Traffic Sign Recognition Project

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Data Set Summary & Exploration

1. Provide a basic summary of the data set.

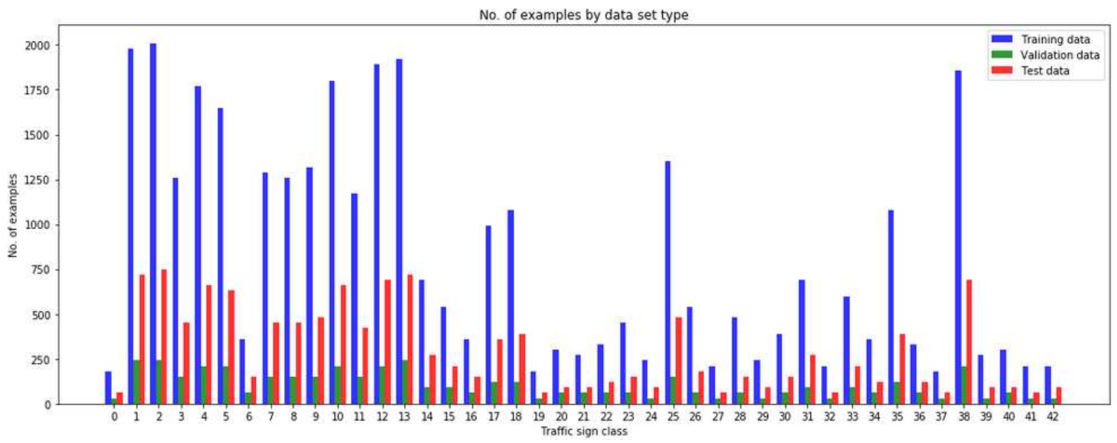
I used the numpy library to calculate summary statistics of the traffic signs data set:

- Number of training examples = 34,799
- Number of validation examples = 4,410
- Number of testing examples = 12,630
- Image data shape = (32, 32, 3)
- Number of unique classes = 43

2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. First just simple plotting of 5 random images from the training data set:

Secondly, I added an overview of traffic sign classes distribution. It shows that the classes are not distributed evenly - I guess it might have an impact on the learning process:



Design and Test a Model Architecture

1. Describe how you preprocessed the image data.

As a first step, I decided to convert the images to grayscale, under the assumption that RGB colors are not vital for image recognition. However, my model's accuracy was lower when training on grayscaled images. I suspect that's because of the key difference between blue, white and red color on the signs. Therefore I decided to skip grayscaling.

As a result, the pre-processing in my model is limited to normalization and data shuffling.

2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.)

My final model consisted of the following layers:

Layer	Description
Input	32x32x3 RGB image
Convolution	1x1 stride, valid padding, outputs 28x28x6
RELU	
Max pooling	2x2 stride, outputs 14x14x6
Dropout	Keep probability = 80%
Convolution	1x1 stride, valid padding, outputs 8x8x16
RELU	
Max pooling	2x2 stride, outputs 4x4x16
Dropout	Keep probability = 80%
Flattening	Output 4x4x16 = 256
Fully connected	Output = 120
RELU	
Dropout	Keep probability = 80%
Fully connected	Output = 60

RELU	
Dropout	Keep probability = 80%
Fully connected	Output = 43

3. Describe how you trained your model.

I used batch size = 128 (same as the original LeNet), learning rate = 0.001 (according to the rule that lower is better, at least at the beginning) and a slightly higher number of epochs than LeNet (12). Actually, most of the time I used only 10 epochs (to shorten the time waiting for the model training to be finished). Only towards the end of the "trial and error" tweaks I increased the number of epochs to 12.

I used AdamOptimizer, as according to the tutorial video it's more sophisticated than the SGD and therefore a good choice :-)

4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93.

My final model results were:

- training set accuracy of 98.1%
- validation set accuracy of 93.2%
- test set accuracy of 92.3%

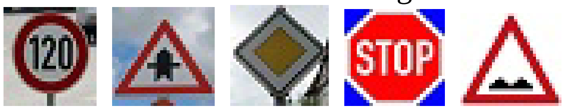
An iterative approach was chosen:

- The first architecture I tried was pure LeNet with just the necessary dimensioning adjustments. The results were: validation accuracy = 0.892, training accuracy = 0.979 - suggesting an overfitting and quite far from the goal of at least 0.93 validation accuracy.
- Then I applied grayscaling - under the assumption that it will make the network's job easier - and to my surprise the validation accuracy dropped to 0.867. I suspect it's because RGB colors are important differentiators after all (e.g. difference between red vs. yellow traffic sign framing vs. blue background). Additionally, it's probably the network was overfitted anyway, so there was no need to make its job "easier". Therefore, I decided to skip grayscaling.
- Then in a couple of steps I experimented with adding convolutional and/or fully connected layers. This did not help, on contrary these changes decreased validation accuracy to just around 0.813. I suspect the reason is that additional layers increased overfitting which was already there anyway.
- Therefore, I decided to go in the other direction and apply dropout. First attempt with the keep rate of 0.5 resulted in a very low validation accuracy (0.567).
- Increasing keep rate to 0.75 increased the validation accuracy to 0.927. I was getting close to the solution.
- In a series of following steps I experimented with changing the dimensions of convolutional and fully connected layers. Finally I found that slightly lower dimensions than LeNet, combined with the keep probability (dropout) of 0.8 yielded the targeted validation accuracy above 0.93 (0.932)

Test the Model on New Images

1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.

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



I expected that all five will be easy to classify because they are very clear, clearer then the images that I have seen in the training set.

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set.

Here are the results of the prediction:

Image	Prediction
8 ,Speed limit (120km/h)	7, Speed limit (100km/h)
11,Right-of-way at the next intersection	11,Right-of-way at the next intersection
12, Priority road	12,Priority road
14, Stop	14, Stop
22, Bumpy road	22, Bumpy road

The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%. This is a lower accuracy than the test set accuracy of 92.3%

3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)

The code for making predictions on my final model is located in the 18th cell (and below) of the Ipython notebook.

Interestingly, for all images the model is very sure of its predictions (see the overview below). Even in case of image 8 (speed limit 120 km/h) - where the model was wrong - it still "believes" to have a 95.6% certainty. So basically the model is 95.6% certain that the 120 km/h speed limit is actually 100 km/h speed limit (#7). What is even more interesting, the second-highest probability is allocated to #5 (speed limit 80 km/h). Only third and negligible probability (0.41%) is allocated to the correct prediction (#8).

This high certainty - even in case the model is wrong - is worrying. I would like to understand how is it possible and what could be done to improve this behavior.

My initial conclusion - after this 5-image test - would be that the model is not good at recognizing different speed limits.

```
Image 12 (Priority road):
Image type: 12 (Priority road)          --> probability: 99.69%
Image type: 14 (Stop)                   --> probability: 0.28%
Image type: 17 (No entry)               --> probability: 0.02%
Image type: 10 (No passing > 3.5 tons)  --> probability: 0.01%
Image type: 9  (No passing)             --> probability: 0.00%

Image 22 (Bumpy road):
Image type: 22 (Bumpy road)             --> probability: 100.00%
Image type: 29 (Bicycles crossing)      --> probability: 0.00%
Image type: 24 (Road narrows on right)  --> probability: 0.00%
Image type: 18 (General caution)        --> probability: 0.00%
Image type: 26 (Traffic signals)        --> probability: 0.00%

Image 8 (Speed limit (120km/h)):
Image type: 7 (Speed limit (100km/h))   --> probability: 95.61%
Image type: 5 (Speed limit (80km/h))    --> probability: 3.25%
Image type: 8 (Speed limit (120km/h))   --> probability: 0.41%
Image type: 2 (Speed limit (50km/h))    --> probability: 0.37%
Image type: 1 (Speed limit (30km/h))    --> probability: 0.18%

Image 14 (Stop):
Image type: 14 (Stop)                   --> probability: 98.08%
Image type: 29 (Bicycles crossing)      --> probability: 1.32%
Image type: 17 (No entry)               --> probability: 0.30%
Image type: 25 (Road work)              --> probability: 0.11%
Image type: 22 (Bumpy road)             --> probability: 0.10%

Image 11 (Right-of-way at the next intersection):
Image type: 11 (Rght-f-wy nxt intrsctn) --> probability: 100.00%
Image type: 30 (Beware of ice/snow)     --> probability: 0.00%
Image type: 27 (Pedestrians)            --> probability: 0.00%
Image type: 21 (Double curve)           --> probability: 0.00%
Image type: 42 (End of no passing > 3.5 tons)--> probability: 0.00%
```