Traffic Sign Detection Project

Manuel Scurti Mingjie Ye
Universidad Politecnica de Madrid
Deep Learning course
2019

1 Introduction

Traffic Sign Detection is an important task in Traffic Sign Recognition Systems. These systems can be directly employed inside new trending technologies of this decade such as: Autonomous Vehicles and Automatic Traffic Infraction Detection for Smart Cities. Many advances had been made in this research field to improve the general performances of signs recognition, but the most promising field able to efficiently address this task are for sure Deep Learning models. These models have exceeded expectations in terms of performance and arriving at state of the art results in most of researches and industry applications. In particular, the whole Computer Vision industry has reached unexpected amazing results in terms of accuracy and performances thanks also to techniques inspired by the natural human visual cortex architecture: Convolutional Neural Networks.
In the past, the mainstream methods for traffic sign detection were based on traditional object detection algorithms. The pipeline of traffic sign detection generally uses hand-crafted features to extract region proposals, and then combines classifiers to filter out the negatives. Convolutional neural networks (CNNs) can learn features from many samples without preprocessing, which avoids the design difficulty of hand-crafted features and learns more generalized features.
CNN has already been presented as a classifier in machine learning and has been used in traffic sign classification. The performance of the CNN-based method is the state-of-the-art on the German Traffic Sign Recognition Benchmark (GTSRB) [5], which is, the dataset we will be using to test and develop our project. The GTSRB is a dataset containing more than 40 classes, used to evaluate the performances of a model against a multi-class classification problem with unbalanced class frequencies. GTSRB was presented as a competition at IJCNN 2013 (International Joint Conference on Neural Networks). Traffic signs can provide a wide range of variations between classes in terms of color, shape, and the presence of pictograms or text. However, there exist subsets of classes (e. g., speed limit signs) that are very similar to each other.

In general, the research of traffic sign recognition is divided into sign detection and sign classification, and thus, this project has been divided in two main objectives: getting a working model for traffic signs classification, then integrate the model into a box proposal network to identify the location of the signs inside an image.
We will first start by investigating the differences in terms of performances between a classic dense neural network and a convolutional neural network. Then, we will try to improve the classification accuracy using an advanced technique known as Transfer Learning. Finally, we will integrate one chosen model into a traffic sign detector.

2 Experiment Design

Firstly, we built a simple classic dense neural network to check the performance. During this section, we tried various architectures, hyperparameters, activation functions, optimization algorithms and numbers of epoch. Besides, in order to improve the result, we did some preprocessing such as resizing the input images or data augmentation. And we also use early stopping to address overfitting tendencies.
Secondly, Convolutional Neural Network was used in our model and improve our result a lot. Our architecture is inspired by the VGG Net which is characterized by its simplicity, using only 3×3 convolutional layers stacked on top of each other in increasing depth. Max pooling reduced volume size. On top of them, two fully-connected layers and then followed by a softmax layer. Our chosen optimization algorithm is the Stochastic Gradient Descent with Nesterov. Also Batch Normalization and Dropout are employed respectively as normalization and regularization techniques. To help the net visualize better the content of the image. We then taught to apply Histogram Equalization. Data augmentation was also used here.
Then we used a pre-trained model on a similar task and reuse the feature extractors for our images by transfer leanring. By tring various pre-trained model, we finally chose InceptionV3 because it has both high accuracy and low time.
Finally, we want to build a traffic sign detector based on what we have. It is divided by two stages. The First one is to generation of regions of interest using a Region Proposal Network which are the object we want to classify. During this stage, we have to manually implement a filter for these boxes to improve the quality of the proposals otherwise most of them are not related with traffic signs. The second stage is to determine the type of traffic sign contained in the proposal. Therefore, for each box proposal, we could determine if the box proposal contains traffic sign or not with the binary classifier. If yes, determine which kind of traffic sign is in the proposal using the Inception NN. Finally

save the results and use mAP (mean Average Precision).

3 Dense neural network

We will start these experiments by trying a very basic approach using only dense layers, to see how far we can get. Our starting architecture is composed by just one wide hidden layer of around 400k neurons, using ReLU as activation function. The input layer receives each single pixel composing a traffic sign image of size 224x224. The chosen optimization algorithm is a Stochastic Gradient Descent with Nesterov with a learning rate value of 0.001. Furthermore, Dropout is employed as regularization technique. In the figure below it is shown a resume of the architecture.

By training this net for 50 epochs and with a batch size of 16, we achieved these results:

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 224, 224, 8)       32
_________________________________________________________________
flatten_1 (Flatten)          (None, 401408)            0
_________________________________________________________________
activation_1 (Activation)    (None, 401408)            0
_________________________________________________________________
dropout_1 (Dropout)          (None, 401408)            0
_________________________________________________________________
dense_2 (Dense)              (None, 43)                17260587
_________________________________________________________________
activation_2 (Activation)    (None, 43)                0
=================================================================
Total params: 17,260,619
Trainable params: 17,260,619
Non-trainable params: 0
```

Figure 1: Architecture of DNN

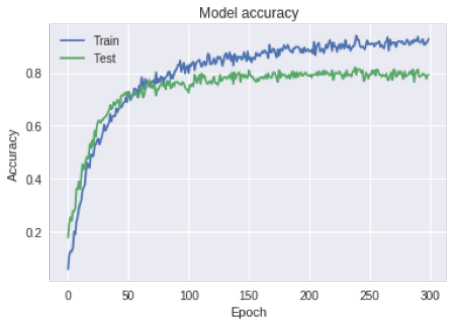
```
MLP took 1.0451841354370117 seconds
Test loss: 0.8119410662769941 - Accuracy: 0.81994459850306
```

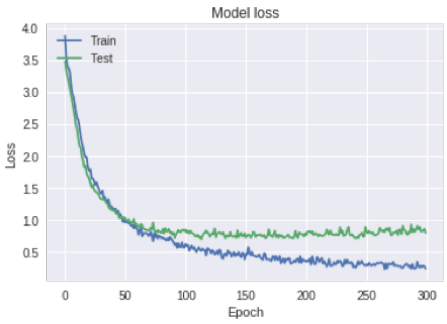
Actually the results are pretty good, but the inference time of the network is way too slow to be used inside a traffic sign detector. Let's see how we can improve the network. At first sight, what we noticed is that the network can't improve the performances if it remains so wide, because adding new layers dramatically worsens the speed and even worse this net was overfitting all the time. For this reason, we will try to reduce the width of the net and increase the depth. Furthermore, we tried to speed up the whole process by introducing a little preprocessing on the input image by resizing it to 32x32 pixels. The final network architecture is then composed by 5 hidden layers of 2048,

1024, 256, 256, 100 units per layer, while the activation function has been changed to Leaky ReLU because has been showed multiple times his better performances when the depth of the net increases [11][1]. The dropout regularization uses different probability values depending on the layer of the net: top layers have higher probability to take out units from training steps, using a probability of 0.3, while bottom ones have only a probability of 0.15. The optimization algorithm and the learning rate remains unchanged. With this configuration, after a training lasted 300 epochs, the overfitting problem was completely solved and the network showed the following results:

```
MLP took 0.053617238998413086 seconds
Test loss: 0.4928138810841991 - Accuracy: 0.8781163436554146
```

And not only we achieved a better accuracy but also we improved the time performances by a magnitude of 2. Due to the cut we made on the number of neurons, also the general computational cost of the net is decreased because the number of parameters passed from the initials 17,260,619 to just 4,559,711. Here we show the evolution of accuracy and loss by epochs.



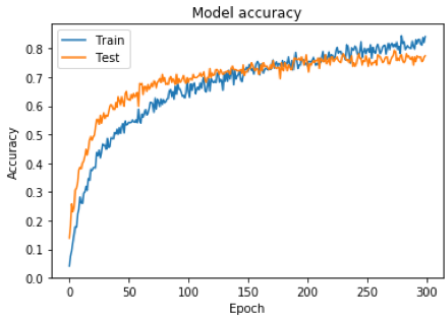


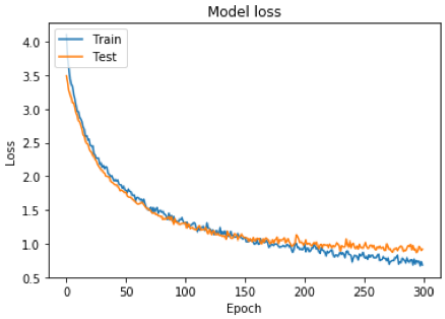
Because of the plots showing an overfitting tendency during the last half of the epochs, we taught that we could do more than that by adding some normalization techniques, such as Batch Normalization, between the linear layers (the dense layers) and the non-linear layers, because it normalizes the input arriving at the activation function. Even more, we were not happy with the size of the input image, because we taught that was too little to describe the content of a traffic sign, so we increased the size up to 48x48 pixels. Also we changed the optimization algorithm to Adam, such that we decreased the number of hyperparameters. Thus, in the final version of this model we obtained:

```
MLP took 0.035666704177856445 seconds
Test loss: 0.6443077728688882 - Accuracy: 0.8836565098604007
```

This is our final results for the dense neural network model. At this moment, the model started to show some limitations, and was very difficult to get a better score. We even experimented different

[1]https://arxiv.org/pdf/1505.00853.pdf





configurations or training techniques but without success. These experiments included:

- Early stopping, to address overfitting tendencies
- Data augmentation, to increase the robustness of the network and thus the abstraction power
- Also other optimizers, like Adagrad, Adadelta and Adamax

It is interesting to notice that data augmentation decreased the network performances putting evidences on the fact that a dense neural network has a very weak abstraction power. These are in fact, the results obtained by increasing the population of the dataset.

```
MLP took 0.03498220443725586 seconds
Test loss: 1.8645096412986268 - Accuracy: 0.4709141274651002
```

As we can see, the results are a lot worse than without data augmentation. This suggests to us that by only using dense layers the network is not able to deal with new traffic signs images derived from the original set, even with little modifications of lighting condition, orientation and centering of the sign inside the image. This of course, is a problem to be taken into account when deciding which network should be employed inside the traffic sign detector, as this will often present traffic signs in very different sizes, partial occlusions and lighting conditions. We will try to address the abstraction power using a CNN, known to be successfully at solving most of the computer vision tasks in research and industry fields.

4 Convolutional neural network

A Convolutional Neural Network (CNN) is a Deep Learning model which can take in an input image, assign importance (learnable weights and biases) to various features in the image and be able to differentiate one from the other. While in primitive methods for image recognition, filters are hand-engineered, with enough training, CNNs have the ability to learn these filters/characteristics. As an example, in the case of a traffic sign, a CNN will be able to automatically understand what it

has to look for to distinguish signs and, as a result, Convolutional Layers will learn to extract visual characteristics such as color, shape and text in the sign, in the same way as we humans do. For this reason, the pre-processing required in a CNN is much lower as compared to other classification models.

The architecture of a CNN is inspired by the connectivity pattern of neurons in the Human Brain and the organization of the Visual Cortex. Individual neurons respond to stimulation only in a restricted region of the visual field known as the Receptive Field. The receptive fields of different neurons may overlap, and together they tile the whole visual field [7].

The reason why CNNs are often performing way better than Fully Connected NN, at least in Computer Vision tasks, is that a CNN is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. In fact, the strengths of this model are:

- The ability to deal with high dimensional data without losing features
- Exploits the 2D/3D topology of pixels
- Comes with a built-in certain degree of invariance
- Reduction in the number of parameters involved and Reusability of weights

A typical convolutional net is characterized by a set of Convolutional layers, Pooling layers and fully connected layers. The Convolution layer is where the convolutional operator is applied to the input, it's objective is to extract high-level features such as shapes, from the image. By stacking different convolutional layers we get a stack of feature extractors starting from low to high level features, such that the net gets a wholesome understanding of images in the dataset.

The Pooling layer is responsible for reducing the spatial size of the previous layer. This is to decrease computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

Finally, Fully Connected layers, placed on top of the Convolutional layers, represent the part of the net responsible for the classification and thus, they act like a normal dense neural network but with the difference that they don't receive the original image but feature maps obtained from the feature extractors.

We wanted to experiment with CNNs and see if we can get better results on our traffic sign classification task. Our architecture is inspired by the VGG Net, a popular convolutional neural network model presented by the Visual Geometry Group at the University of Oxford [2]. This network is character-

[2]https://arxiv.org/pdf/1409.1556.pdf

ized by its simplicity, using only 3×3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. On top of them, two fully-connected layers and then followed by a softmax layer. Our chosen optimization algorithm is the Stochastic Gradient Descent with Nesterov. Also Batch Normalization and Dropout are employed respectively as normalization and regularization techniques.

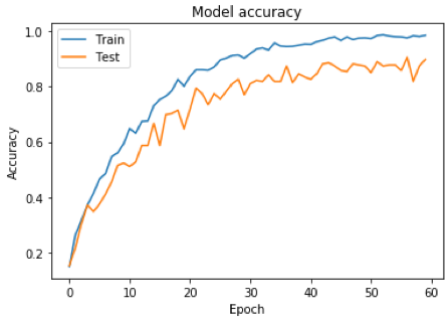
- Conv Layer – 224 x 224 x 32 – stride 1x1
- Max Pooling – size 2x2
- Conv Layer – 112 x 112 x 32 – stride 1x1
- Max Pooling – size 2x2
- Conv Layer – 56 x 56 x 64 – stride 1x1
- Max Pooling – size 2x2
- Conv Layer – 28 x 28 x 64 – stride 1x1
- Max Pooling – size 2x2
- Conv Layer – 14 x 14 x 128 – stride 1x1
- Max Pooling – size 2x2
- Conv Layer – 7 x 7 x 128 – stride 1x1
- Max Pooling – size 2x2
- Fully Connected Layer – 512 units
- Fully Connected Layer – 256 units

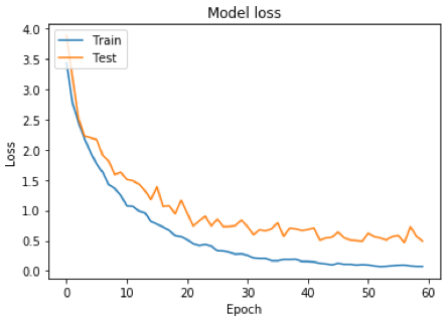
The input size of the image, is then left as the original, 224x224 pixels, in order to let the net learn spatial features. With this architecture we achieved these results:

```
CNN took 0.3768482208251953 seconds
Test loss: 0.3178430259062643 - Accuracy: 0.9252077563977968
```

Even if the input size of the image is quite big, the inference time of the net is still pretty good. In fact, compared with a Dense Neural Network with the same input size, we achieved a very good speed performance. Furthermore, the accuracy has been improved a lot. Let's run some other tests on this model and see if also the abstraction power is better than the dense neural network. To do this, we applied the same type of transformations to the images to increase the size of the dataset. These are the results of the same model but this time using data augmentation.

```
CNN took 0.3935418128967285 seconds
Test loss: 0.32424788181141145 - Accuracy: 0.9279778395002899
```

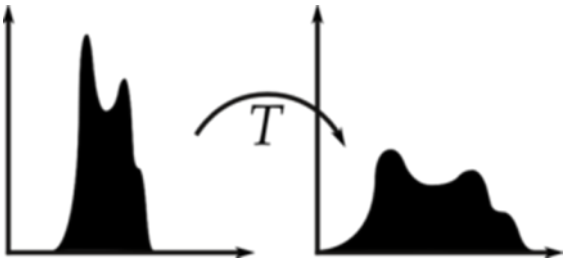




The results are interesting, the CNN seems both to increase the classification accuracy while still maintaining a reasonable speed. Furthermore, this model only uses 1.024.459 parameters and it has been trained in only 60 epochs, and thus saving a lot of computational cost, compared with the previous models. However, even if the results are good enough to continue with the project, we wanted to try additional experiments.

Our first experiment focused on some advanced pre-processing techniques on images, to help the net visualize better the content of the image. We then taught to apply Histogram Equalization[4]. First, we find that the contrast of images in our dataset varies from low to high while a good image has pixels from all regions, so we use histogram equalization to improve the contrast of the traffic signs images and keep them in a relatively balanced contrast.

An image histogram is a graphical representation of the intensity distribution of an image. It quantifies the number of pixels for each intensity value considered. And histogram equalization is a method that stretches out the intensity range in order to improves the contrast in an image.

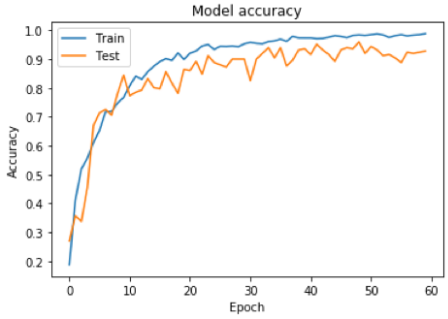


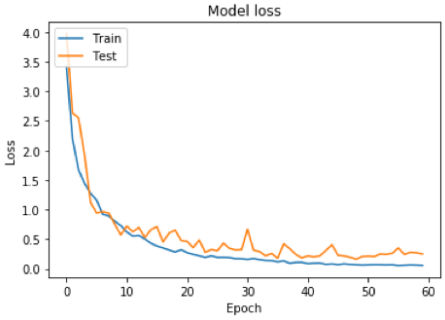
By applying this technique we achieved.

```
CNN took 0.39928770065307617 seconds
Test loss: 0.09524870017244173 - Accuracy: 0.9584487534626038
```

Let's see how it behaves using this time data augmentation.

```
CNN took 0.37682485580444336 seconds
Test loss: 0.19986060492364158 - Accuracy: 0.9362880886426593
```



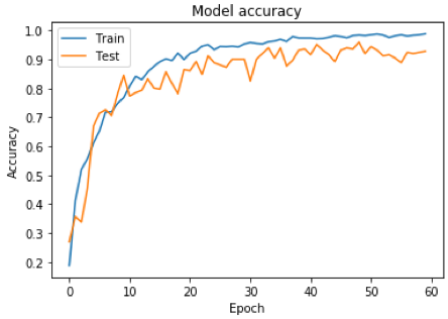


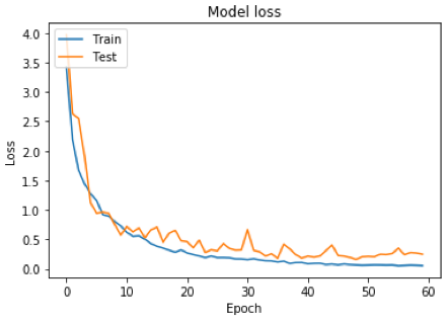
These results show that the net is receiving better images, with less variance in lighting conditions and thus it can classify more traffic signs than before. However, we still want to try resizing the input image to speed up the computational time. We will scale all the images from 224x224 to 48x48 pixels. The architecture, needs some minor adjustments to fit with the new image size. This is the new structure of the network:

- Conv Layer – 48 x 48 x 32 – stride 1x1
- Conv Layer – 48 x 48 x 32 – stride 1x1
- Max Pooling – size 2x2
- Conv Layer – 24 x 24 x 64 – stride 1x1
- Conv Layer – 24 x 24 x 64 – stride 1x1
- Max Pooling – size 2x2
- Conv Layer – 12 x 12 x 128 – stride 1x1
- Conv Layer – 12 x 12 x 128 – stride 1x1
- Max Pooling – size 2x2
- Fully Connected Layer – 512 units
- Fully Connected Layer – 256 units

By training the net with identical configuration and data augmentation we obtained these results in 60 epochs.

```
CNN took 0.07359910011291504 seconds
Test loss: 0.3487075207615658 - Accuracy: 0.9501385041551247
```





Now the accuracy is slightly worse than before, but we've gained a lot of speed. We decide that this could be a promising model for the final traffic sign detector.

5 Transfer learning technique

Up to this point, we've seen that a Convolutional Neural Networks can learn extremely complex mapping functions when trained on enough data. We have also seen that a CNN works like a combined feature extractor and classification model, where features to extract are self-learned. However, training such models from scratch is possible for small projects, most applications require the training of very large CNN's and this takes extremely huge amounts of processed data and computational power. What if we could use a pre-trained model on a similar task and re-use the feature extractors for our images? That's where transfer learning comes into play. In transfer learning, we take the pre-trained weights and model on a different task, such as object recognition, and use these already learned features to predict new classes, and thus, in our case traffic signs.
The Keras high-level API[2] for Deep Learning provides a set of pre-trained models trained on the object classification benchmark. We based our experiments on 5 pre-trained, well-known, architectures:

- DenseNet201 [6]
- VGG19 [8]
- InceptionV3 [10]
- InceptionResNetV2 [9]
- NASNetMobile [12]

What we expect as results is to have a fast network (since we are using a pretrained network) and a great accuracy score. We started the experiments trying different ways to combine a pre-trained model. All the training processes were conducted with data augmentation for big models

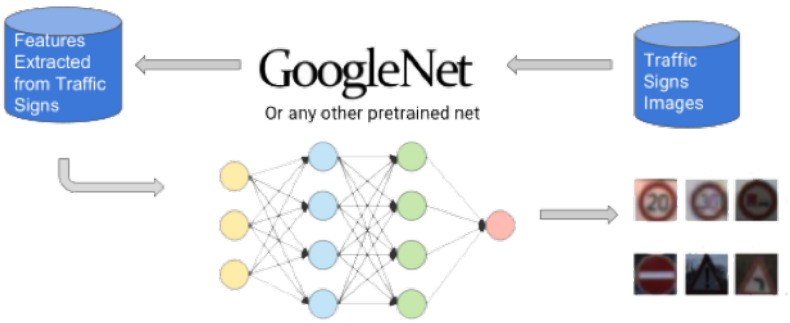
(convolutional layers + fully connected). and classic training for fully connected models.
The first attempt followed this process:

1. Take a pre-trained model without top fully connected layers
2. Attach to the top of the model a different fully connected network structure of our choice, with the 43 units output layer
3. Set ONLY the fully connected layers to be trainable/set at most the first convolutional part just behind the top layers
4. Train the whole model and see the results

As a result, training the model took 9 to 20 seconds per epoch and accuracy was stucked in different attempts between 6% and 70%.
The second attempt followed this process:

1. Take a pretrained model without fully connected layers (this time VGG19)
2. Use it as a bottleneck features extractor. i.e. transform our dataset into a newer one containing features found at the output of the last conv layer
3. Train a fully connected model from scratch using this dataset
4. Merge the trained fully connected model and the baseline model, freezing all convolutional layers except the last one, and train it using very small weight updates

In this experiment we achieved an accuracy score of 85% with an inference time of 0.40 seconds. It's still not what we expected. However, what emerges from the results is that the best accuracy we got was using only the pretrained model, while the best performance obtained was when we transformed our dataset using a feature extractor. Thus, we come up with the idea of caching the bottleneck features (i.e. the features coming out after all the convolutional layers) of the feature extractor and use them as an input for a completely new fully connected model.

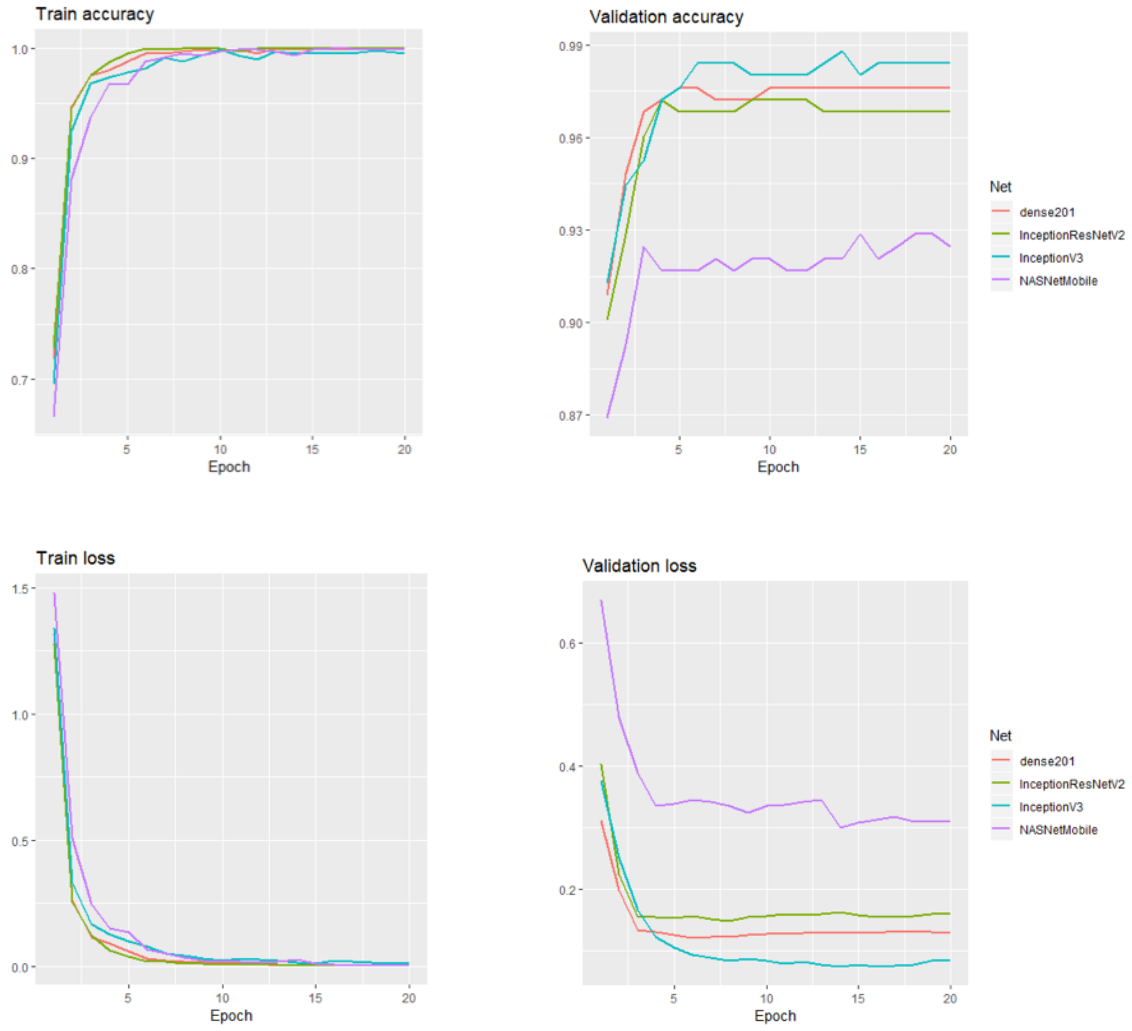


The final process followed is:

1. Take a pretrained model without fully connected layers
2. Train the pretrained model using traffic signs images for few epochs
3. Use it as a bottleneck features extractor
4. Train for few epochs a fully connected model from scratch using this new dataset

We gained a +13% in inference time and +4% better accuracy score. However, we're aware that caching the results is not possible in a real application of the network, such as a traffic sign detection system. In the below table, we show all the different pre-trained model that we used.

Bottleneck Feature Extractor	Training Parameters of the Extractor	Training Parameters of ffNN	Accuracy	Time
InceptionV3	Data aug, epochs 60	Epochs 20	0.9833	0.2174
Dense201	Data aug, epochs 60	Epochs 20	0.9889	0.4528
InceptionResNetV2	**Data aug, epochs 40**	**Epochs 20**	**0.9861**	**0.2952**
NASNetMobile	Data aug, epochs 40	Epochs 20	0.9667	0.3020
VGG19	Data aug, epochs 60	Epochs 20	0.0747	0.1560



Note: VGG19 is excluded from the plots.

However, even if these are great results. The problem is that the inference time showed here is lying about the real time needed to start from an image an get to a traffic sign label. In fact, this time is affected by two contributions:

$$Inferencetime = featureextractiontime + classificationtime$$

If we cache the results, as we showed in the last experiment, we're completely dropping the feature extraction time, because this process has been already done to all the images, resulting in the following:

$$Inferencetime = classificationtime$$

For this reason we decided to not include the results of the transfer learning technique inside the traffic sign detector and instead include the classical CNN showed above. In fact, the pre-trained model with its inferences time will affect the whole traffic sign detector performance, resulting in an unfeasible system for real world application.

6 Traffic sign detection

Finally, it is time to wrap up all the things we made so far and include them into a complete Traffic Sign Detector. For this scope, we will be using an already implemented Region Proposal Network. A Traffic Sign Recognition system is composed by two stages that can be merged to a single stage depending on the implementation. In the R-CNN architecture for object detection there are two stage: generation of regions of interest using a Region Proposal Network, classification of the entity inside the region with Convolutional Neural Network. The objective of the RPN is to extract all the interesting regions of the image in which it could reside the object we want to classify. In our case, the RPN will be trained to extract bounding boxes containing traffic signs.

We will be using an RPN based on Recurrent Neural Networks. However this network is giving to the classifier a lot of proposals, and most of them are not related at all with a traffic sign, thus we have to manually implement a filter for these boxes to improve the quality of the proposals.
On top of the RPN we will place two CNNs, with same structure but different tasks. The first one will be in charge of distinguish if the proposal is either a traffic sign or not. The second one will be used to determine the type of traffic sign contained in the proposal. Therefore, for each box proposal, we determine if the box proposal contains traffic sign or not with the binary classifier. If yes, determine which kind of traffic sign is in the proposal using the Inception NN. Finally save the results and use

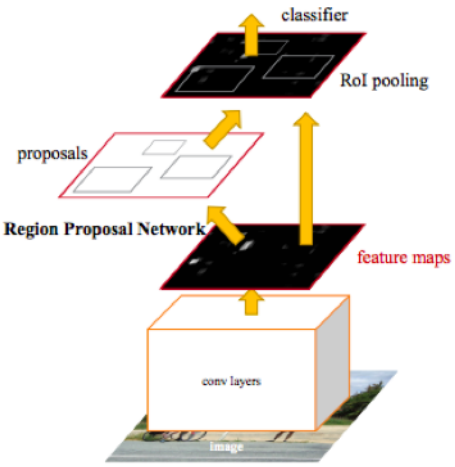


Figure 2: [3]

mAP (mean Average Precision). We will use a python script to automatically compute the mAP score [1]. The mAP, Mean Average Precision, is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. The AP value is computed for each class that the detector is in charge to classify, and it is basically the area under the precision-recall curve. Once we have the AP for all the distinct classes, we can compute the average and we will call it mAP. An object is said to be correctly detected and classified if the Intersection of Union (IOU) between the ground-truth (where the traffic sign really is) and the identified bounding box is more than 0.5, and the object label is correctly predicted.
The process followed to build this system is the following:

1. Build a dataset of negative samples to train the binary classifier to recognize whether a box
2. proposal contains a traffic sign or not
3. Filter, in a smart way, box proposals generated by the RPN in which the probability to contain a traffic sign is low
4. Check the results

One of the main problem of the first experiments was to deal with an high need of computational power to handle all the region proposals given by the RPN. For this reason, it was impossible to even think to use a pre-trained model, such as InceptionV3 that was performing very well, into this system. So we started to implement some filters to reduce the number of proposals, based on our common sense.

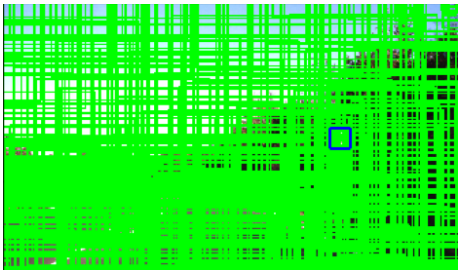
1. Filter out the boxes with large overlap areas with bounding.

To do this we used a basic implementation of non maximum suppression to delete the detected boxes whose overlap areas with the bounding box are larger than a certain threshold, which means these boxes are so large that are hardly to be granted as traffic signs from our human common sense.

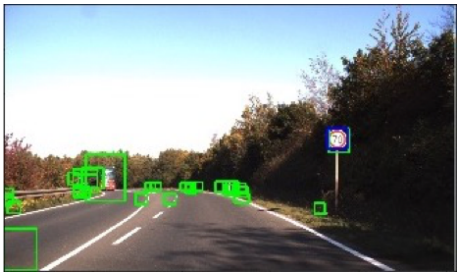
2. Filter out the boxes whose width-height ratio is too large or too small.

   We also filter out the boxes whose width-height ratio is too high or too low, because the traffic signs are generally circles and the width-height ratios of boxes could not be too far from 1. In our project, we delete these boxes whose width-height ratio are higher than 1.3 or lower than 0.7.

We can see the difference before and after using filter.



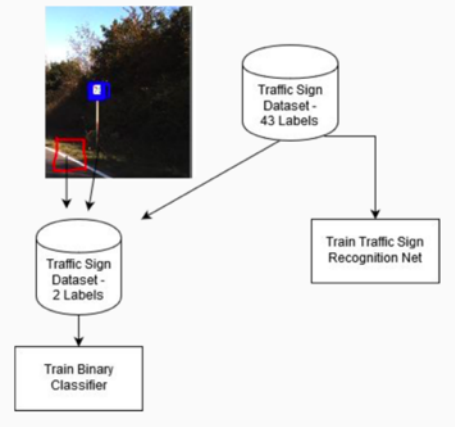
Before using filter



After using filter

We passed from 264015 proposals to 18730 proposals. It's a 92% decrease of the number of proposals.

To build a population of negative samples, i.e. images containing no signs, we extracted all the regions from RPN whose IOU with the ground-truth is zero.



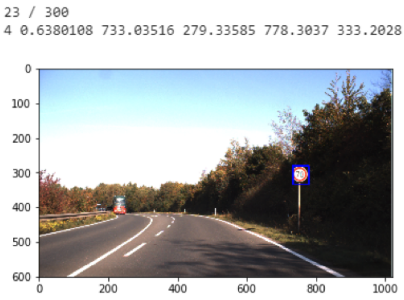
Then, our binary CNN was trained in 11 epochs, using also data augmentation, and pre-processing the images using histogram equalization and we obtained:

```
BIN CLASSIFIER took 0.7812988758087158 seconds
Test loss: 0.0612204083992256 - Accuracy: 0.9764542936288089
```

Finally, we loaded our CNN model, described in the Convolutional Neural Network section of this paper, into the system and wrap all together.

7 Result

Our system achieves a mAP of **18.68%** in **90.36** seconds. These results are far from what we expected, at least in terms of mAP. Let's analyze some detection examples. Our first example shows a correctly identified traffic sign.



Detected label = 4

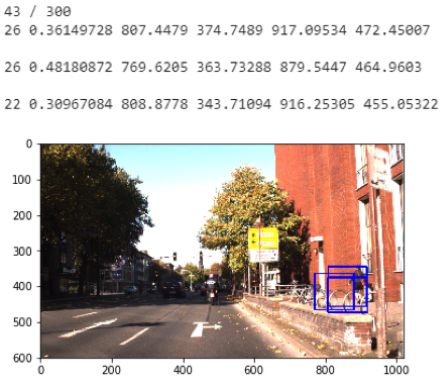
Mapping numbers - traffic signs

However, this example shows a good lighting condition and well defined contours and contents of the traffic sign. Let's analyze some more advanced examples to see where the problems are, but before this let's discuss about the source of errors. Errors can derive from:

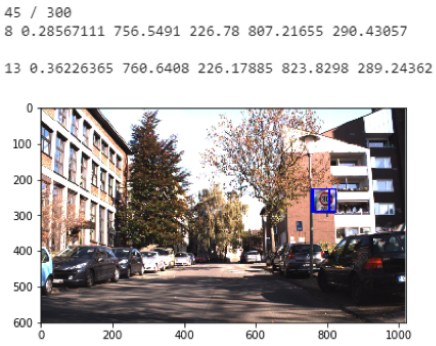
- Wrong region proposal, classified as a traffic sign
- Misclassified traffic sign
- Mix of the two errors

The most frequent errors happening in our system are of type 2. These are some examples:

On the left, an error of type 1, while on the right an error of type 2. These errors come mostly from the way the region is proposed. Regions appears most of the times not centered to the traffic sign position, such that the network, eventually not trained well on these situations, is not able to recognize them. Wrong proposals errors can be addresses by a more accurate training of the RPN or by some advanced filtering techniques. To address misclassification cases, we need to gain generalization power of the CNN to recognize traffic signs in possibly all kind of conditions. Another possible cause of these misclassifications can be also the preprocessing techniques applied to the



Before using filter



After using filter

images. In fact, the CNN could have misclassified regions according to the colors contained in the box, and thus could be too biased towards colors instead of shapes.

8 Conclusion and future work

There is room for improvements here. Some possible experiments we can run include:

1. Training a network with 43 labels of signs + 1 label representing “no sign” situation

   This could reduce the effort of training and maintaining two CNNs inside the system, while also speed up and getting more accurate predictions.

2. Using a pre-trained model

   This could solve the generalization power that we are experiencing. It has to be said that we tried this approach but at that time we didn’t come up with the idea of filtering proposals and thus we obtained unfeasible processing time and actually no mAP value, since the process has been stopped.

3. Using a more balanced dataset In the preprocessing section, we could balance the dataset with close proportion of good and bad labels. Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally and it’s common if we do not balance it. In order to balance it, we could use oversampling and undersampling, resampling or try to get more data.

References

[1] Cartucho - mean average precision implementation. https://github.com/Cartucho/mAP.

[2] Keras document - applications. https://keras.io/applications/.

[3] Region proposal network rpn: Backbone of faster r-cnn. https://medium.com/@tanaykarmarkar/region-proposal-network-rpn-backbone-of-faster-r-cnn-4a744a38d7f9.

[4] Wikipedia contributors. Histogram equalization. https://en.wikipedia.org/wiki/Histogram_equalization, 2018. Online; accessed 18-May-2018.

[5] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In International Joint Conference on Neural Networks, number 1288, 2013.

[6] 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, pages 4700–4708, 2017.

[7] Sumit Saha. A comprehensive guide to convolutional neural networks - the eli5 way.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[9] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[10] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2818–2826, 2016.

[11] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853, 2015.

[12] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 8697–8710, 2018.