binary dataset needed to be formed by both positive samples representing the true traffic signals labeled as “1” and by negative samples representing other objects and labeled as “0”.

For the positive samples the ground truth boxes containing the traffic signs of each image were collected. On the other hand, the challenge was to collect the negative samples. At this point many approaches appeared, such as selecting random boxes in each image as negative samples. Nonetheless, in most of the approaches there was the problem of having actual traffic signs as negative samples, which would have meant to incorrectly train the binary model. Another issue to take into consideration was the size of the dataset. In this regarding the limitation was set by the Colab RAM memory of 15 GB.

Thus, a new approach for gathering the negative samples was proposed. The aim of this approach was to tackle the challenges stated above. The proposed strategy implied the next steps.

1. . In order to gather negative samples, first, the proposed boxes were filtered by their size and ratio. The threshold area was set to 15000, which meant that all the boxes with a greater area were dropped. Furthermore, boxes that did not fulfill the threshold ratio of 0.8 were dropped as well.

2. Once the proposed boxes for each image were filtered, the next step was to compute the Intersection Over Unit (IOU) among the ground truth boxes and the proposed boxes. The IOU threshold was set to the value 0.1. In that way, it was avoided to include traffic signs within a negative sample box with a margin of confidence. The result of this computation for each image, was an IOU matrix with columns representing ground truth boxes and rows proposed boxes. Then, for every row if it was found any IOU higher than the 0.1 threshold the box was dropped as it was matching any of the ground truth boxes. Otherwise, the box was included as a negative sample in the binary dataset. The Figure 16 displays the ground truth boxes as green boxes, the boxes included as negatives samples in the dataset were drawn in blue and the boxes with an IOU higher than 0.1 with any of the ground truth boxes were drawn in red.

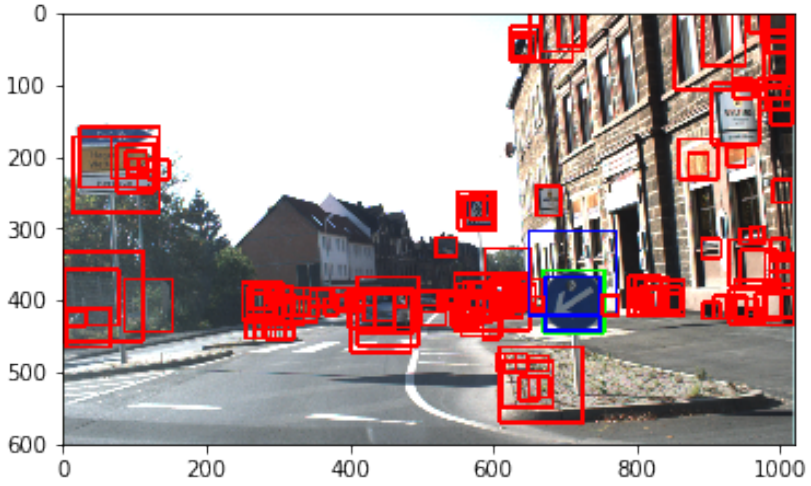


Figure 16: Proposed boxes for training dataset

After the positive and negative samples were collected, it was found that positives and negatives were very unbalanced counting with nearly 800 and 1700 samples respectively. The technique used in this approach to address this problem was data augmentation. The positive samples were augmented by applying shearing, rotation and channel shifts to the original ground truth traffic signs.

The final step was to one hot encode the corresponding sample labels in order to provide them to the neural network that is presented in the following section.

6.6 Binary model

The aim of this section was to build and to train a neural network to classify input images between traffics signs represented by "1" and other objects (not traffic signs) represented by a "0".

The binary dataset generated previously was split into a training set (70%), a validation set (15%) and a test set (15%). The architecture of the neural network was designed similarly to a VGG model. First, a set of convolutional layers were added. The first layers started with few filters but with greater size for the feature maps. As going deeper in the convolutional layers the number of filters was incremented, meanwhile the size of of the feature maps was reduced by MaxPooling layers. The final architecture of the binary classifier neural network is presented in the Figure 17.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 64, 64, 4)	112
activation_1 (Activation)	(None, 64, 64, 4)	0
conv2d_2 (Conv2D)	(None, 64, 64, 8)	296
activation_2 (Activation)	(None, 64, 64, 8)	0
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 8)	0
conv2d_3 (Conv2D)	(None, 32, 32, 16)	1168
activation_3 (Activation)	(None, 32, 32, 16)	0
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 16)	0
conv2d_4 (Conv2D)	(None, 8, 8, 24)	3480
activation_4 (Activation)	(None, 8, 8, 24)	0
conv2d_5 (Conv2D)	(None, 8, 8, 32)	6944
activation_5 (Activation)	(None, 8, 8, 32)	0
max_pooling2d_3 (MaxPooling2	(None, 4, 4, 32)	0
flatten_1 (Flatten)	(None, 512)	0
dense_1 (Dense)	(None, 1024)	525312
activation_6 (Activation)	(None, 1024)	0
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
activation_7 (Activation)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 256)	131328
activation_8 (Activation)	(None, 256)	0
dropout_3 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 128)	32896
activation_9 (Activation)	(None, 128)	0
dropout_4 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 2)	258
activation_10 (Activation)	(None, 2)	0

Total params: 1,226,594
Trainable params: 1,226,594
Non-trainable params: 0

Figure 17: Architecture of binary model

Successively, the model was trained using early stopping over the validation loss with a patience of 15 epochs. Hence, the model stopped training in the 35 epoch when the validation loss did not decrease for 15 epochs. The final *training accuracy* was 0.997 and the *training loss* 0.011, the training time was 193.69 seconds. The performance of the binary classifier on the test set was of 0.990 for the *test accuracy* and of 0.080 for the *test loss*. Finally, the binary classifier took 0.231 seconds to classify the test set.

As a final step the model architecture was saved as a *.json* file in the project folder. In the same way, the model weights were stored in the same folder as *.h5* file.

6.7 Classifier model

In this section the classifier presented in Part 2 was adapted to fit the final integration code. The input size of the images was resized to $64x64$ and for that reason the structure was updated in order to maintain the accuracy and improve the speed even more. The new structure is shown in the Figure 18

Layer (type)	Output Shape	Param #
conv2d_41 (Conv2D)	(None, 64, 64, 8)	224
activation_65 (Activation)	(None, 64, 64, 8)	0
conv2d_42 (Conv2D)	(None, 32, 32, 16)	1168
activation_66 (Activation)	(None, 32, 32, 16)	0
conv2d_43 (Conv2D)	(None, 16, 16, 32)	4640
activation_67 (Activation)	(None, 16, 16, 32)	0
conv2d_44 (Conv2D)	(None, 16, 16, 32)	9248
activation_68 (Activation)	(None, 16, 16, 32)	0
conv2d_45 (Conv2D)	(None, 8, 8, 64)	18496
activation_69 (Activation)	(None, 8, 8, 64)	0
conv2d_46 (Conv2D)	(None, 8, 8, 64)	36928
activation_70 (Activation)	(None, 8, 8, 64)	0

(a)

flatten_7 (Flatten)	(None, 4096)	0
dense_25 (Dense)	(None, 512)	2097664
activation_71 (Activation)	(None, 512)	0
dropout_19 (Dropout)	(None, 512)	0
dense_26 (Dense)	(None, 256)	131328
activation_72 (Activation)	(None, 256)	0
dropout_20 (Dropout)	(None, 256)	0
dense_27 (Dense)	(None, 128)	32896
activation_73 (Activation)	(None, 128)	0
dropout_21 (Dropout)	(None, 128)	0
dense_28 (Dense)	(None, 43)	5547
activation_74 (Activation)	(None, 43)	0

Total params: 2,338,139
Trainable params: 2,338,139
Non-trainable params: 0

(b)

Figure 18: Conv. layers on the left, dense layers on the right

The results of the training of the adapted model can be seen in the Figure 19.

```
Epoch 00100: early stopping
Training time:174.09247088432312
CNN took 0.04132723808288574 seconds
Test loss: 0.18848230350256956 - Accuracy: 0.961218836565097
```

Figure 19: Results of final Classifier model

It is possible to analyze that the model doubled in speed and improved the test loss results and remained the same in Accuracy. The last step in this section was to save the model in a *.json* format and the weights in a *.h5* file to load them in the final part, as it was done with the other models.

6.8 Complete Integration

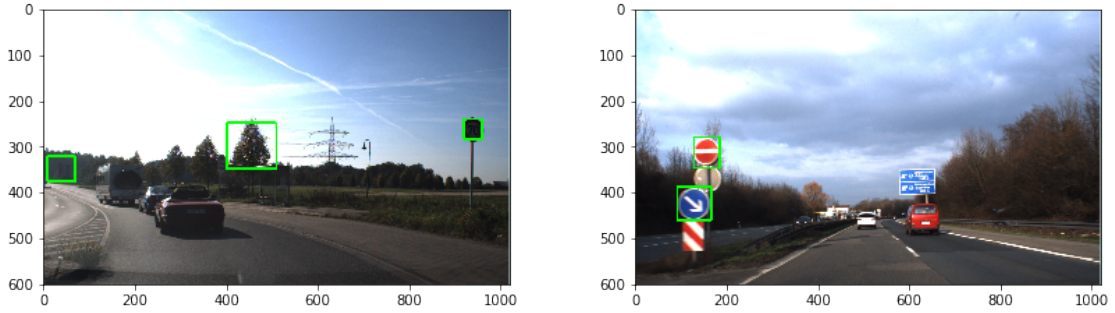
The final stage of the project was to integrate all the previous parts into a final system that is able to detect and classify traffic signs within images. With this goal, all the required test files were loaded, as well as the binary classifier model and the multiple-class classifier model.

It is worth highlighting that it was needed delete the example files in the folders of the cloned repository to calculate the actual mAP of the implemented traffics sign detection system.

First, the ground truth boxes were extracted from the test set and they were stored in the *ground-truth* folder. Afterwards, the for each image the proposed boxes by the RPN are filtered with the sides ratio threshold of 0.8 and with the area threshold of 15000 as stated previously. By doing this the amount of boxes to be predicted as traffic signs or not were reduced in great extent reducing the computation time of the system.

Following the pipeline of the final integration, every proposed box after the filtering is given as an input to the binary model. If a box is predicted to be a traffic signal then the prediction confidence provided by the model is stored in a file. Successively, the image box is introduced in the multi-class neural network that outputs the traffic signal label, which is stored in the same file. Finally, the box coordinates are saved in the same file that is used to compute the AP for each class and the final mAP.

With the aim of better visualizing the results all the boxes classified as traffic signals were plotted in the images. The Figure 20 displays two images of with the corresponding boxes classified as traffics signs. In the first image the system did not perform perfectly. On the other hand, in the second image the system is able to detect correctly all the traffic signs.



(a) (b)

Figure 20: Two different results performances

The final performance of the traffic signs detector and classifier system is presented in the Figure 21. The main performance measures that were used to evaluate the system were the processing time and the mAP. The system was able to detect and classify the images in 137 seconds, with a mAP of 42.8 %.

```
39.32% = 1 AP
44.03% = 10 AP
39.68% = 11 AP
53.46% = 12 AP
37.93% = 13 AP
72.52% = 14 AP
36.00% = 15 AP
50.00% = 16 AP
50.00% = 17 AP
56.57% = 18 AP
83.96% = 2 AP
75.00% = 22 AP
50.19% = 23 AP
66.67% = 24 AP
26.00% = 25 AP
42.86% = 26 AP
48.57% = 28 AP
0.00% = 29 AP
42.50% = 3 AP
75.00% = 30 AP
0.00% = 31 AP
6.67% = 32 AP
83.33% = 33 AP
33.33% = 34 AP
55.00% = 35 AP
0.00% = 36 AP
0.00% = 37 AP
42.76% = 38 AP
0.00% = 39 AP
62.16% = 4 AP
0.00% = 40 AP
33.33% = 41 AP
26.01% = 42 AP
67.02% = 5 AP
0.00% = 6 AP
100.00% = 7 AP
48.21% = 8 AP
77.78% = 9 AP
mAP = 42.79%
Traffic sign detection took 137.96943998336792 seconds
```

Figure 21: Evaluation of final detection system

Conclusion

It was found that every stage of the project was very important for the final performance of the system. From the RPN parameters to the binary dataset all the tasks were challenging and provided us with the opportunity to explore many new techniques. It was found a key task to create a suitable binary dataset, and even more to balance the positive samples and the negatives samples. Besides, the classifier model previously built proved essential to correctly classify images once detected.

The project was understandable but challenging at several points. It required collaborative work and ideas, clear understanding of the theory presented in the class and patience. The great difficulty of that problem was understanding its complex concepts in a short period of time while implementing everything at the same time. Nevertheless, Unit 3 was really interesting and enjoyable.

Remarks and Recommendations

Everything presented in this report was the result of a 6-week collaborative work and study deep into the computer vision problems presented in the Unit 3 part of the EIT-DS-Deep Learning course.

It was clear that the structure of the course was carefully designed to fit the students schedule by distributing the work in weekly assignments.This was greatly appreciated as it helped us understand clearly the concepts taught by the professor each week and narrowed down our focus on each specific task accordingly.

Moreover, the students had the freedom to select between 3 different coding environments: their computer, Google Colab and Google Cloud which allowed us to choose the more intimate environment to us and helped us avoid spending too much time on configurations.

One important thing to add would be that the course credits (3 ECTS) is not at all corresponding to the weight of work that we had to do in order to complete the assignment.To be clear, we are not suggesting that there should be less amount of work, but the credits of the course should increase.

In order to help the course improve even better in the future, we would also suggest to increase the number of teaching hours per class from 2 to 3.This extra (optional?) hour could be perceived as a workshop for the students to familiarize them with computer vision concepts in keras and solve questions or errors.It would greatly help the students get pass introductory boundaries and let them focus on deeper problems of the assignment.