Traffic sign detection and classification (with deep learning)

Mafalda Costa - up202109478@fe.up.pt
Maria Baía - up201704951@fe.up.pt
Ricardo Nunes - up201706860@fe.up.pt

Abstract

This report explains how the group detected and classified images based on 4 classes of traffic signs using Neural Networks. The signs detected are traffic lights, stop signs, speed limit and crosswalk signs. We divided the problem into two tasks, Image Classification and Object Detection. First, we explain how we deal with imbalanced data and what data augmentation we have done. After that, we will get into the three different versions we've done for Image Classification and compare the results and, finally, we will compare the results of the two architectures we used for Object Classification.

1 Introduction

Traffic signs detection became useful with the technologies advances and is important for automated driving, for inventory purposes and for driver assistance systems. The goal of this project is to build an application capable of recognizing and classifying traffic signs, using neural networks, detecting a subset of the traffic signs used in Portugal and grouping them into one of the 4 classes: traffic light, stop sign, speed limit sign and crosswalk signs.

We did image classification of the dataset provided, implementing different models based on Convolutional Neural Networks architectures used for classification, considering the 4 classes mentioned above, and then we evaluated the results obtained and compared them.

First step was the preparation of the dataset to apply the Neural Networks techniques and data augmentation.

We initially approached the problem like a multiclass one and classified the image based on the sign with the biggest area. For that, we used the ResNet architecture and compared them when training the dataset from scratch, using pre-trained weights and fine-tuning. After that, we designed a custom architecture and compared the results obtained with the ResNet architecture ones. Finally, we did multilabel classification adapting the models used before.

All models were trained using a batch size of 32, using 2 processes to load the data and 30 training epochs, saving the best model, that is, the model with the lowest validation loss.

For object detection, we implemented a two-stage architecture first, we chose pytorch Faster R-CNN, and after that we trained YOLOv5, a single stage object detection architecture.

All the results on this report were calculated on the test dataset.

2 Dataset

Before we start developing the models, we need to analyse our data.

Firstly, for the multiclass problem, we have the following class distribution:

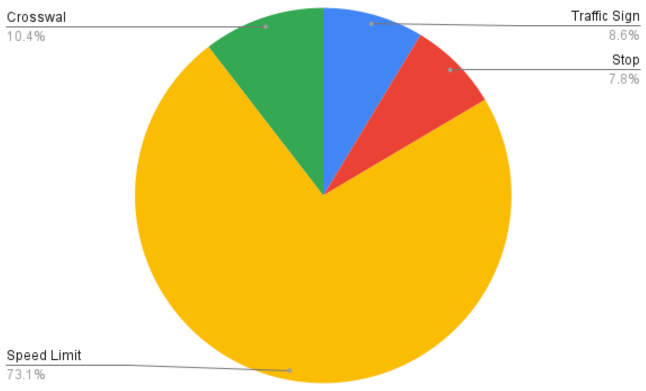


Figure 1 - Class distribution in multi-class problem

As we can see in Figure X, the majority class is *Speed Limit*, with a representation of 73.1% of the dataset. This can be a problem, since our models can learn to classify all images as *Speed Limit* and have a reasonable loss and accuracy. To mitigate this, we introduce weights to the loss, giving more weight to classes with less representation.

For the multi-label problem, we have the following class distribution:

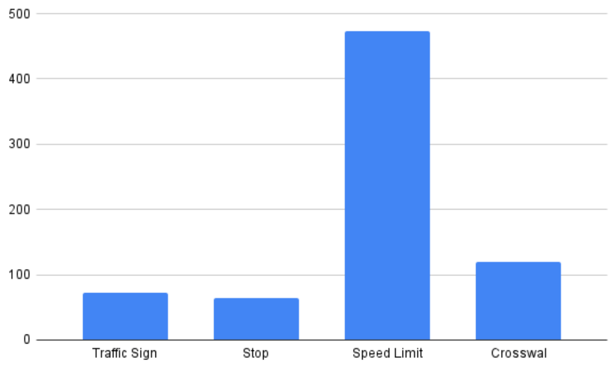


Figure 2 - Class distribution in multi-label problem

We have the same problem of an imbalanced distribution: *Speed Limit* has the most representation in the dataset, by a lot. However, while training the models, we didn't have the same problem as in the multi-class case. So, we decided not to fix the imbalance.

3 Data Augmentation

In order to not be limited to the size of the original dataset, and neither having to collect data from other sources, we perform data augmentation techniques. This phase aims to generate new images from the images given in the dataset to provide a greater volume and variety of data and give a more robust dataset. A set of transformations is applied to the data during the training: the resize with dimension 224x224, the colour jitter to randomly change the brightness to jitter at 0.5, the horizontal flip and the rotation of 20º with a certain probability, the tensor converter and the tensor normalisation.

4 Image Classification

4.1 Basic Version

For the basic version, two different models were developed, a model using ResNet trained from scratch and another model using ResNet with pre-trained weights and fine-tuning.

One of the major problems using Deep Neural Networks is related with the number of the network's layers. The more layers the network has, the deeper and the harder it is to train it - this phenomenon is called the Vanishing Gradient. During back propagation, the neural network learns by updating its weights and biases in order to reduce the loss function. In a network with this problem the weights may not be updated, resulting in the inability to generate models with many layers or models to converge prematurely to bad solutions. For these reasons, it was implemented the Resnet architecture, since residual networks provide residual connections directly to the previous layers. This architecture solves the Vanishing Gradient through shortcuts where layers are skipped, processing in a faster way, leading to shortcuts to change the gradient calculation in each layer. So, it allows stacking Residual blocks more and more, without degradation in network performance and allowing the construction of networks with great depth.

We needed to split the training and test images and notes separately. Subsequently, a validation set corresponding to 15% of the training set was created to provide unbiased evaluation of the model fit of the training set while adjusting the parameters.

We chose Adam optimizer because it is an efficient algorithm and it is simple to implement.

Knowing that fine-tuning consists of adapting a pre-trained model for a task similar to the one we are faced with, adjusting the weights for our task, it is understandable that the results are significantly better than training a model from scratch.

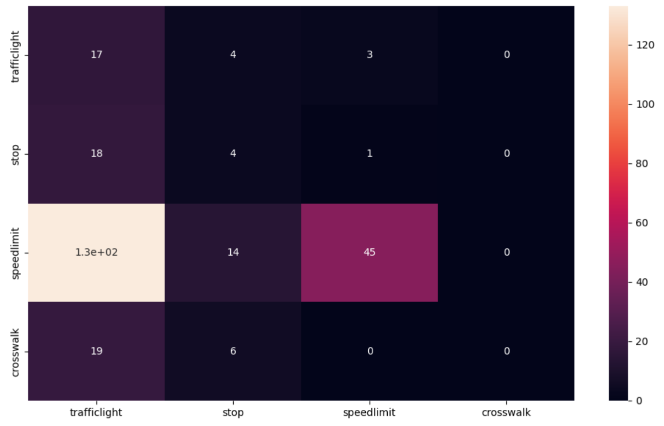


Figure 3 - Confusion Matrix when trained from scratch

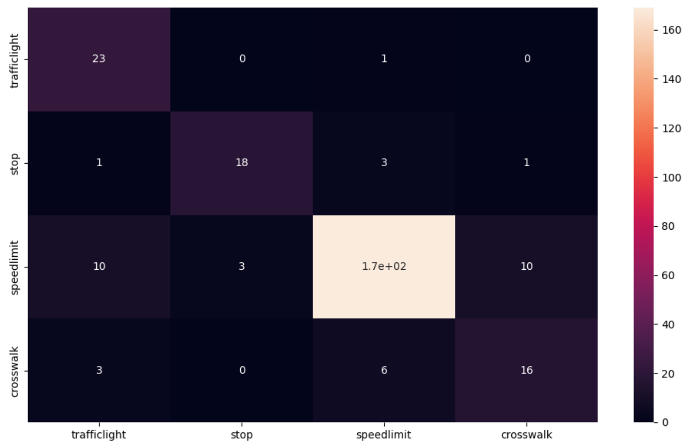


Figure 4 - Confusion Matrix when trained using Fine-tuning

Comparing the confusion matrices, the class distribution of the two models is very different. While the model trained from scratch classifies most of the images as *Traffic Light*, probably because the images containing traffic lights are the first ones the model trains, the fine-tuned model is more accurate to the true distribution of the classes. Micro-Average Precision of the model trained from scratch is 25,29% and from the fine-tuned model was 85,66%, a very significant increase.

Class	Precision - From Scratch	Precision - FineTuned	Recall - From Scratch	Recall- FineTuned
trafficlight	0.71	0.96	0.092	0.62
stop	0.17	0.78	0.14	0.86
speedlimit	0.24	0.88	0.92	0.94
crosswalk	0	0.64	0	0.59

Table 1 - Precision and recall for ResNet Model

Looking at the table we can better understand the performance improvements. Initially, the algorithm was not classifying images as *crosswalk* and best recall overall is for the majority class, as expected.

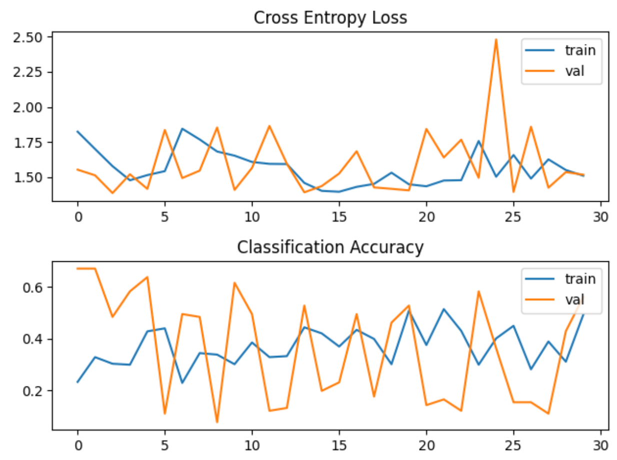


Figure 5 - Loss plot and classification accuracy when trained from scratch

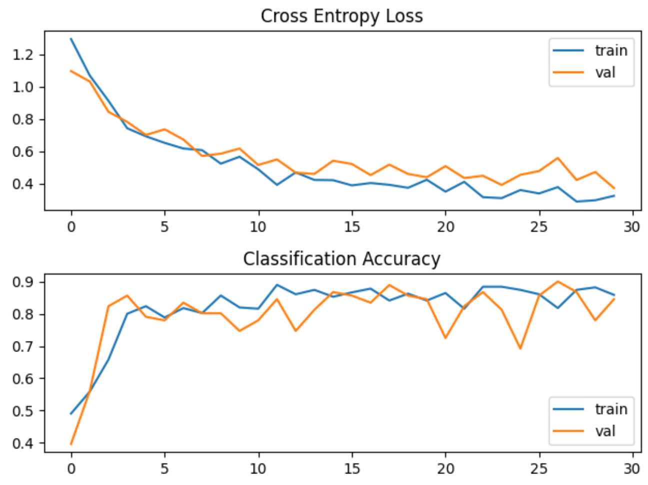


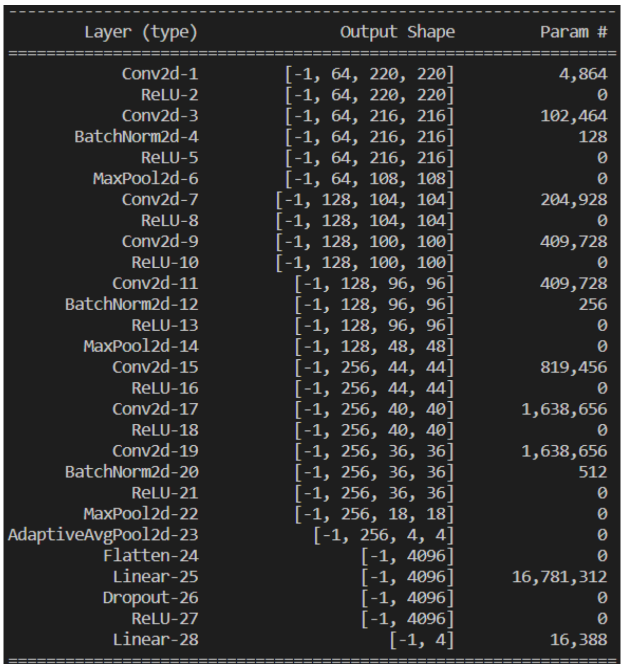
Figure 6 - Loss plot and classification accuracy when trained using fine-tuning

Comparing the performance of the two models, the fine-tuned model behaves much better. In the fine-tuned mode, loss and accuracy for validation and training are increasing and decreasing, respectively, together. This means that the knowledge learned with the training set is reflected on the validation set.

4.2 Intermediate Version

Our network was inspired by the VGG architecture [2], which is a classic architecture in the Convolutional Neural Networks, based on an analysis of how to increase the depth of the networks that preceded it.

Based on this architecture, the model of the intermediate neural network is the following:



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 220, 220]           4,864
              ReLU-2         [-1, 64, 220, 220]               0
            Conv2d-3         [-1, 64, 216, 216]         102,464
       BatchNorm2d-4         [-1, 64, 216, 216]             128
              ReLU-5         [-1, 64, 216, 216]               0
         MaxPool2d-6         [-1, 64, 108, 108]               0
            Conv2d-7        [-1, 128, 104, 104]         204,928
              ReLU-8        [-1, 128, 104, 104]               0
            Conv2d-9        [-1, 128, 100, 100]         409,728
             ReLU-10        [-1, 128, 100, 100]               0
           Conv2d-11          [-1, 128, 96, 96]         409,728
      BatchNorm2d-12          [-1, 128, 96, 96]             256
             ReLU-13          [-1, 128, 96, 96]               0
        MaxPool2d-14          [-1, 128, 48, 48]               0
           Conv2d-15          [-1, 256, 44, 44]         819,456
             ReLU-16          [-1, 256, 44, 44]               0
           Conv2d-17          [-1, 256, 40, 40]       1,638,656
             ReLU-18          [-1, 256, 40, 40]               0
           Conv2d-19          [-1, 256, 36, 36]       1,638,656
      BatchNorm2d-20          [-1, 256, 36, 36]             512
             ReLU-21          [-1, 256, 36, 36]               0
        MaxPool2d-22          [-1, 256, 18, 18]               0
AdaptiveAvgPool2d-23            [-1, 256, 4, 4]               0
          Flatten-24                 [-1, 4096]               0
           Linear-25                 [-1, 4096]      16,781,312
          Dropout-26                 [-1, 4096]               0
             ReLU-27                 [-1, 4096]               0
           Linear-28                    [-1, 4]          16,388
================================================================
```

Figure 7 - Custom model summary

The network structure is composed of four sequential main blocks. The first block constitutes two convolution operations, initially taking 3 input channels and generating 64 channels. The second and third blocks have one more convolution operation: the second block receives 64 input channels from the first block, and generates 128 channels; the third block receives 128 channels from the latter and generates 256 channels. All convolutional operations have kernel size of 7, followed by an ReLU activation function. At the end of each of the first three blocks is added the max pooling operation, with a kernel size of 2, and a batch normalisation to reduce overfitting.

Before the final block an adaptive average pooling layer is added, for the model to not depend on the size of the image input.

The final block, the classifier, consists in four operations: the flattening layer, to convert the 3D tensors into 1D tensors to be fed to the following layer, being the linear layer, which takes in 4096 inputs and outputs also 4096, an dropout with 0.25 probability, an ReLU activation function and to finalise another linear layer which takes in 4096 inputs and outputs 4 (related with our four classes).

Compared to the VGG architecture, our network has fewer convolution layers, due to the fact that our problem is simpler than the problems VGG architecture was designed to solve. Also, as mentioned previously, two layers were added: an adaptive average pooling layer and batch normalisation layer.

After training the model, we obtain the following results:

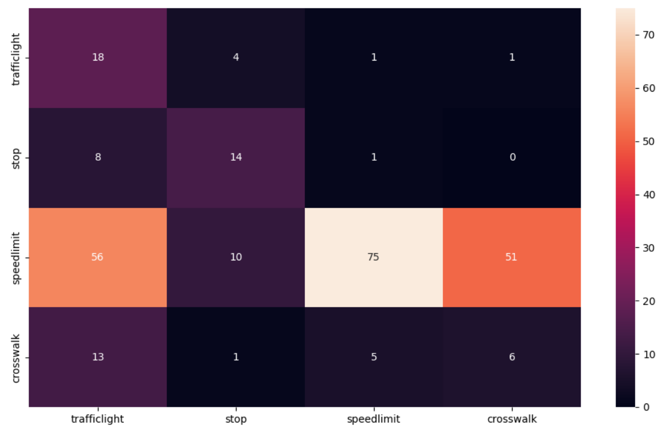


Figure 8 - Confusion Matrix when trained using custom model

Class	Precision	Recall
Traffic Light	0.75	0.19
Stop	0.61	0.48
Speed Light	0.39	0.91
Crosswalk	0.24	0.10

Table 2 - Precision and recall for Custom Model

For the confusion matrix we can conclude that the model is particularly good at classifying *Speed Limit*, to be expected since most images belong to this class. Also, we obtained an accuracy and precision of 71.40% and 49.7%, respectively. We can also see that the model has a high recall and low precision of Speed Light. This can mean that the model is biased towards this class (which is the majority class).

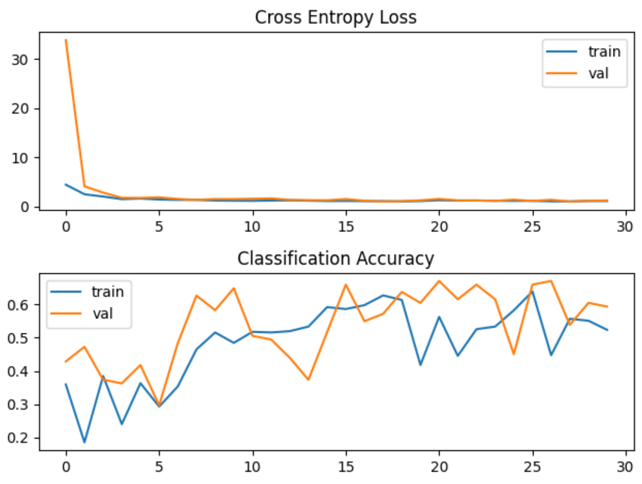


Figure 9 - Loss plot and classification accuracy when trained using custom model

The graphics show us the accuracy and loss of both training and validation sets along 30 epochs of training. As it was expected, the loss both for training and validation decreased over the epochs, which leads us to conclude that the knowledge learned with the training set is reflected in the validation set. Regarding the accuracy, both training and validation sets generally increase with peaks, due to difficulty in converging, caused by the small dataset.

4.3 Advanced Version

For the advanced version of Image Classification, we need to adapt the three models for the multi-label problem. This means that each image will be associated with several labels.

To begin with, we need to adapt the dataset. Instead of returning a single label, it is necessary to return a one-hot list of the labels present in the image. After that, we need to change the loss function. For this problem, we used the BCEWithLogitsLoss [3], which already includes the Sigmoid activation function. By using this type of loss function, we are considering the problem as one-vs-all, or, in other words, we consider four binary problems, one for each class, that considers if the class is present or not.

The results of the multi-label problem are presented below.

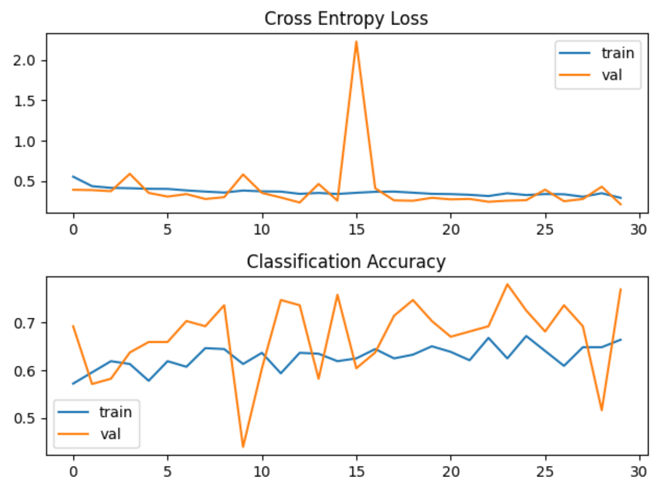


Figure 10 - Training history of ResNet from scratch, multi-label

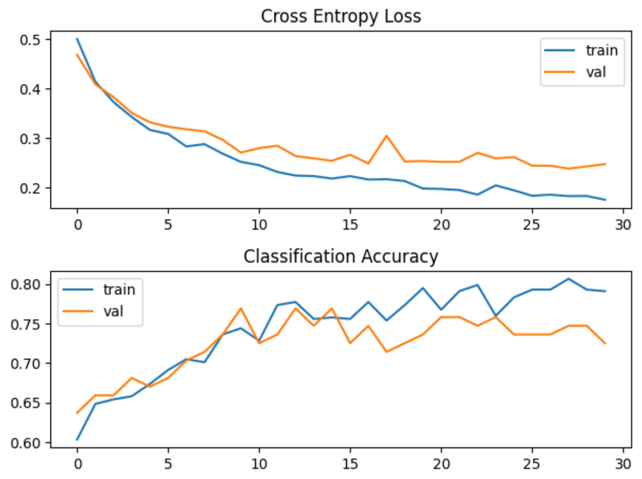


Figure 11 - Training history of ResNet with pre-trained weights and fine-tuning, multi-label

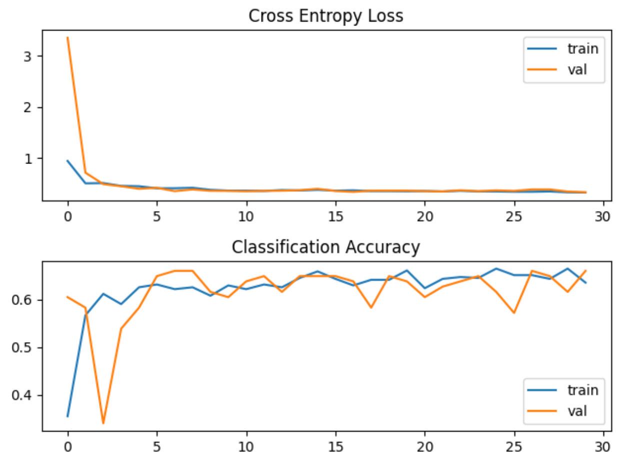


Figure 12 - Training history of Custom Model multi-label

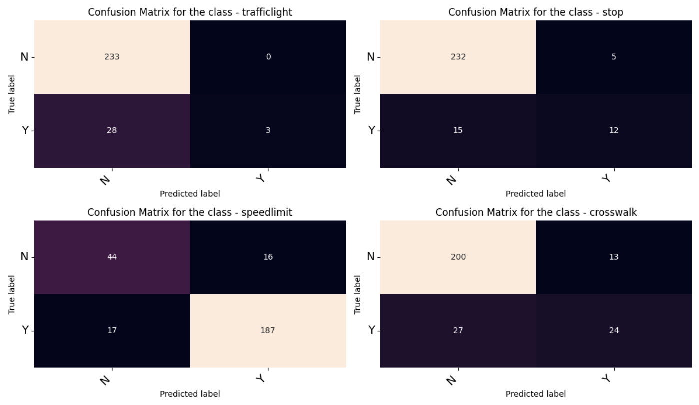


Figure 13 - Confusion matrix of ResNet from scratch, multi-label

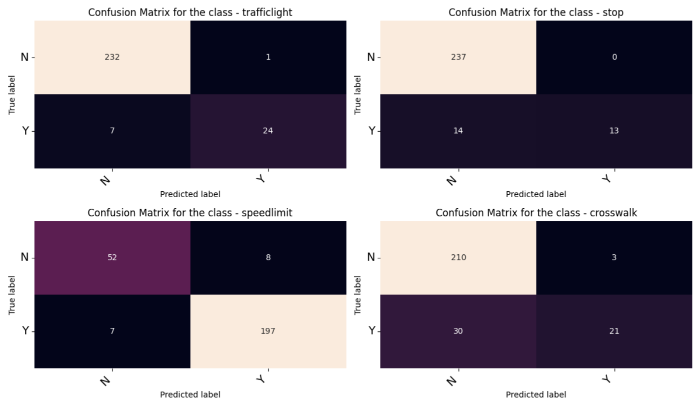


Figure 14 - Confusion matrix of ResNet with pre-trained weights and fine-tuning, multi-label

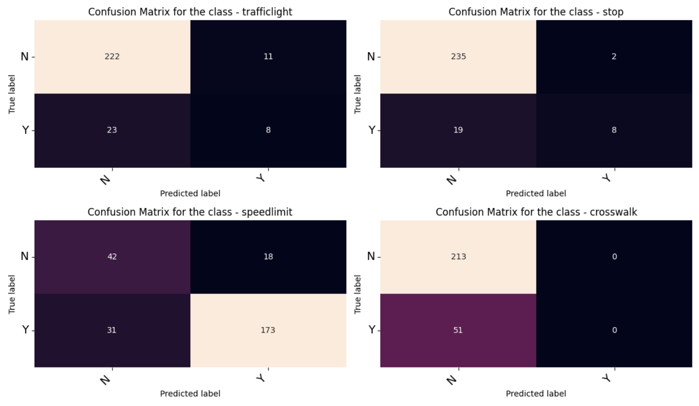


Figure 15 - Confusion matrix of Custom Model multi-label

As expected, the model that performed best was the Resnet with fine tuning. He had a lower performance compared to the multi-class problem, which was expected since it is a more complicated problem.

5 Object Detection

5.1 Basic Version

Object Detection is an advanced version of Image Classification where Neural Networks detect objects on videos or images that belong to predefined classes, using bounding boxes.

In order to do object detection we had to change our dataset structure and format the xml files. We needed the bounding boxes size for every object on the image and that information was on the xml file on the tag *bndbox* so we got that and parsed it into a txt file.

For the two-stage architecture for object detection, we chose the **Faster R-CNN model**, a model that is an improvement of the well-known R-CNN architecture, however, much faster. The first phase of Faster-RCNN is to detect the location of objects implementing a Region

Proposal Network internally trained, and the second phase is to classify the objects present in the image.

We couldn't evaluate our test with this model.

5.2 Advanced Version

YOLO is a one stage detector meaning that makes the localization of the object and computes de class probabilities all at once, using an end-to-end Neural Network.

This is an architectura known for generating good accuracy(when compared to single-stage object detection architectures), for determining a good *Intersection over Union* of the different bounding boxes and for being fast, and for that reason, it was chosen by the group. We trained a YOLO detector to train our dataset and analysed the results obtained.

We choose the most recent version of YOLO (v5). We can see a example of a result below:

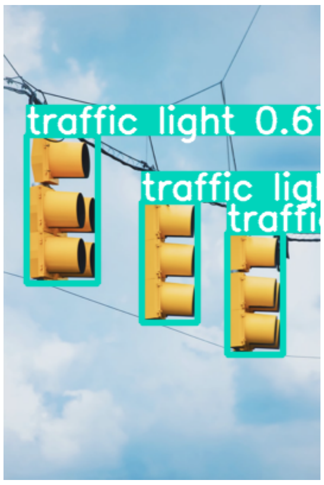


Figure 16 - Example result of YOLOv5

		Faster R-CNN	YOLOv5
	Labels	mAP	mAP
Stop Sign	27	-	0.829
Traffic Light	49	-	0.995
Speed Limit	234	-	0.994
Crosswa lk	58	-	0.945

Table 3 - mAP of object detection models

Just by observing the YOLOv5 results we can conclude that they are very good, getting on MAV over all the classes of 94.1%. This is a normal value since yolo pre-trained the weightson COCO dataset, which is a huge dataset containing 80 classes.

Note: The notebook of object detection using YOLO has different paths than the send code because it had to be tested using Google Colabs, we kept it like that because the objective is to show the results.

6 Conclusion

The development of this project was more computationally demanding and the time was tighter considering the time that Neural Networks models require to be trained. It was interesting to test the various models and observe the evolution of the results. In conclusion and as expected the best model for Image Classification was the Resnet model using fine-tuning for the multi-label problem.

As for object detection, several challenges have arisen and unfortunately, we were not able to obtain the results of the two models to be able to compare.

References

[1] https://viso.ai/deep-learning/vgg-very-deep-convolutional-network/

[2] Pytorch documentation, BCEWithLogitsLoss

[3] Pytorch documentation, ObjectDetectionWithPytorch

[4] YOLO tutorial, ObjectDetectionWithYOLO