Udacity Self-Driving Car Nanodegree

Project 3: Traffic Sign Classifier

Hana Chew

This third project involves making use of Neural Networks to train a (German) Traffic Sign Classifier. The following files are submitted to meet the project requirements:

* the `Ipython` notebook with the code

* the code exported as an html file

* a write-up report either as a markdown or pdf file (this file)

The goals / steps of this project are set out as follows:

- 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
- Analyse the softmax probabilities of the new images
- Summarize the results with a written report

This report addresses each of the points to meet the above goals (further detailed in the project rubric), thus successfully completing the write-up / README requirement. The `Jupyter` Notebook is programmed pretty much to match up with this report, and is divided into the same sections and numbered the same way for ease of reference.

1. Dataset Exploration

Dataset Summary: The submission includes a basic summary of the data set

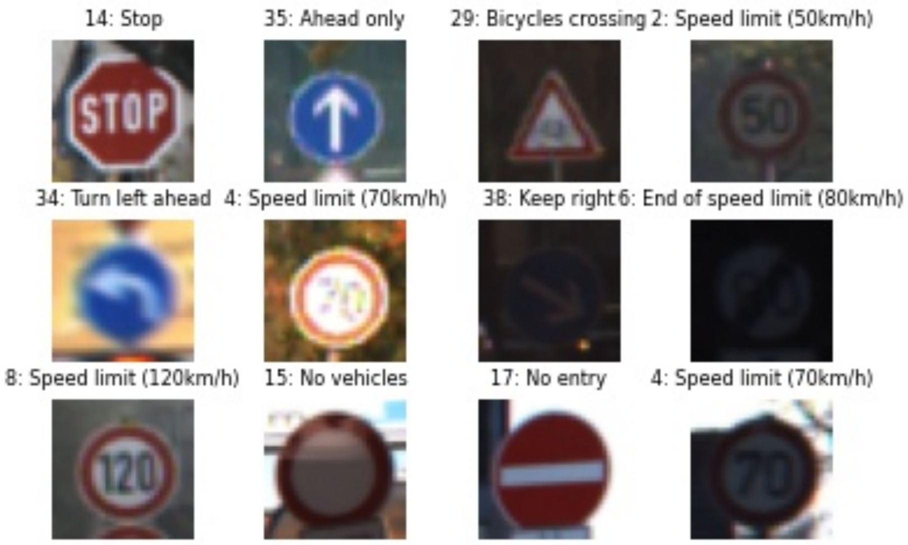

Figure 1: Sample images from the traffic signs dataset

Figure 1 shows the sample images from the dataset, on which the `numpy` library is used in order to calculate summary statistics of the traffic signs data set:

- The size of training set is 34799
- The size of the validation set is 4410
- The size of test set is 12630
- The shape of a traffic sign image is (32, 32, 3)
- The number of unique classes/labels in the data set is 43

Exploratory Visualization: The submission includes an exploratory visualization on the dataset.

Figure 2 shows the number of data samples for each of the classifications in the data sets provided. Additionally, a 3D histogram in Figure 3 shows the availability of a much larger number in the training set as compared to the validation or test set.

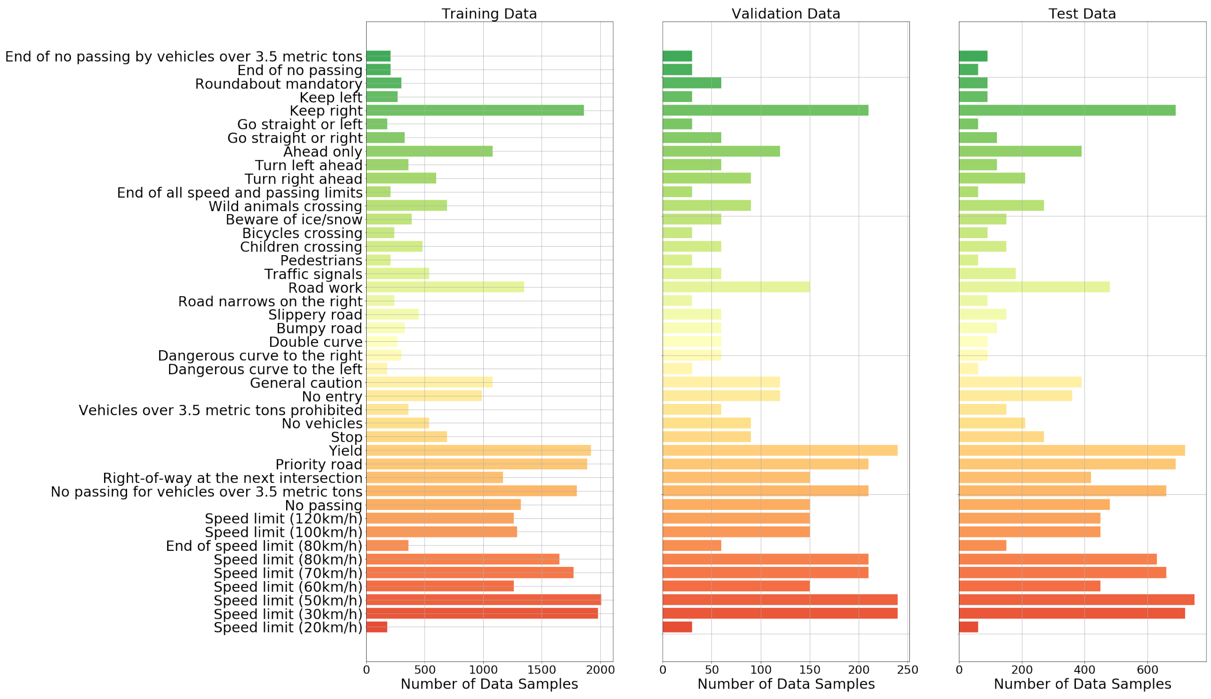

Figure 2: Number of Data Samples in each Dataset

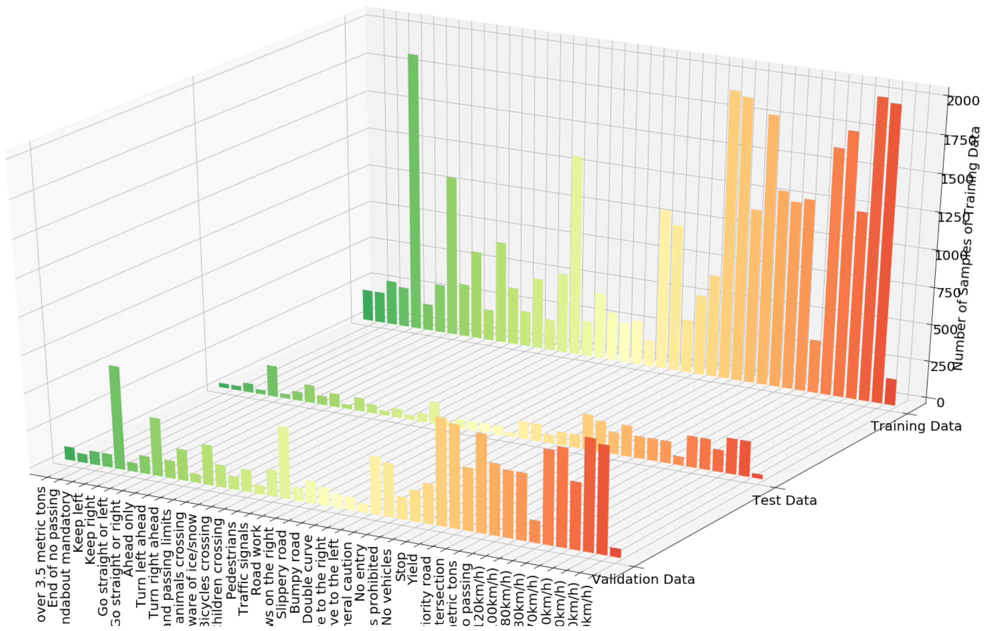

Figure 3: Number of Data Samples in each Dataset on one 3D projection

2. Design and Test a Model Architecture

2.1. Pre-processing

The submission describes the pre-processing techniques used and why these techniques were chosen.

First, the images were converted to grayscale. This improves the speed of the training later on, and the results of the training are not significantly lower than if full colour was used.

The next step involves normalizing the images] by use of the method provided in the lectures.

```
X_train_norm = ( X_train - 128 ) / 128
```

This step converts the data to a smaller deviation, with 0 being the mid-point making it easier to train. The mean of the datasets was now -0.35. After pre-processing, the following results were obtained for all the datasets:

```
              uint8      float64
X_train_min:    0   --->  -0.97 (normalised)
X_train_max:   255  --->   0.99 (normalised)
X_train_mean: 82.68 --->  -0.35 (normalised)

              uint8      float64
X_valid_min:    0   --->  -0.97 (normalised)
X_valid_max:   255  --->   0.99 (normalised)
X_valid_mean: 83.56 --->  -0.35 (normalised)

              uint8      float64
X_test_min:     0   --->  -0.97 (normalised)
X_test_max:    255  --->   0.99 (normalised)
X_test_mean:  82.15 --->  -0.35 (normalised)

X_train.shape            X_train_process.shape
(34799, 32, 32, 3)  ---> (34799, 32, 32, 1)
```

It can be seen that the pixel values were converted from a value of [0,255] to that of [-1,1], and that instead of having 3 colour channels (RGB), the data of size 32 x 32 pixels, now only has 1 channel after the grayscale transformation.

Figure 4 shows a random sampling of 8 traffic sign images before and after the pre-processing step grayscale conversion followed by normalization.

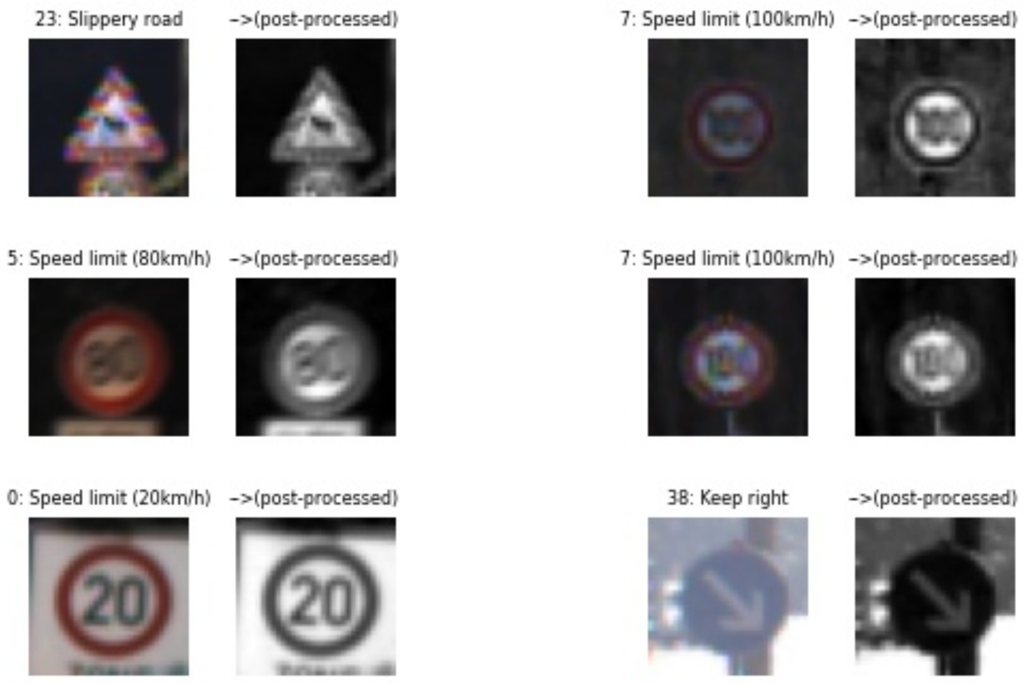

Figure 4: Original Image (left columns) vs Grayscale, Normalised Image (right columns)

2.2. Model Architecture

The submission provides details of the characteristics and qualities of the architecture, including the type of model used, the number of layers, and the size of each layer. Visualizations emphasizing particular qualities of the architecture are encouraged.

My final model architecture is that of the LeNet-5 Convolutional Neural Network postulated by Yann LeCun in the late 90s. Table 1 on the next page, combined with Figure 5 below, summarizes the various layers, including dropouts introduced in the final two full connection layers that were added to improve the model.

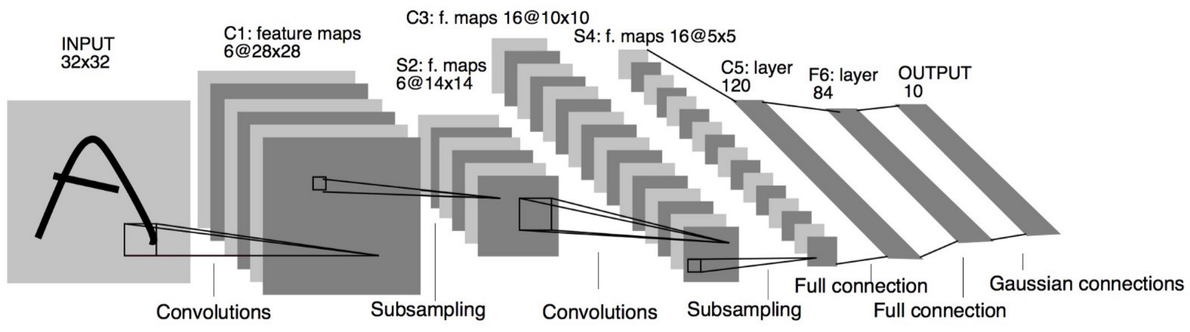

Figure 5: LeNet-5 Convolutional Neural Network where each plane is a feature map. Source: Lecun et al. Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, Dec 1998

	Stride Size	Input size	Output Size
Input Layer: 32 x 32 x 1 – grayscale image			
Layer 1: Convolution	1 x 1	32 x 32 x 1	28 X 28 x 6
Activation Layer (RELU)			
Pooling :	2 x 2	28 x 28 x 6	14 x 14 x 6
Layer 2: Convolution	1 x 1	14 x 14 x 6	10 x 10 x 16
Activation Layer (RELU)			
Pooling:	2 x 2	10 x 10 x 16	5 x 5 x 16
Flatten:		5 x 5 x 16.	400 (x 1)
Layer 3: Fully Connected		400 (x 1)	120 (x 1)
Activation Layer (RELU) – followed by dropouts (50%)			
Layer 4: Fully Connected		120 (x 1)	84 (x 1)
Activation Layer (RELU) – followed by dropouts (50%)			
Layer 5: Fully Connected.		84 (x 1)	n_classes (x 1)

Table 1: Layout of LeNet Architecture used in the model

2.3. Model Training

The submission describes how the model was trained by discussing what optimizer was used, batch size, number of epochs and values for hyperparameters.

The model was trained with the AdamOptimizer also used in the LeNet lab during the lessons, with the following hyperparameters:

```
epochs = 40                learnRate = 0.001
batchSize = 128            dropouts = 0.5
```

2.4. Solution Approach

The submission describes the approach to finding a solution. Accuracy on the validation set is 0.93 or greater.

My final model gave the following results:

```
Test Accuracy       = 0.935
Validation Accuracy = 0.953
Train Accuracy      = 0.997
```

The very first set tried was that of the barebones LeNet Architecture, with the following parameters:

```
epochs = 10
batchSize = 128
learnRate = 0.001
```

Table 2 that follows gives some of the results from trial runs to manually tune the parameters by comparing the results obtained.

Tuned Parameters			Accuracy of model on Datasets			
epochs	learnRate	keepProb	Training	Test	Validation	Comments
10	0.001	1	0.990	0.893	0.924	
40	0.001	1	1.000	0.921	0.942	
10	0.0005	1	0.971	0.856	0.868	
40	0.0005	1	0.891	0.886	0.995	
10	0.001	0.5	0.969	0.905	0.919	
40	0.0008	0.55	0.998	0.938	0.949	
40	0.001	0.5	0.997	0.944	0.964	--petered out at 15-20
20	0.001	0.5	0.991	0.933	0.960	
20	0.001	0.5	0.989	0.927	0.937	batchSize: 256
20	0.0008	0.5	0.988	0.930	0.946	
40	0.001	0.6	0.999	0.946	0.963	
20	0.001	0.5	0.991	0.933	0.960	
20	0.001	0.55	0.995	0.933	0.947	
40	0.001	0.55	0.997	0.928	0.949	
40	0.001	0.5	0.998	0.942	0.960	

Table 2: Parameters and the Resulting Accuracies

After an initial run, it was clear 10 epochs were not enough, but it was mainly for a quick start to get a rough idea as to how the tuned parameters would make a difference, and what tweaks would be required – or if further augmentation of the data would be necessary. It was quickly apparent that the model, updated with a dropout rate would give much better results, and thus the algorithm was modified. The batch size was increased, but it seemed to give slightly less accurate results, so 128 was kept.

The model was sufficient to give a result of >0.93 on the final run, without overfitting too much on the training set, so those were the results chosen for the scope of this project. The discussion section below shows the improvements should we require a higher accuracy.

3. Test a Model on New Images

3.1. Acquiring New Images

The submission includes five new German Traffic signs found on the web, and the images are visualized. Discussion is made as to particular qualities of the images or traffic signs in the images that are of interest, such as whether they would be difficult for the model to classify.

Figure 6 shows the resized versions of six German traffic signs found on the web. The upper images are 100 x 100 pixels, while the lower ones are that required by the classifier, 32 x 32 pixels.

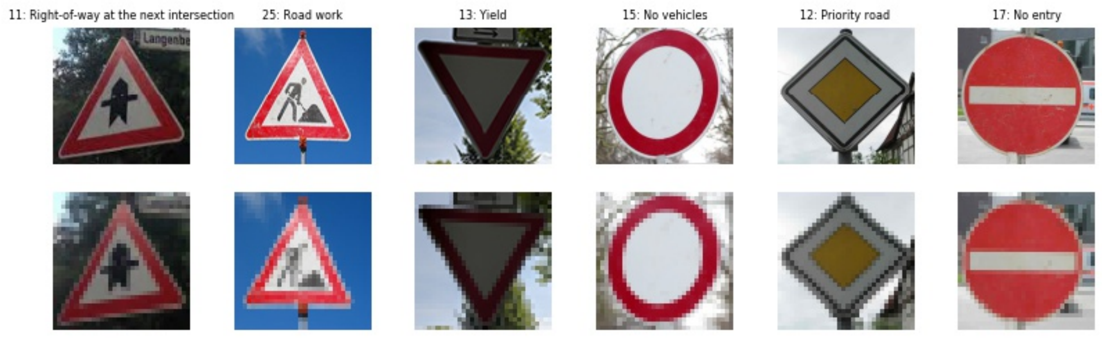

Figure 6: Hi- and Low-Res (32 x 32px) Images of the German Traffic Signs

The images are all relatively easy to classify. There could be issues on some traffic signs where there are multiple images next to each other, or even some signs that are not in the database so far!

3.2. Performance on New Images

The submission documents the performance of the model when tested on the captured images. The performance on the new images is compared to the accuracy results of the test set.

The model was able to correctly guess all of the traffic signs, giving a 100% test accuracy! This is of course ideal, but since it is such a small sample size, cannot be compared to the accuracy when applied to the test set.

```
         actual predicted
Correct!!  11      11: Right-of-way at the next intersection
Correct!!  25      25: Road work
Correct!!  13      13: Yield
Correct!!  15      15: No vehicles
Correct!!  12      12: Priority road
Correct!!  17      17: No entry
```

Model Certainty - Softmax Probabilities

The top five softmax probabilities of the predictions on the captured images are outputted. The submission discusses how certain or uncertain the model is of its predictions.

The model predicts the different traffic signs rather accurately, having almost a 100% accuracy for each, as can be seen by the top five softmax probabilities for each image, as well as the histograms in Figure 7.

```
INFO:tensorflow:Restoring parameters from lenet
TopKV2(values=array([[  9.99921918e-01,   4.33876121e-05,   2.35343614e-05,
          8.11977225e-06,   1.12508201e-06],
       [  7.36336827e-01,   2.48007372e-01,   1.19879302e-02,
          1.23718416e-03,   1.08138286e-03],
       [  1.00000000e+00,   4.40171011e-18,   2.88518669e-22,
          1.41312756e-24,   1.09682926e-28],
       [  9.99997020e-01,   2.77208483e-06,   1.02017374e-07,
          3.31652643e-08,   1.64066627e-08],
       [  1.00000000e+00,   9.89085758e-11,   1.00843097e-16,
          4.27884980e-18,   1.32234381e-26],
       [  1.00000000e+00,   7.16849383e-16,   2.66718923e-16,
          2.75359379e-19,   3.29421092e-20]], dtype=float32), indices=array([[1
1, 27, 40, 12, 42],
       [25, 22, 30, 29, 26],
       [13, 12,  9, 15, 35],
       [15, 12, 13, 26,  2],
       [12, 40, 32, 41, 27],
       [17, 40, 14, 36,  0]], dtype=int32))
```

Summarising these values for the top 3 probabilities, Table 3 is given below.

No	Most Likely Result	Second Most Likely Result	Third Most Likely Result
1	11: Right-of-way at the next intersection 99.999%	27: Pedestrians 0.004%	40: Roundabout mandatory 0.002%
2	25: Road Works 73.633%	22: Bumpy Road 24.800%	30: Beware of ice/snow 1.200%
3	13: Yield ~100%	12: Priority road << 0.001%	9: No passing << 0.001%
4	15: No vehicles 99.999%	12: Priority road < 0.001%	13: Yield < 0.001%
5	12: Priority road ~100%	40: Roundabout mandatory << 0.001%	32: End of all speed and passing limits << 0.001%
6	17: No entry ~100%	40: Roundabout mandatory: << 0.001%	14: Stop << 0.001%

Table 3: Softmax Probabilities of Predictor on Test Images

As can be seen, the softmax values are very good predictors of how good the model is, with very high values for the correct label, and almost negligible values for the wrong ones, at least with this training set, also obvious in Figure 6a.

Only in the second image is there is a 24.8% probability of another sign being predicted – in this case, road works and a bumpy road are both signs that require a reduction in speed, so it would not lead to too dire consequences should the mistake be made. However, it clearly shows the model needs to be further worked on. "Yield", "Priority Road" and "No entry" signs are also extremely distinguishable, probably with the former two have such distinct shapes compared to the standard round and upright triangular formats.

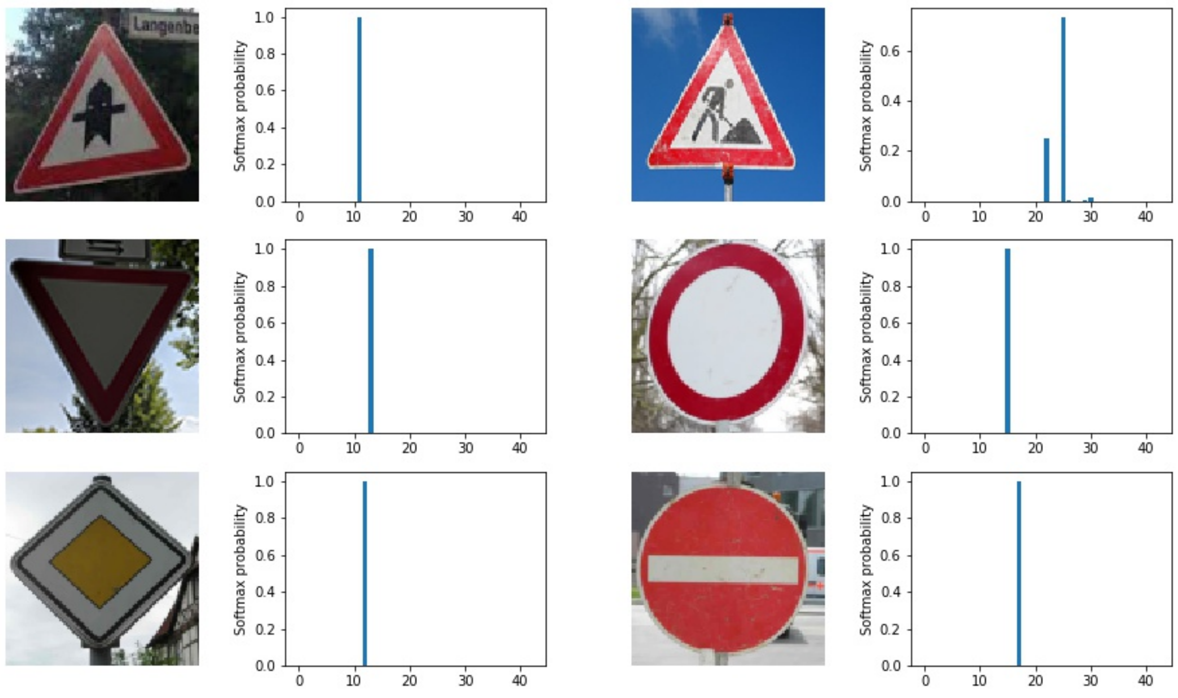

Figure 7: Softmax distribution probabilities of the Test Images

4. Discussion

The most tedious part of solving this project was trying various different values of the parameters to tune the LeNet model in order to get better results, and the extremely long time it takes to converge without use of the GPU (on a local machine). There were dozens of times that I thought the perfect balance when it came to the validation set, but then when applied on the test data, was not as good as I had expected.

There are a number of issues which could be addressed given more time.

I. The tweaks and corrections to be made to the model are limitless and could spend hours of involvement – the process could be automated over the course of a few days with a script and a rubric implementing various values of the parameters for a fixed data structure, however, this is slightly beyond the scope of the assignment for now.

II. The results of this classifier was already enough to meet the requirements set forth (>0.93 accuracy on the validation set), but it could be made very much more accurate with simple tweaks to the images of the traffic signs: Traffic signs of different sizes, orientations and even brightness could be simulated from the standard dataset and be used concurrently in order to augment the data. The greater the training set, the better the results on the validation and test models – and eventually on real life models will be, because in real life, the same sign could appear different in the eyes of a classifier day to day by simple shifts in weather or time of day.

III. The model may also be fine-tuned with specific performance indicators of each of the classifiers or labels, and this will be further explored