Traffic Signifier - Classifying & detecting traffic signs

Ivo Saavedra
up201707093@edu.fe.up.pt
Telmo Baptista
up201806554@edu.fe.up.pt
Tiago Silva
up201806516@edu.fe.up.pt

Faculty of Engineering
University of Porto
Porto, PT

Abstract

In this article we tackle the problem of traffic sign classification and detection using Deep Learning techniques. Various experiments were performed, and we also proposed a custom architecture for the problem. In the first stage we tackled a multi-class problem where we only classified the signal with the highest area in an image. On later stages, we addressed the true purpose of this project, which was to solve a multi-label classification problem of traffic signs. As an extension of the previous task, we also present an approach for traffic sign detection, using a two-stage architecture and a one-stage architecture. All our approaches performed well according to our expectations.

The images used in the experimental phase were retrieved from a Kaggle dataset for road sign detection. *Code available at:* `https://github.com/Telmooo/traffic-signifier`.

1 Introduction

Image classification has multiple applications, from medical image classification for a specific disease detection, to monitoring coral reefs [14]. One of the main problems, usually associated with automatic driving and inventory tracking, is the detection and classification of traffic signs.

Image classification and object detection are two very important techniques for computer vision problems, because they help computer understand real-life objects or environments [2]. We will tackle both these problems throughout this document with Deep Learning Convolutional Neural Networks (CNNs).

CNNs have become a widely used machine learning approach for object recognition tasks [5]. They are comprised of multiple node layers, with an input layer, one or more hidden layers and an output layer. Nodes connect to other nodes and have associated weights and thresholds. If the output of the layer is higher than the threshold, then that node is activated and the data will be fed forward to the subsequent layers [6].

Throughout, this documents we will explain the steps that were taken as well as the models that were implemented/adapted to reach the proposed goals.

2 Dataset

To better suit the needs of our models and the characteristics of the chosen dataset [9], we created a custom Dataset by making use of PyTorch's Dataset class [11]. In this class we converted the image labels, bounding boxes, areas and the classes to the required format for the specific classification task. We also defined a set of transformations, which we will go into further detail in the following section.

2.1 Data Augmentation

The chosen dataset was lacking in terms of training data size. To tackle this problem and to better generalize each of the models, we applied data augmentations to each image with Python's Albumentations [1] library. This process is done with Albumentation's *Compose* function where small transformations (like rotations, mirroring, translations etc.) can be applied on the target areas (bounding box areas) of the training dataset. We will now explain each of the chosen transformation techniques, taking into consideration the characteristics of the dataset.

For the initial transform, there is a choice between a *Posterize* transform and *CLAHE*, with probabilities of 0.3 and 0.25, respectively. The first reduces the number of bits in each colour channel, and was chosen as a way of possibly reducing the colour gradients in each of the signs of the images' low definition. The other was chosen to increase the images' contrast by a very small amount, to reduce some weather effects of the image.

Next, the *RandomBrightnessContrast* transform is applied with a probability of 0.25, which reduces/increases the brightness of the image by small amounts. This is done to simulate different lighting conditions on the given images, for example, to get a larger amount of "night" images so our model can generalize better in those lighting conditions, as the images in the dataset are overwhelmingly during day-time.

The third augmentation is the *HorizontalFlip* with a probability of 0.4, which, like the name says, flips the image on horizontal axis. This transform was more aimed at the traffic lights, as traffic signs usually aren't flipped horizontally.

After this, there is a choice between a *Rotate* and *Perspective* transforms with equal probabilities. The first transform applies a small rotation of up to 30º on the image, and the second changes the perspective of the image by performing a random four point transform of the input bounding box. These were chosen, to better generalize the models to signs with sightly different perspectives and angles.

The next transformation as two possibilities, *Blur* and *MotionBlur* with probabilities of 10% and 20%, respectively. The former is meant to simulate blurred images, and the latter to mimic the effects of motion blur on an image, as both are relevant in our problem. Blurred images are bound to happen in a camera inside a car.

Due to some images having unusual weather effects which severely reduce the image's colours, we also applied a greyscale conversion with a probability of 5%, this helps the model also generalize better as it gives it a necessity to learn the shapes and not only rely on colour.

Finally, in the last steps we padded the images that were too small with the *PadIfNeeded* transform; then a random crop of the input with *RandomCrop*, to simulate clipped signs and also match the input images that were trained on our chosen models; and lastly, we normalized the inputs with ImageNet's normalization parameters.

This data augmentation takes place only during the training phase. The testing images are kept as they are, only being normalized.

3 Image Classification

3.1 Multi-class Classification

For this stage, and as was suggested by the assignment, we only trained the model to classify the largest traffic sign in a given image. We used as a base model the DenseNet201 [5] extracted from *torchvision* [12] and replacing the classifier layer with a custom classifier with the number of output classes.

For the optimizer, the best results were obtained with the *Adam* optimizer with a learning rate of 1e-3 and a weight decay of 5e-4. We also applied the *ExponentialLR* scheduler, which exponentially decays the learning rate with the number of epochs by a factor of 0.9.

The output of the model is a tensor with the outputs of each image in the batch with the weights associated with each class. To obtain the loss of a batch, we pass this tensor through the Cross Entropy Loss function. The probabilities are calculated by passing the prediction tensor through a *softmax* activation function, which normalizes the values to the probability of 1.

To measure the performance of each iteration of the model, we made use of the Accuracy function from *torchmetrics*, with micro averaging (to calculate the metric globally across all samples and classes).

3.1.1 Transfer Learning

Using the Densenet201[5] with pretrained weights, we have access to a feature extractor that is already competent in its task. So to now train the model to our task, we trained the model in two stages. In the first stage, *freeze learning*, we froze the feature extractor layers, as to only train the classifier head. In this stage, we trained the model for 30 epochs with the optimizer described above. Afterwards, with the objective to tune the feature extractor specifically to extract traffic signs features, we did a second stage of training, where we unfroze the feature extractor layers and trained the model for another 15 epochs, starting from the best model obtained in the previous stage, with a smaller initial learning rate of 1e-5.

With this setup, we did two experiments, the first one we used the entire training dataset, split into training (80%) and validation (20%) subsets, in a balanced fashion in order to maintain the class ratios across both subsets. On the second one, given that the majority of the largest traffic signs on the dataset are of the class *speedlimit* (73%), we tried to cull the dataset into a smaller but more balanced distribution. To achieve this, we chose to keep a ratio of *speedlimit* instances equal to the double of the maximum ratio of the minority classes. This lead to a decrease of 53% on the dataset size but more balanced distribution of classes, with 43% being the highest.

On both occasions, we did an additional fine-tuning stage to try to escape the local minima, as the recall for the minorities classes were still on the lower side.

On the first experiment, the validation loss evolution stagnated on all stages on the later epochs and accompanied the training loss on a well-behaved manner (without overfitting to training data). We obtained a final cross entropy loss of 0.13 with a micro-accuracy of 98.15% on 9th epoch of the third stage.

On the second experiment, while training on a more balanced dataset, the lack of instances lead to a worse precision than the first experiment without improving the recall, obtaining a final cross entropy loss of 0.15 with a micro-accuracy of 0.92 on the 12th epoch of the second stage of training. In this experiment, the model didn't see any improvement during the third stage of training.

3.1.2 Training from scratch

For this step, we loaded the base Densenet201 [5], not pretrained in the ImageNet dataset. The model was trained for 30 epochs, with a learning rate of 1e-3 and the same optimizer scheduler as in the previous models.

The fine-tuning step, was not done on this model as it didn't have the pretrained feature extraction layers.

The best result was obtained in epoch 16 with a cross entropy loss of 0.7 and an accuracy of 0.76 (table 4).

3.1.3 Custom Architecture

For the task of traffic sign classification, we purpose a custom architecture based on Inception[15] and Densenet[4]. Taking advantage of the densely connected convolution networks presented in the Densenet architecture and multiscale convolutional modules of the Inception architecture, our purposed architecture is built on blocks, Fig. 1. Each block contains three branches, a branch with a 1x1 convolution and a branch with a 3x3 average pool followed by a 1x1 convolutional like the Inception architecture. The third branch is a series of densely connected convolutional layers, with all the previous features extracted being used on the deeper layers, similar to the Densenet architecture. In addition to these blocks, our network has an initial convolutional layer and a final adaptative average pool before entering the classifier layer. Batch normalization and activation layers are not mentioned here, but are present throughout the architecture.

Opposed to the train from scratch done with the Densenet, we attempted to use a different approach with this architecture to try to improve the classification recall of the minority classes. First, we trained the model for 10 epochs without the *speedlimit* traffic signs, in an attempt to counter the bias of the network to classify signs as *speedlimit*. With the weights already initialized, we trained the model now including the *speedlimit* signs, for another 30 epochs with a smaller initial learning rate, so the network could learn to classify *speedlimits* while attempting to maintain the bias towards the minority classes. This proved to be effective, to an extent, as it will be discussed in the next section3.1.4.

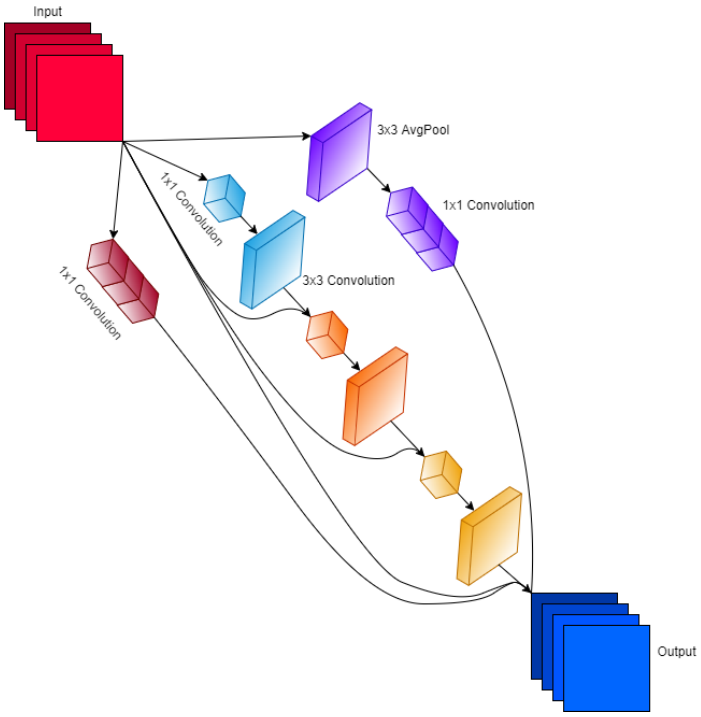

Figure 1: Traffic Signifier block.

3.1.4 Summary

In this section, we will compare the performances of the models by comparing the measured metrics and present a brief analysis of the results. The values for the accuracy correspond to the micro-accuracy of the model.

	Basic Transfer Learning			
	precision	recall	f1-score	support
trafficlight	0.89	1.00	0.94	24
speedlimit	0.96	0.98	0.97	192
crosswalk	0.81	0.68	0.74	25
stop	1.00	0.87	0.93	23
accuracy			0.95	264
macro avg	0.92	0.88	0.90	264
weighted avg	0.95	0.95	0.95	264

Table 1: Test results basic transfer learning model

	Basic Stratch			
	precision	recall	f1-score	support
trafficlight	0.50	0.04	0.08	24
speedlimit	0.76	0.92	0.83	192
crosswalk	0.09	0.04	0.06	25
stop	0.63	0.52	0.57	23
accuracy			0.72	264
macro avg	0.50	0.38	0.38	264
weighted avg	0.66	0.72	0.67	264

Table 2: Test results for basic scratch model

As it can be seen from table 1, the transfer learning model was able to classify almost every signal class with high precision. At a first glance, even tough the crosswalk class has a comparable number of occurrences as the traffic lights and stop classes, the model performed far worse on this class. This is corroborated by the low recall and f1-score that was obtained for the crosswalk. The reasons for this are the class imbalances of the dataset and that the crosswalk traffic signs tended to be placed closer to speed limits, which has a much larger support. This constituted a problem to model, which became very good at classifying the speed limit signs and overshadowed the crosswalks.

The model above can not be compared to the following ones, trained from scratch and custom architecture, as the amount of data each was trained is completely different. This is due to the Densenet used in the

	Basic Custom			
	precision	recall	f1-score	support
trafficlight	0.40	0.08	0.14	24
speedlimit	0.79	0.94	0.86	192
crosswalk	0.47	0.32	0.38	25
stop	0.75	0.39	0.51	23
accuracy			0.76	264
macro avg	0.60	0.43	0.47	264
weighted avg	0.72	0.76	0.72	264

Table 3: Test results basic custom model

	Accuracy	Loss	Best Epoch
Transfer Learning	0.981	0.130	54/55
Scratch	0.759	0.702	16/30
Custom	0.632	0.860	12/30

Table 4: Result summary of the best training epochs

transfer learning model being already trained on a large dataset, ImageNet[3]. As such, these models will only be compared with each other.

Both the model trained from scratch and custom architecture had a low recall on the minority classes, as expected, since their presence on the dataset are overshadowed by the major class. However, we can observe the strategy used for training the custom architecture proved to improve the recall of the minority classes, obtaining a significantly higher recall on crosswalks (32% compared to 4% of the model from scratch). Overall, the custom architecture managed to achieve a better performance across all classes compared to the Densenet trained from scratch.

3.2 Multi-label Classification

To solve this problem, we used the same base model as the in the last section. As the predictions were now binary, some changes had to be made to the labels, which needed to be a list of binary numbers with the same size as the number of target classes. Each index of the list represents a class. If an image has a given class, then the list of labels for that image will take a value of 1 on the index of the represented class.

We this requirement in mind, we changed the activation function from softmax to Sigmoid, to ensure that the outputs were between 0 and 1. We then passed these converted predictions to a Binary version of the Cross Entropy Loss function.

The metric used was the same as in section 3.1 but with macro average, which calculates the metrics for each class separately, and then averages it.

3.2.1 Transfer Learning

Following the same procedure as in section 3.1.1, without making the balanced split and without the third stage of training, we froze the pretrained feature extraction layers and only trained the classifier layer for 30 epochs with a learning rate of 1e-3. Next, we selected the best model from the previous stage and unfroze the feature extraction layers and trained it for 15 more epochs with a learning rate of 1e-5. By doing this, the model could adjust the weights of its feature extraction layers to better extract the features of traffic signs.

3.2.2 Training from scratch

With the same model as in the previous sections, with the changes that were discussed in 3.2, we loaded a version that was not pretrained on the ImageNet dataset, and trained it for 30 epochs with a learning rate of 1e-3 and the *ExponentialLR* scheduler.

For the same reasons as in section 3.1.2 no fine-tuning was performed. With this setup, we obtained a binary cross entropy loss of 0.357 and an accuracy of 0.867 (table 9) on epoch 18.

3.2.3 Summary

For simplicity, we only show the confusion matrices of the transfer learning model, in which we can observe a struggle to classify the minority classes, leading to higher number of false negatives. The other models suffer even more from this problem, as they lack the pretraining on ImageNet dataset, even obtaining zero true positives on traffic lights, in both cases.

		Predicted	
		Not present	Present
True	Not Present	233	0
	Present	10	21

Table 5: Confusion matrix for traffic lights by transfer learning model

		Predicted	
		Not present	Present
True	Not Present	60	0
	Present	2	202

Table 6: Confusion matrix for speed limits by transfer learning model

		Predicted	
		Not present	Present
True	Not Present	213	0
	Present	29	22

Table 7: Confusion matrix for crosswalks by transfer learning model

4 Object Detection

4.1 Two-Stage Architecture

For the two-stage object detection architecture, we opted to use the Faster-RCNN[13] model provided by Pytorch[10]. For the backbone of the model, we used a Densenet201 with a Feature Pyramid Network built from the third and fourth dense block of the Densenet. For the region proposal network, we utilized anchors with five different sizes and three different aspect ratios.

For the strategy used to train this model, we utilized the same optimizer and scheduler as in the other task. To keep track of the performance during training, we use the mean average precision, *mAP*, to evaluate the model, being the objective to maximize this metric.

Similarly to the transfer learning strategy, we trained the model in two stages. In the first, we froze the feature layers of the backbone, leaving every other layer unfrozen, including the feature pyramid network, and proceeded to train for 30 epochs. Afterwards, we further tuned the model by training another 15 epochs with a smaller initial learning rate, but with the entire backbone frozen, including the feature pyramid network.

The mean average precision on the validation set stagnated after 20 epochs, obtaining the best model in the last epoch with a mean average precision of 43.27%. On the second stage, the mean average precision increased on the first epoch but began to be unstable on the following epochs. This led to the best epoch during the second stage to be the first one, obtaining a mean average precision of 44.17%.

When generating the predictions for the inference stage, we perform additional operations on the outputs of the model, namely, non-maximum suppression with an intersection over union threshold of 30%, followed by a variation of the non-maximum suppression using intersection over self. This variation uses takes the ratio of the intersection over the area of the bounding box with lower score, instead of the union, and excludes that bounding box with lower score if the ratio is above a threshold, in our experiment, we used a threshold of 70%. Finally, we apply a filter by confidence score with a threshold of 50%.

4.2 Single-Stage Architecture

For the single-stage object detection architecture, we used Glenn Jocher's [7] version 5 of the YOLO object detector, trained on the COCO dataset [8]. To train the model, we had to reorganize the data directories to fit the requirements of YOLO's train method. The train, validation and test images and labels had to be under their specific folders.

In the training phase, we defined a set of hyperparameters which performed similar image augmentations as the ones discussed in section 2.1.

		Predicted	
		Not present	Present
True	Not Present	237	0
	Present	7	20

Table 8: Confusion matrix for stop signs by transfer learning model

	Macro Accuracy
Transfer Learning	0.9545
Scratch	0.8561
Custom	0.8456

Table 9: Summary of results for the different models on the test dataset

Some augmentations, such as CLAHE, Posterize and greyscale transformations were not used, as they weren't supported by their implementation. We used the default values for the learning rate, the momentum and weight decay.

After having defined the hyperparameters for the model, we then executed the train script with 15 epochs and the *SGD* optimizer. The cross validation is done automatically by their implementation, we only had to provide a valid train validation split. We also tried the *Adam* and *Weighted Adam* optimizers, however, they didn't produce the results as good as *SGD*.

In the validation stage, we executed the provided validation script with a confidence threshold of 25% and a non-maximum suppression with an intersection over union of 45%.

The best training model, had a mean average precision of 69.2% and an object loss of 1%.

4.3 Summary

Judging the results manually, we observed issues with F-RCNN that may have likely originated from the dataset itself. In multiple occasions, the model would detect two traffic lights, Fig. 4 of Appendix, but in the dataset it was only annotated as one. This may have affected the performance of the model significantly, as we can observe the *mAP* for the traffic lights is the lowest. Another problem that may have originated from the dataset is the missing labels for traffic signs that should've been annotated, like smaller crosswalks on the image that aren't annotated, like in Fig. 3, that can lead to a slower or worse training of the model, as the model may detect those crosswalks but get punished by it.

	Mean Average Precision (mAP)	
	F-RCNN-Densenet201	YOLOv5
trafficlight	0.2564	0.53
speedlimit	0.7142	0.822
crosswalk	0.4752	0.68
stop	0.6471	0.775
macro avg	0.5232	0.702

Table 10: Test results for F-RCNN and YOLOv5

Comparing the results between YOLO and F-RCNN, it can be stated that the former has an overall better classification performance over the latter. YOLO fared better in all classes, but the traffic sign class stands out the most, with almost two times the score of the F-RCNN. We attribute this difference to the characteristics of the pretrained YOLO model, which was already trained to classify traffic lights from the COCO dataset [8].

5 Conclusion

In this article, we studied multiple approaches to solve the tasks proposed, image classification and object detection, using Deep Learning techniques. On the image classification task, we observed the transfer learning model to perform the best, as expected, since these were pretrained on ImageNet, a much larger dataset, and as such, required a lower amount of instances to adapt to our task. Contrary to the model trained from scratch and custom architecture that could only rely on our small and imbalanced dataset, leading to an overwhelming bias towards the majority class, speed limits. In this task, we also purposed a custom architecture, based on the Inception and Densenet model, although its potential is difficult to measure on such a small dataset.

In the second task, object detection, we experimented with two different architectures and obtained similar results, visually by human evaluation, although, YOLO performed better in most occasions. As explained in the relevant section, this task may have been hindered by the lack of quality of the dataset used.

For future work and improvements, we could explore more with fine-tuning of the parameters and studying more deeply the training evolution. Additionally, we could also train the non-pretrained models on a larger dataset (for example, tiny ImageNet for image classification and tiny COCO for object detection) and then fine-tune to our task.

Figure 2: Traffic Sign detection on road463 by F-RCNN-Densenet201.

References

[1] Albumentations. Albumentations, 2022.

[2] Ambika Choudhury. What is the difference between image classification & object detection techniques?, 2022.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[4] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2016. URL https://arxiv.org/abs/1608.06993.

[5] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[6] IMB. What are convolutional neural networks?, 2022.

[7] Glenn Jocher. ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference, February 2022. URL https://doi.org/10.5281/zenodo.6222936.

[8] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014. URL https://arxiv.org/abs/1405.0312.

[9] Larxel (Owner). Road sign detection, 2020.

[10] PyTorch. Pytorch, 2022.

[11] PyTorch. Data tutorial, 2022.

[12] PyTorch. Torchvision, 2022.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015. URL https://arxiv.org/abs/1506.01497.

[14] Ruthger Righart. 3 fascinating applications of deep learning image classification, 2022.

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015. URL https://arxiv.org/abs/1512.00567.

A Appendix

A.1 Problems with missing labels

(a) Traffic sign detection with F-RCNN-Densenet201 for road167

(b) Traffic sign detection with YOLO for road167

Figure 3: Missing annotations for crosswalks in the dataset

A.2 Problems with traffic lights

(a) Traffic sign detection with F-RCNN-Densenet201 for road26

(b) Traffic sign detection with YOLO for road26

Figure 4: Different traffic lights annotated as one in dataset