Traffic Sign Recognition Program

Udacity CarND

Writeup - Daniel Alejandro Reyna Torres

In this project, goal is to write a software pipeline to identify and recognise traffic signs.

The Project

The goals / steps of this project are the following:

- Load the data set
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

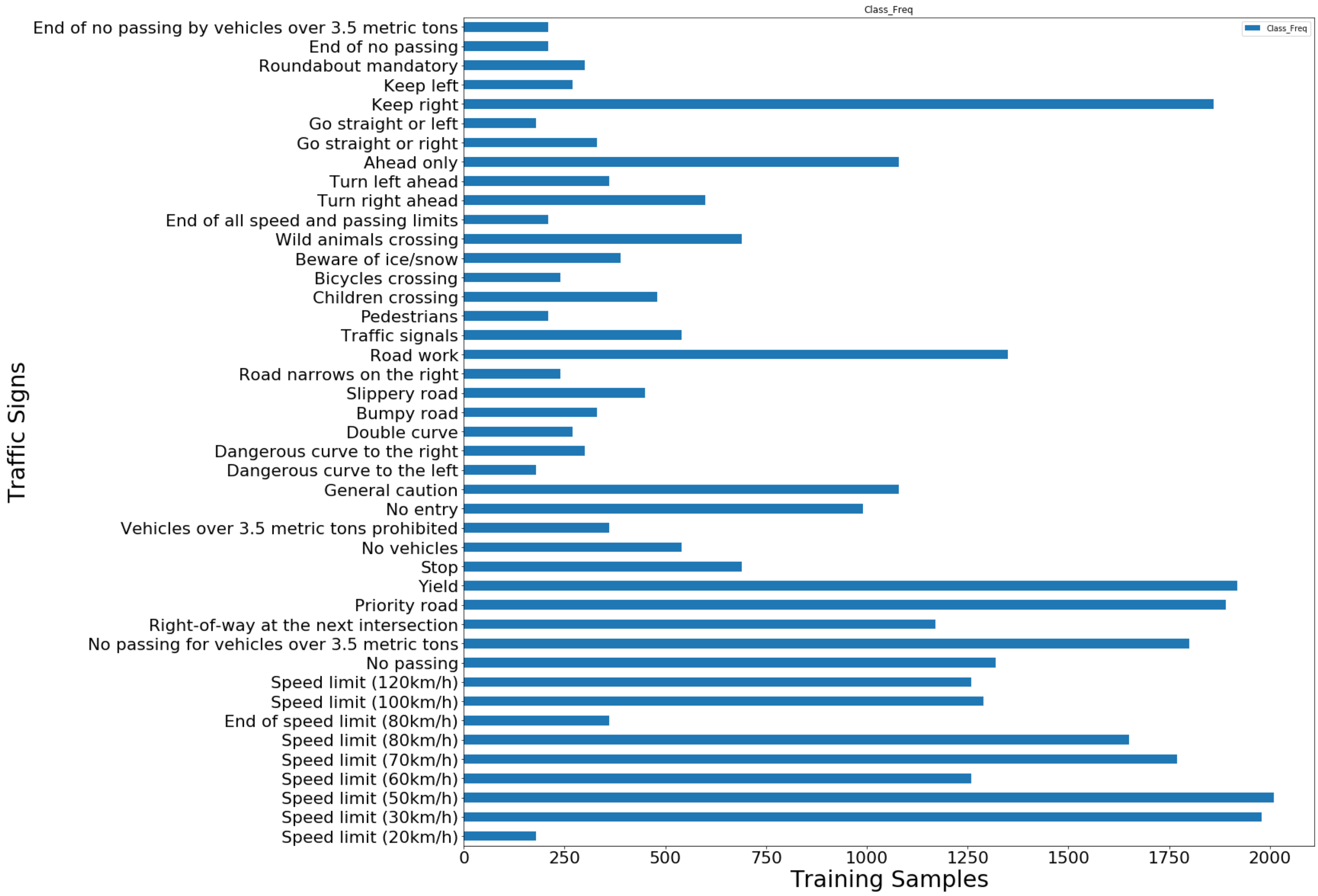
Data Set Summary & Exploration

The very fist step in every Machine Learning task is to load and understand the data. The data set corresponds to traffic signs from the German Traffic Sign Dataset. I used the numpy and pandas libraries to calculate summary statistics of the traffic signs data set:

Above is an **exploratory visualization** of the training set. Summary of data is:

- Number of training examples = 34799
- Number of testing examples = 12630
- Number of validating examples = 4410
- Image data shape = (32, 32, 3)
- Number of classes = 43

Here is an exploratory visualization of the data set. It is a bar chart showing how the amount of samples for each traffic sign is distributed.



If we take a closer loog on the data set we can see the following:

	Class_Freq	Sign_Name		Class_Freq	Sign_Name
2	2010	Speed limit (50km/h)	**0**	180	Speed limit (20km/h)
1	1980	Speed limit (30km/h)	**19**	180	Dangerous curve to the left
13	1920	Yield	**37**	180	Go straight or left
12	1890	Priority road	**27**	210	Pedestrians
38	1860	Keep right	**32**	210	End of all speed and passing limits
10	1800	No passing for vehicles over 3.5 metric tons	**41**	210	End of no passing
4	1770	Speed limit (70km/h)	**42**	210	End of no passing by vehicles over 3.5 metric ...
5	1650	Speed limit (80km/h)	**24**	240	Road narrows on the right
25	1350	Road work	**29**	240	Bicycles crossing
9	1320	No passing	**21**	270	Double curve

Traffic signs with most samples:

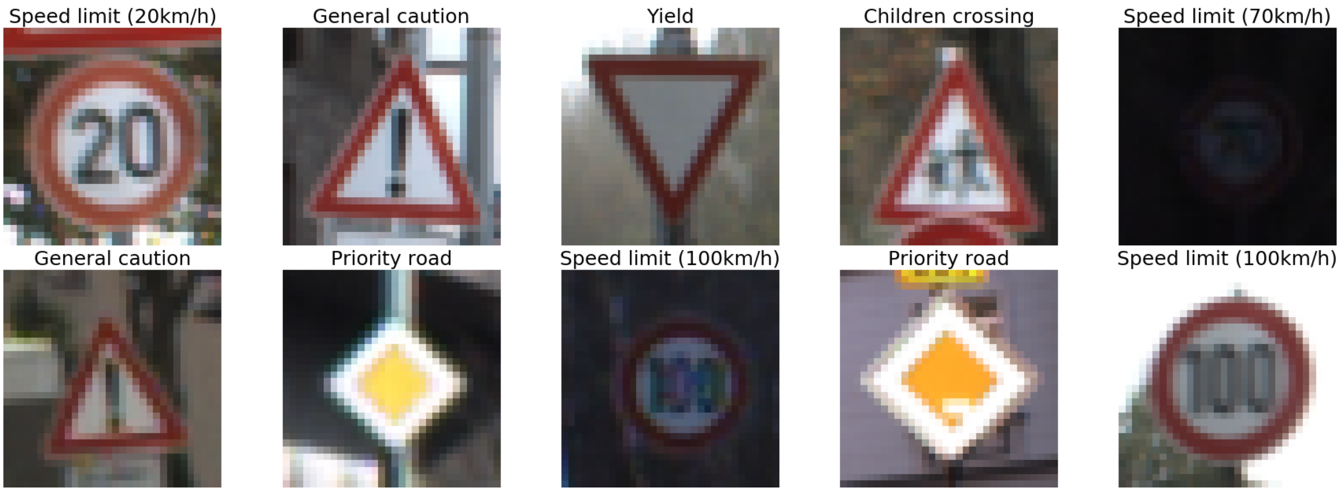
- Speed limit (50km/h) - 2010 samples
- Speed limit (30km/h) - 1980 samples
- Yield - 1920 samples
- Priority Road - 1890 samples
- Keep Right - 1860 samples

Traffic signs with fewer samples:

- Speed limit (20km/h) - 180 samples
- Dangerous curve to the left - 180 samples
- Go straight or left - 180 samples
- Pedestrians - 210 samples
- End of all speed and passing limits - 210 samples

It can be seen that there is an uneven number of samples for each traffic sign. Between the sign with most samples and the one with less samples, there are **1830** samples! This is something to consider in the design of the classification pipeline since this class imbalance could bring wrong classification results because the model would be reflecting the underlying class distribution.

Here are some samples from the data set.



Now, let's deep dive into our pipeline for traffic sign classification!

Design and Test a Model Architecture

Pre-process the Data Set

As a first step, I decided to convert the images to grayscale because it reduces model complexity and also because at the end, patterns, brightness, contrast, shape, contours shadows, and other image properties, are well captured by gray images without extra costs. Of course, use of coloured images will depend mainly on the task we want to solve,whether we need the extra information provided by the RGB channles or not will be part of the approximation.

Luma represents the brightness in an image ("black-and-white" of an image). Althought Luma can be computed in different manners, depending on the sensibility, the Luma component (gray scale transformation), based on luminance considering different human perception/sensibility towards RGB colors, for this project is computed as:

`Y'=0.299 R + 0.587 G + 0.114B`

Where: R, G, and B correpond to the information of RGB-channels of each image. This color-to-grayscale process is addressed in the interesting paper by Kanan, 2010 that exposes the importance of color-to-grayscale convertion for image recognition.

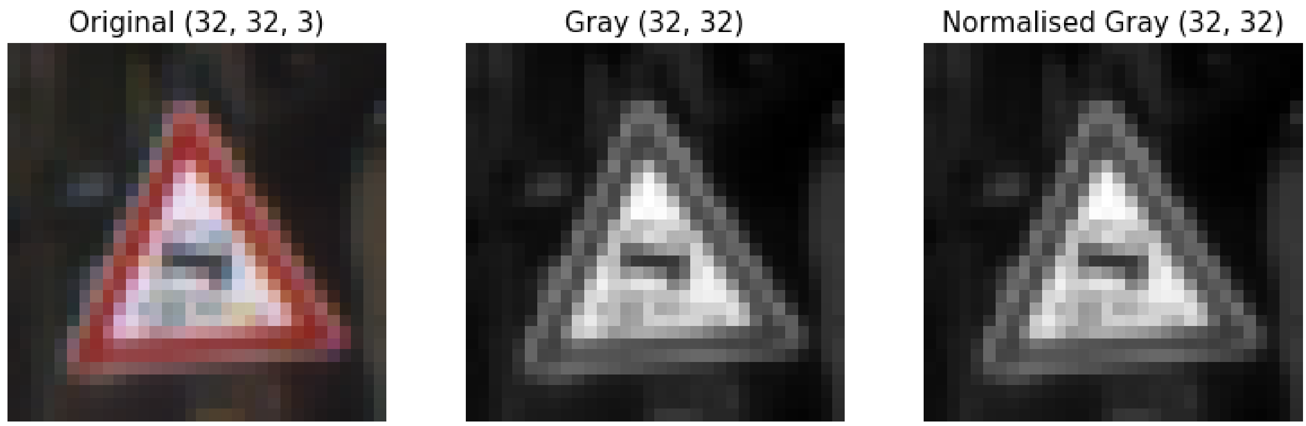
Now it's time to **normalise** our data! This will bring the input data to the same range of values and eliminates big variations across the data set. I decided to implement a Min-Max normalisation, images are scaled to a range of `a=0.1` and `b=0.9`.

Min-Max Normalisation:

$$X' = a + \frac{(X - X_{\min})(b - a)}{X_{\max} - X_{\min}}$$

Where: a and b are the scaling factors, X is the image and X' is the normalised image.

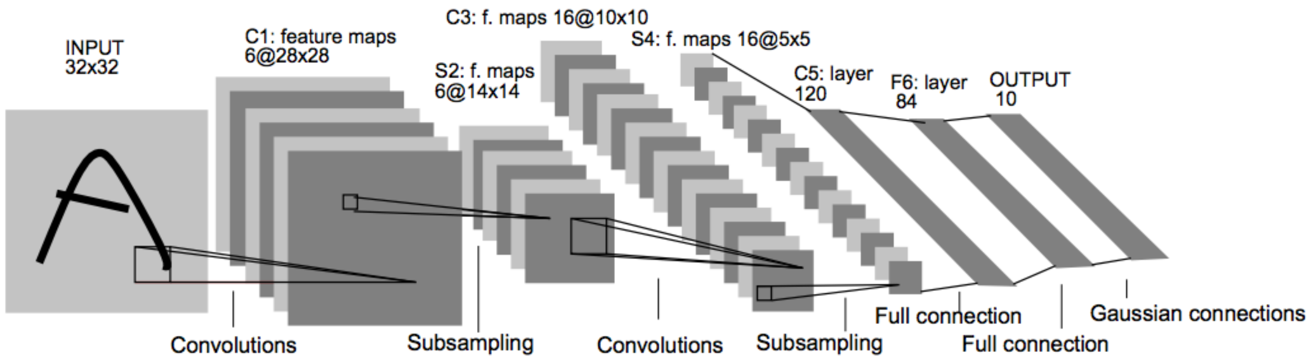
Results after color-to-grayscale convertion and normalisation shown below. Now images have shape of 32x32 instead of 32x32x3:



Model Architecture

Training with coloured images and using the original LeNet-5 architecture introduced by [LeCun et al. in their 1998]. This yield to an accuracy of 88% without any changes, seems to be a good start. Let's try to improve it.

Architecture of LeNel-5:



Model Training

To train the model, at the very beginning, I set the learning rate to 0.001, a batch size of 128 and 50 EPOCHS. This achieved an accuracy about 87% using coloured images. The final hyperparameters used for training were:

- Batch size: 100
- Epochs: 100
- Learning rate: 0.001
- Mu: 0
- Sigma: 0.1
- Dropout rate: 0.75
- Color channels: 1

My final model consists of the following architecture:

Layer	Description
Input	32x32x1 GRAY image
Convolution 3x3	1x1 stride, same padding, outputs 28x28x8
RELU	Activation (Nonlinearity)
Max pooling	2x2 stride, outputs 14x14x6
Convolution 3x3	1x1 stride, same padding, outputs 10x10x16
RELU	Activation (Nonlinearity)
Max pooling	2x2 stride, outputs 5x5x16
Flatten	Flatten the output shape 3D->1D
Fully connected	Array of 120 elements
RELU	Activation (Nonlinearity)
Dropout	Regularisation
Max pooling	2x2 stride, outputs 5x5x16
Fully connected	Array of 84 elements
RELU	Activation (Nonlinearity)
Dropout	Regularisation
Fully connected	Array of 43 elements (number of classes)
Softmax	Probabilities for each predicted class

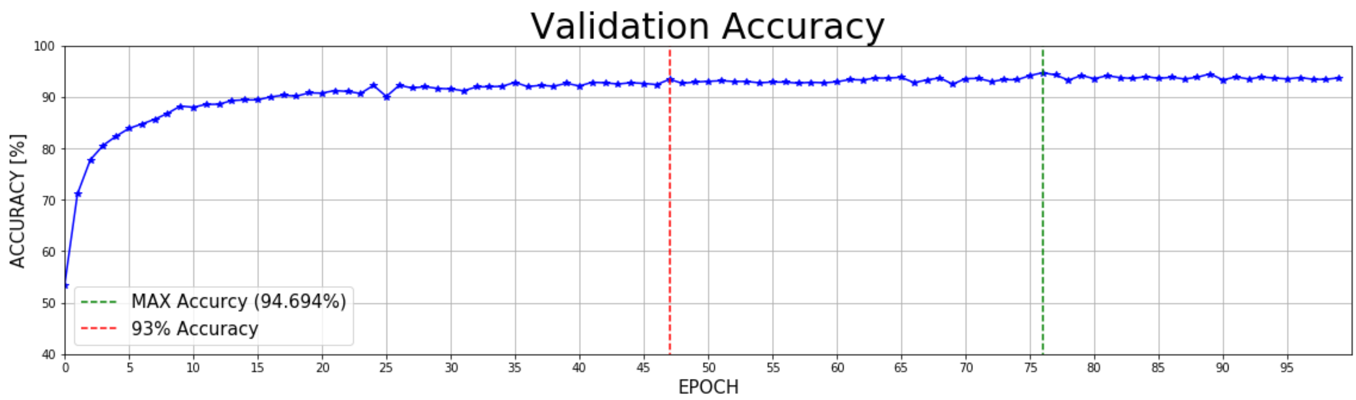
Solution Approach

As mentioned, the original LeNet-5 architecture was a very good start in order to classify traffic signs. Based on the LeNet-5, my final solution included the Adam optimizer (already implemented in the LeNet lab), added two dropout layers between the fully connected layers, a color-to-grayscale process and normalisation.

Final results:

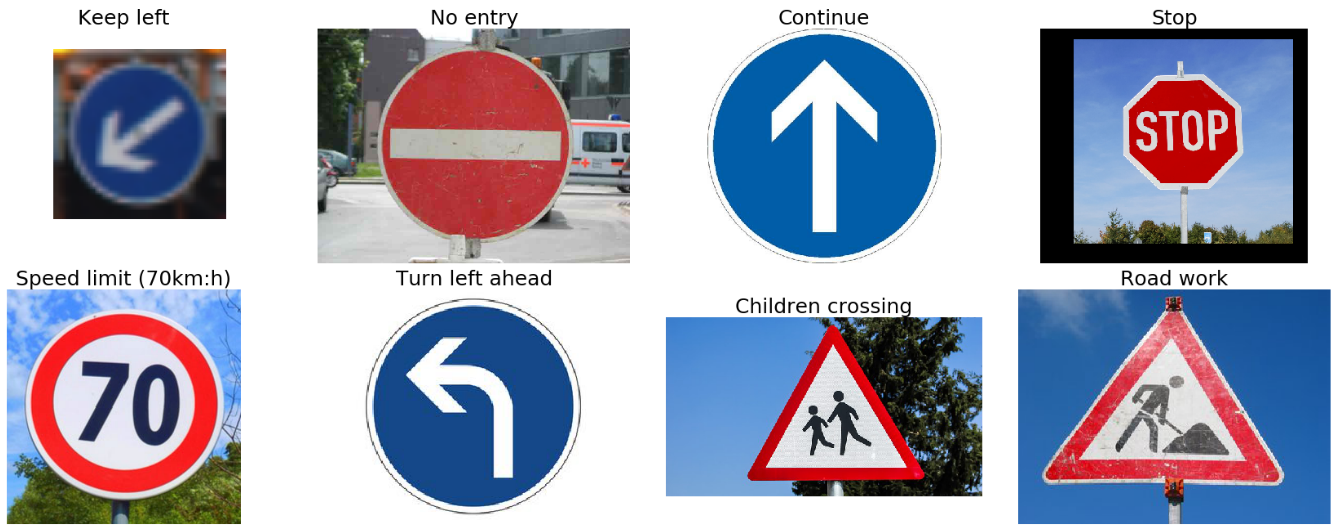
- Valid Accuracy: 93.696%
- Test Accuracy: 91.853%
- Test on New Images Accuracy: 87.5%

The next plot shows the validad accuracy for each epoch. After 47 epocs, classification accuracy goes above 93%.



Test a Model on New Images

Here are some German traffic signs that I found on the web:



Here are the results of the prediction:

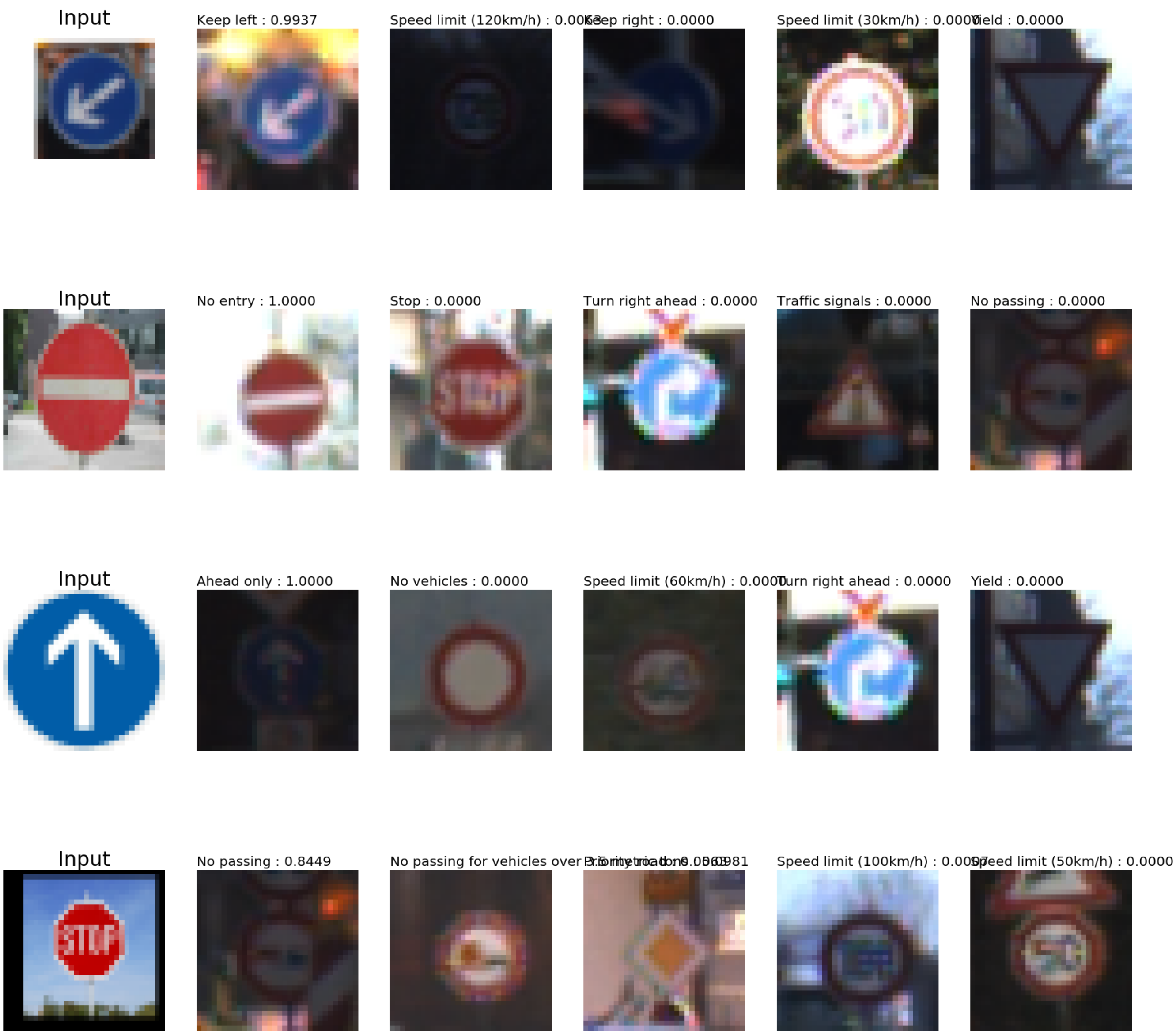
Image	Prediction
Keep left	Keep left
No entry	No entry
Continue	Continue
Stop	No passing
Speed limit (70km/h)	Speed limit (70km/h)
Turn left ahead	Turn left ahead
Children crossing	Children crossing

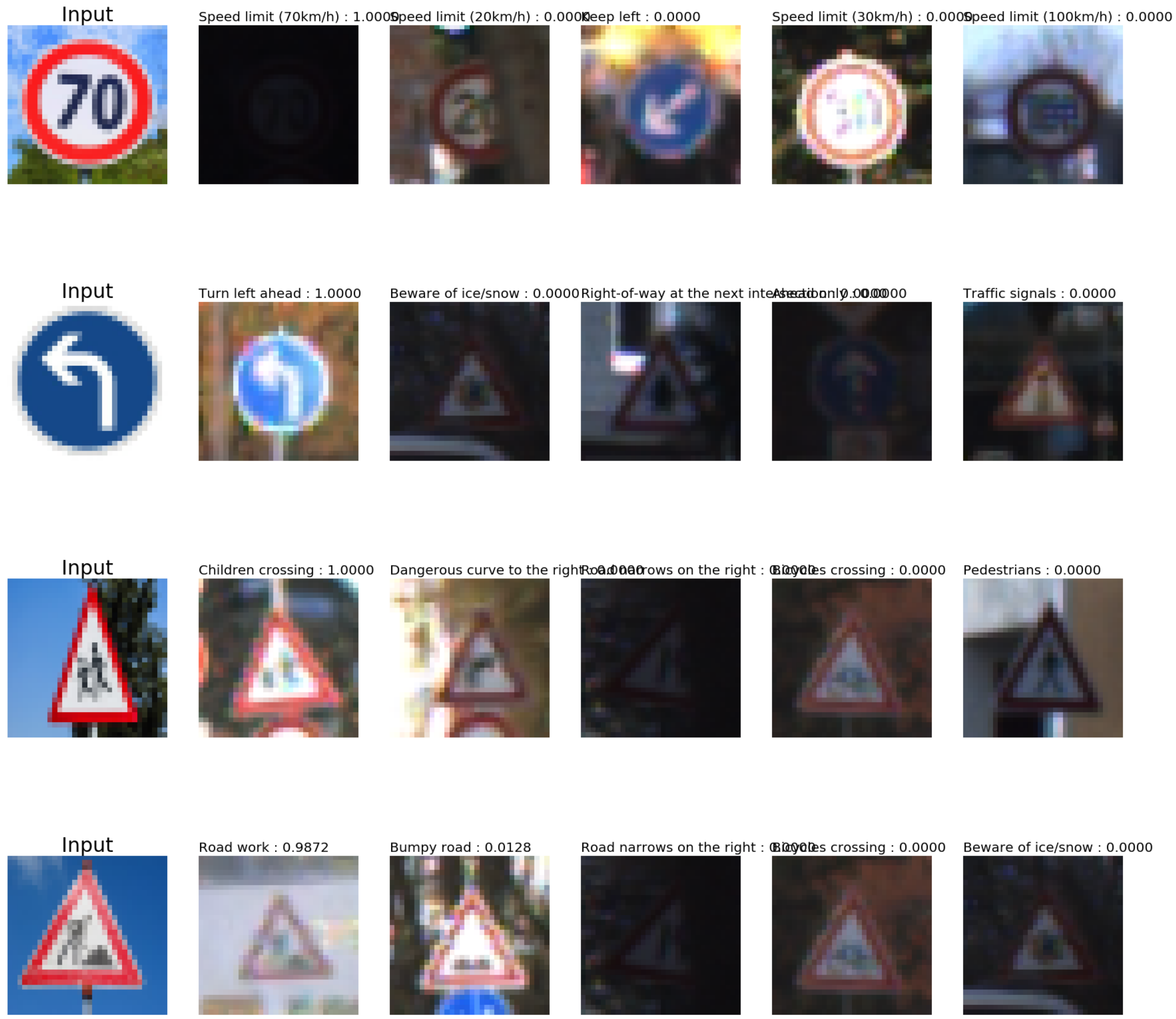
Image	Prediction
Road work	Road work

The model was able to correctly guess 7 of the 8 traffic signs, which gives an accuracy of 87.5%. For the first image, the model is 99.37% sure that this is a keep left! The top five soft max probabilities were

Probability	Prediction
.9937	Keep left
.0063	Speed limit (120km/h)
.0	Keep right
.0	Speed limit (30km/h)
.0	Yield

All images were in the same way analysed through the classification pipeline. For each row in the next image: the first image is the original input image and the rest are the top five softmax predictions.





Final Results

In order to achieve at least a 93% of accuracy, I did what mentioned in the Solution Approach section but also I changed the input size of 8 rather than the original 6 for the input layer.

Results developed like this:

- Adding dropout incremented the model accuracy in ~3.2% (90.2%)
- Increased batch size and epochs, model accuracy ~(91.2%)
- Changing the input depth to 8, model accuracy of ~92.3%
- Grayscale and Normalisation, model accuracy of ~93.7% (up to 94.69%)

Here's a link to my jupyter notebook pipeline. The folder `report_images` contains examples of the output from each stage of this pipeline.

Discussion

This project has been very interestin throughout all the excercise. One update I would do, would be to generate synthethic samples in order to diminish the class imbalance that exist in the data set used. This imbalance might have yield in a wrong classification for the stop sing. The stop sign class has about 1/3 of the samples than the keep right sign, which is the class with more samples.

Moreover, as mentioned, I changed the input size for the fist layer and results improved significantly. Thus, tunning hyperparameters and "playing" with the depth of the network,using stochastic gradient descent for optimisation or even a L2 regularisation would be interesting to run in order to compare outputs.

Thank you for reading this report.

--

Daniel