Traffic Sign Recognition

Student: Guido Rezende de Alencastro Graça

This is the report of the third project of the Self Driving Vehicles course on Udacity. This project goal is to build a code that is able to classify traffic signs using a Convolutional Neural Network (convnet). The code can be found in Traffic_Sign_Classifier-Final.ipynb and .html files. The project Rubrics are:

1. Dataset Exploration
2. Design and Test a Model Architecture
3. Test a Model on New Images

1. Dataset Exploration

The dataset used on this project is the German Traffic Sign Dataset[1]. It is a dataset containing 51.839 images divided into three datasets: **Training, Validation and Testing:**

Table 1 - Images datasets

Dataset	Images
Training	**34799**
Validation	4410
Testing	12630

These images are colorful 32x32 pixels. The dataset has 43 different classes. Some examples from each dataset are shown below (Figure 1):

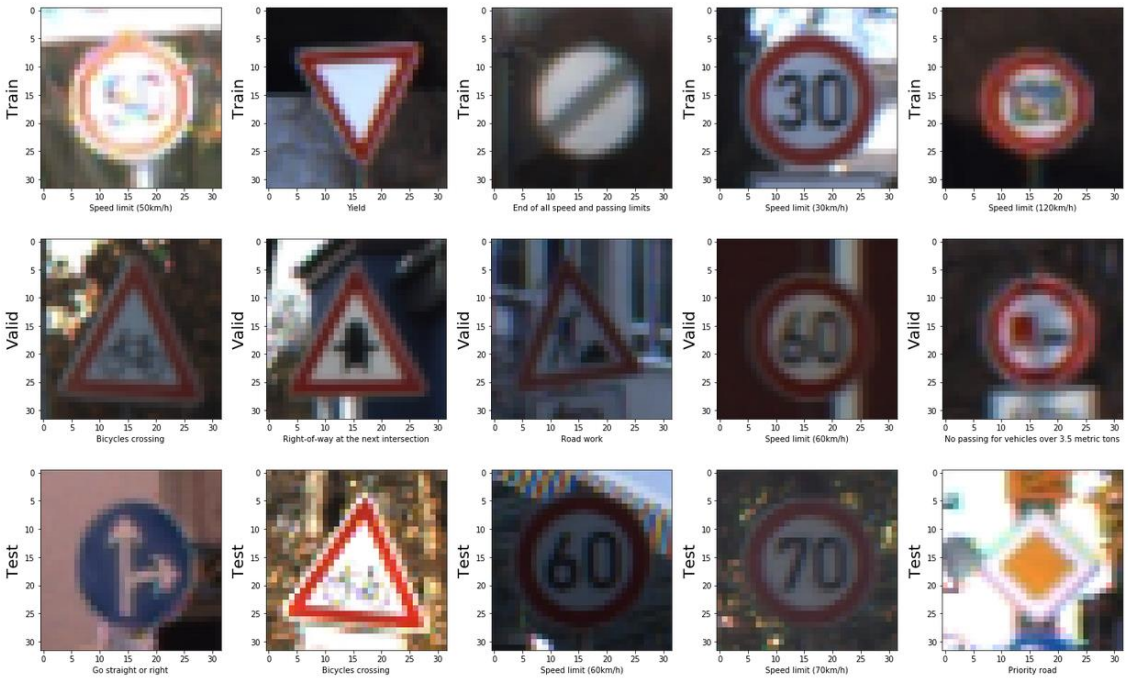

Figure 1- Examples of signs in the datasets

The distribution of each class on each dataset is shown in Figure 2. This distribution is not homogenous. This implies that some classes of signs are more common than others. Thus,

some images are easier than others to predict. Although, the distribution between the datasets are very similar, so the validation and testing stages are less tended to be biased.

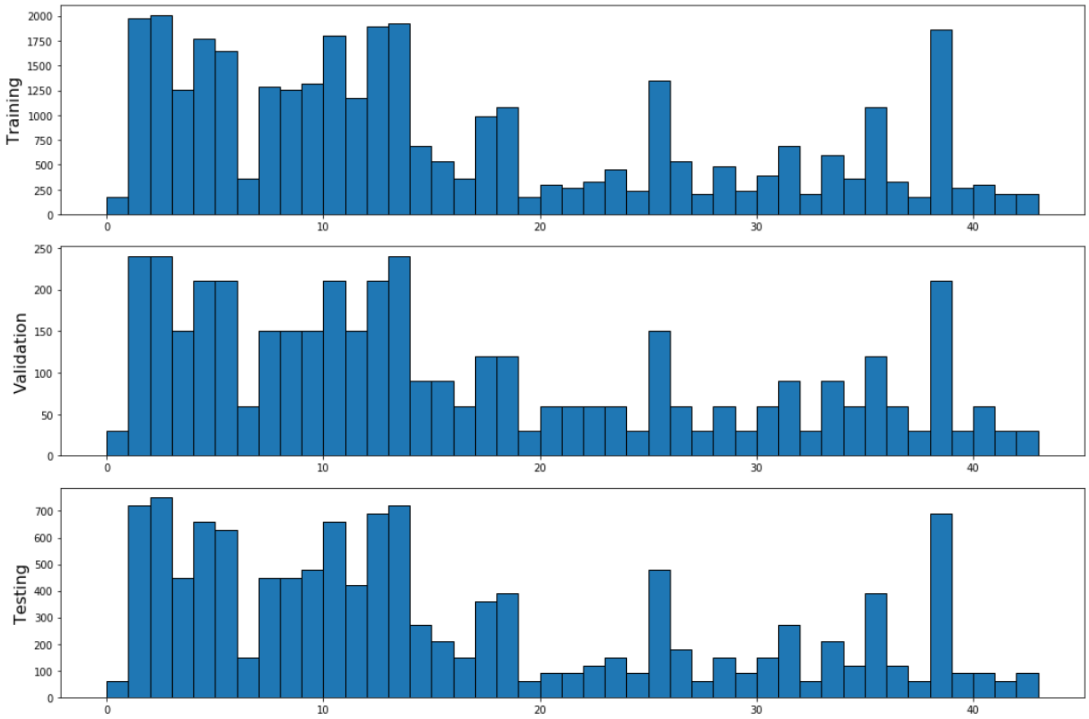

Figure 2- Distribution of the classes in each dataset

2. Design and Test a Model Architecture

Before the images were inputted in the convnet, they were converted to grayscale. Then, the images were normalized using the following equation:

$$P_{norm}(i,j) = \frac{(P(i,j) - P_{mean})}{\sigma_P}$$

Where $P_{norm}(i,j)$ and $P(i,j)$ are the normalized and original pixel value of the grayscale image, whereas P_{mean} and σ_P are the mean pixel value and the standard deviation of each image. Using this approach, each image has now zero mean and standard deviation of 1.

No augmentation of the dataset was used. The mirroring operation was not used because many signs would change their meaning. Rotation was not applied either because the signs usually are on the upright position on the road.

The used convnet architecture was a slightly different LeNet-5. The network was increased to use bigger convolutional layers on the second convolution (24 filters instead of 16) and on the fully connected hidden layers (180 and 120 instead of 120 and 84). This was used to augment the network learning capacity (Table 2).

Table 2 - Convnet architecture

Layer / Operation	Description	Image/Array output size
Input image	Grayscale normalized	32x32x1
Convolution 3x3	No padding, 6 filters, Stride of 1	28x28x6
Activation	Relu	28x28x6
Max Pooling	Same padding. Stride of 2	14x14x6
Convolution 3x3	No padding, 24 filters, Stride of 1	10x10x24
Activation	Relu	10x10x24
Max Pooling	Same padding. Stride of 2	5x5x24
Flattening	Flatten the layers	600
Fully connected	Hidden Layer 1	180
Activation	Relu	180
Fully connected	Hidden Layer 2	120
Activation	Relu	120
Fully connected	Final Layer	43

The learning hyperparameter used were:

Table 3 - Hyperparameters used in the convnet

Hyperparameter	Value
Learning rate	0.005
Batch size	256
Early termination threshold	-0.2%
Dropout keep probability	50%

The number of epochs used was not defined. Instead, it was used the **early termination method**. The training runs until the following validation accuracy decreases 0.2% relative the last iteration. In addition, it was used the **dropout method**. This method increased drastically the accuracy of the network, allowing it to reach over 90% accuracy on the validation dataset.

The accuracy values found are:

```
Training accuracy: 98.5%
Validation accuracy: 94.2%
Test accuracy: 92.3%
```

3. Testing the Model on New Images

New sign images were obtained using [a video on YouTube] of a car driving in Germany[2]. It was used nine different signs obtained from different frames of the video. They were cropped and resized to 32x32 size. They were saved as png files and then imported in the code.

Figure 3 - Example of signs found in the web

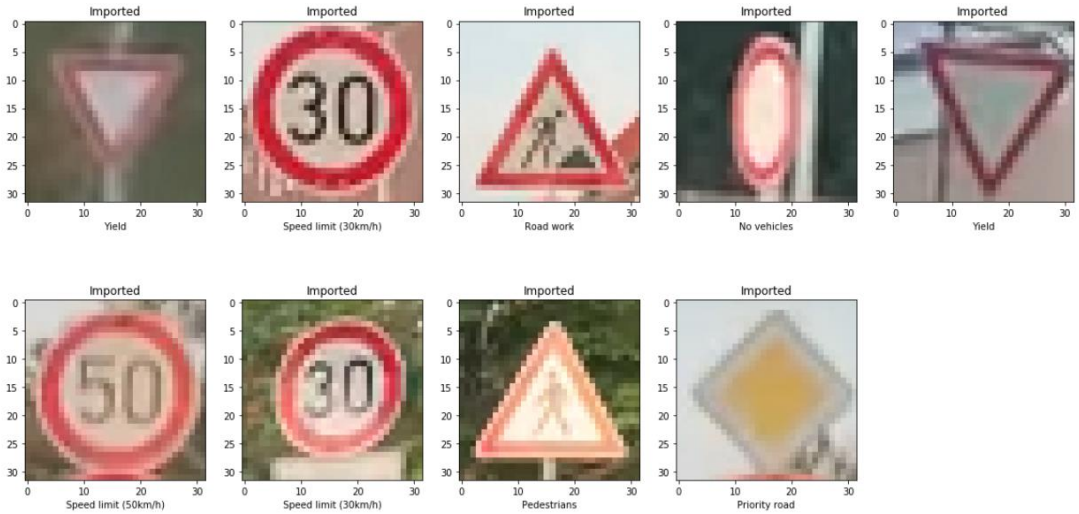

Figure 4 - Signs found on the video and their labels

The used signs were:

Table 4 - Signs classes found on the web

Sign type	Quantity
Speed limit (30 km/h)	2
Speed limit (50 km/h)	1
Yield	2
Road work	1
No vehicle	1
Pedestrian	1
Priority road	1
Total	9

The images were normalized before applying the convnet. The prediction on the imported images classified 7 out of 9 images correctly, resulting in a 77,8% accuracy. Below is shown how the network predicted on the images (Figure 5). The top 5 Softmax of the erroneously prediction are show (Figure 6). The prediction of the others images can be found at the html file

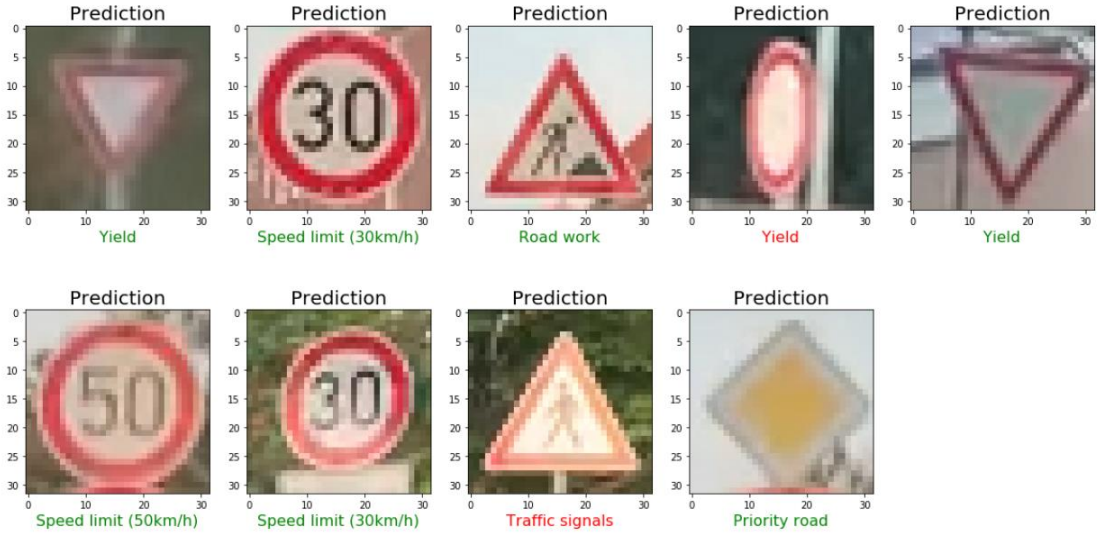

Figure 7 - Prediction on the imported images

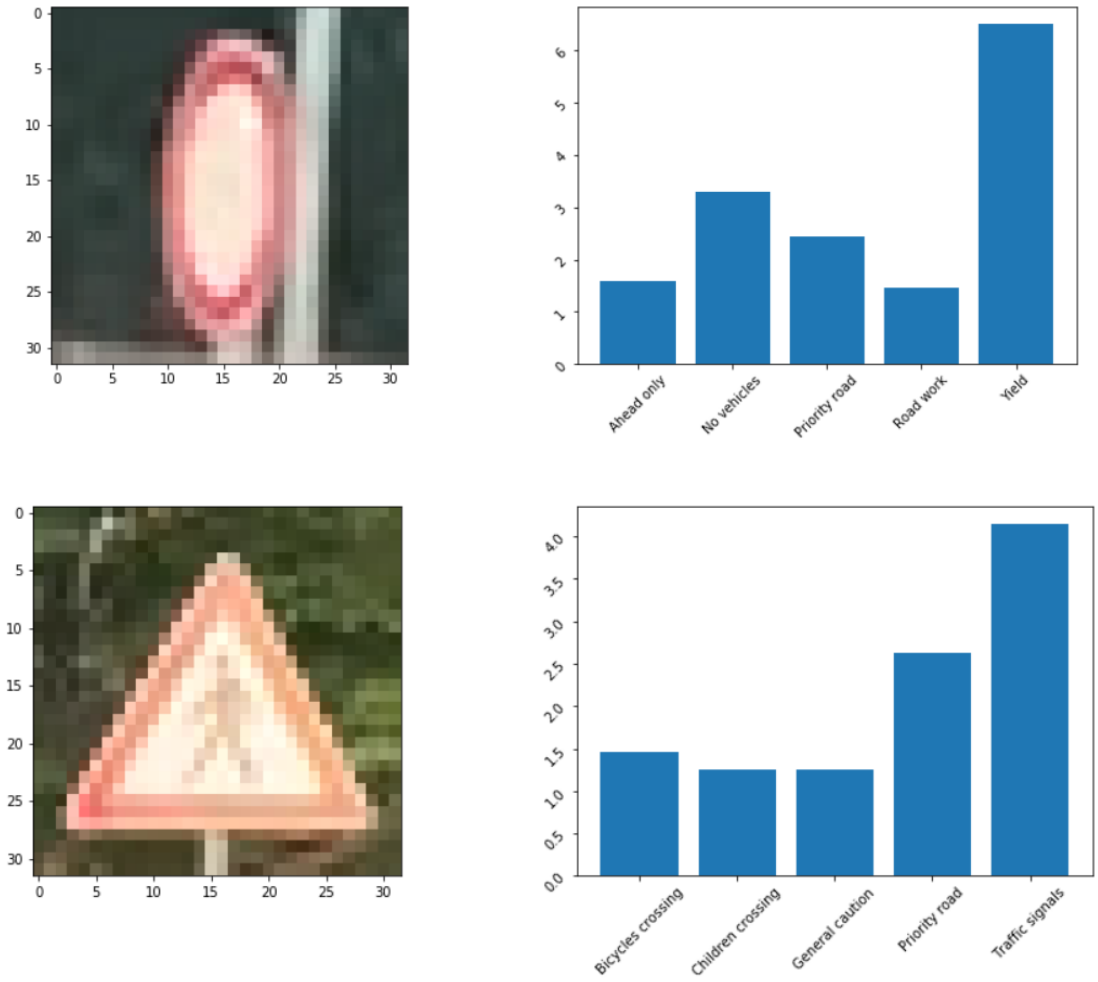

Figure 8 - Top 5 softmax on the misclassified images

4. Overall result and points of improvement

The convolutional neural network passed on the 93% accuracy on the validation dataset, as demanded by the rubric point. However, there is plenty of room for improvement. The datasets could be augmented by applying on the images a slightly rotation around 5° or so. Adding noise to the images may augment the dataset.

A more drastically approach is to change the architecture of the network entirely. A former student of the course, [mohamedameen93], posted in github his approach, in which he reaches 99% accuracy on the validation dataset using another convnet architecture: **VGGNet[3].** Other architecture might reach over 97% accuracy on the validation dataset.

The result on the web images was satisfactory. The images were took from a video, which is similar as how the code would run on a self-driving vehicle. However, it must be said that the images were not calibrated, so the distortion might play a small role on identifying the image. As for the two misclassified images, the error on the "**No vehicle**" sign is not surprising. There are some possible reasons for it:

- The sign is somewhat twisted, so the sign is harder to identify. A more robust algorithm on a self-driving car shall be able to apply a perspective transformation to see the sign right in front
- There is a very similar sign, the "**Yield"** sign. The only difference is on its shape, whereas one is a circle and the other is a triangle. The angle of the image contributes significantly on this misunderstand
- The number of signs of this class on the dataset is significantly smaller than the yield sign, as shown in image (large and small red bars on Figure 9).

The misclassification on the "**Pedestrian**" sign might have similar reasons:

- The number of signs of this class on the dataset is dramatically small. This makes the "**Pedestrian**" sign tougher to identify (small yellow bar on Figure 10).
- There is much brightness on the center of the sign, which is exactly where lies the difference between "**Pedestrian"** and "**Traffic signals"** signs. It is possible that a preprocessing adjusting brightness might solve this issue

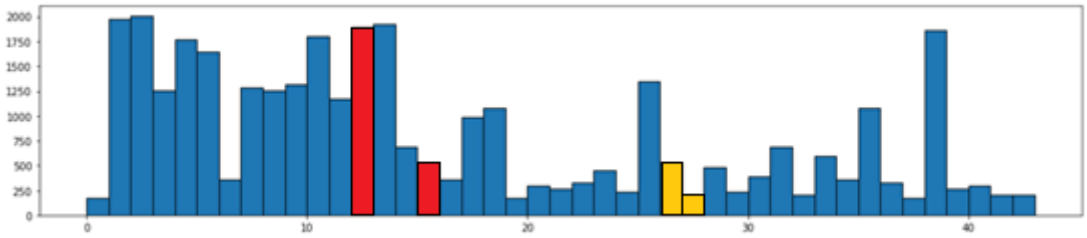

Figure 11 - Misclassified images proportion on the histogram. Yield and No Vehicle in red (large and small) and Traffic signals and Pedestrian (large and small)

As a side note, I would like to thank mohamedameen93 for posting his code on github as it is very clear and very helpful.

5. Links

1. http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset
2. https://www.youtube.com/watch?v=QfnkMGgvFGs
3. https://github.com/mohamedameen93/German-Traffic-Sign-Classification-Using-TensorFlow