Traffic Sign Recognition

Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:

1) Load the data set (see below for links to the project data set)
2) Explore, summarize and visualize the data set
3) Design, train and test a model architecture
4) Use the model to make predictions on new images
5) Analyze the softmax probabilities of the new images
6) Summarize the results with a written report

Rubric Points

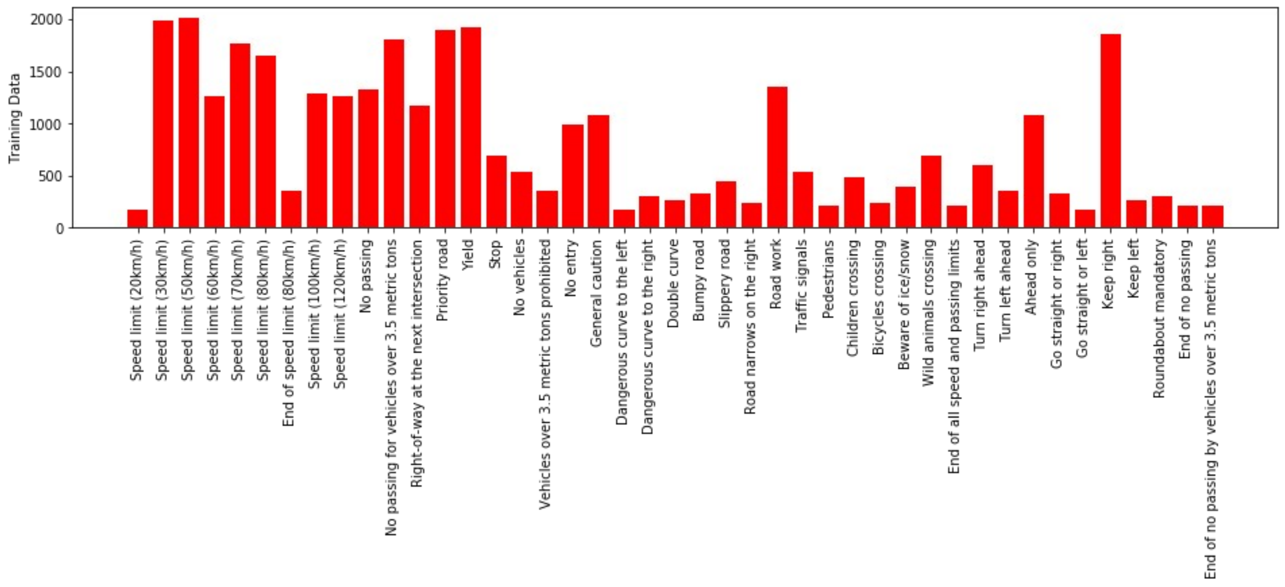
Data Set Summary & Exploration

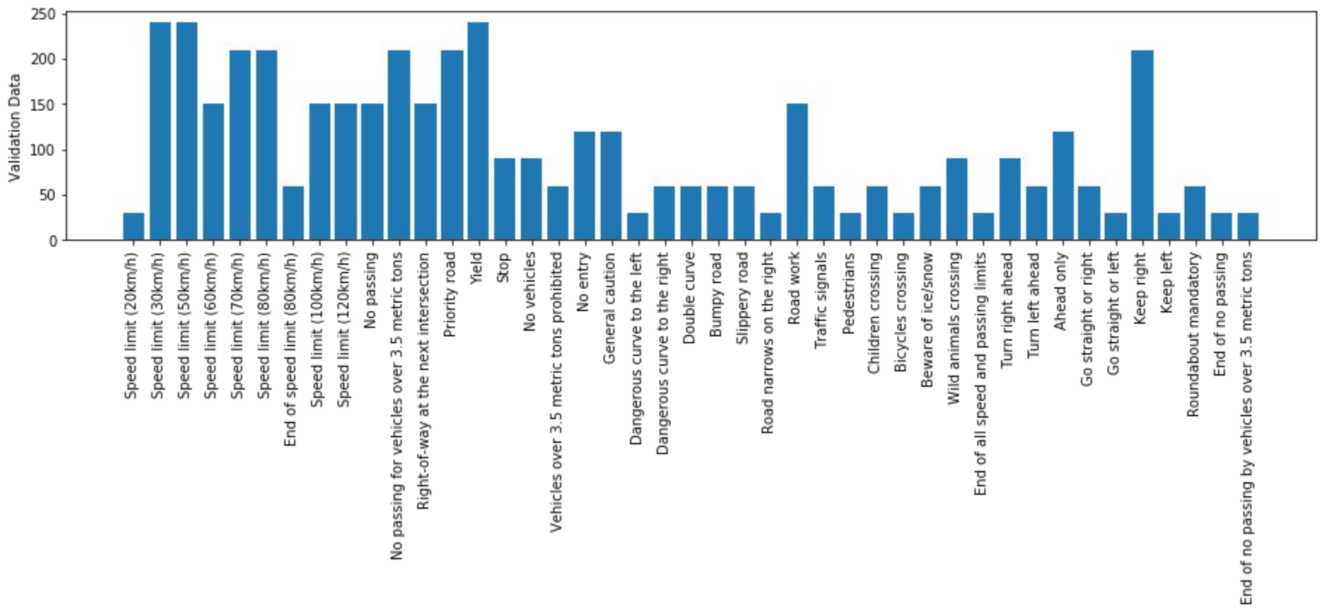
1. I used the numpy library to calculate summary statistics of the traffic signs data set:

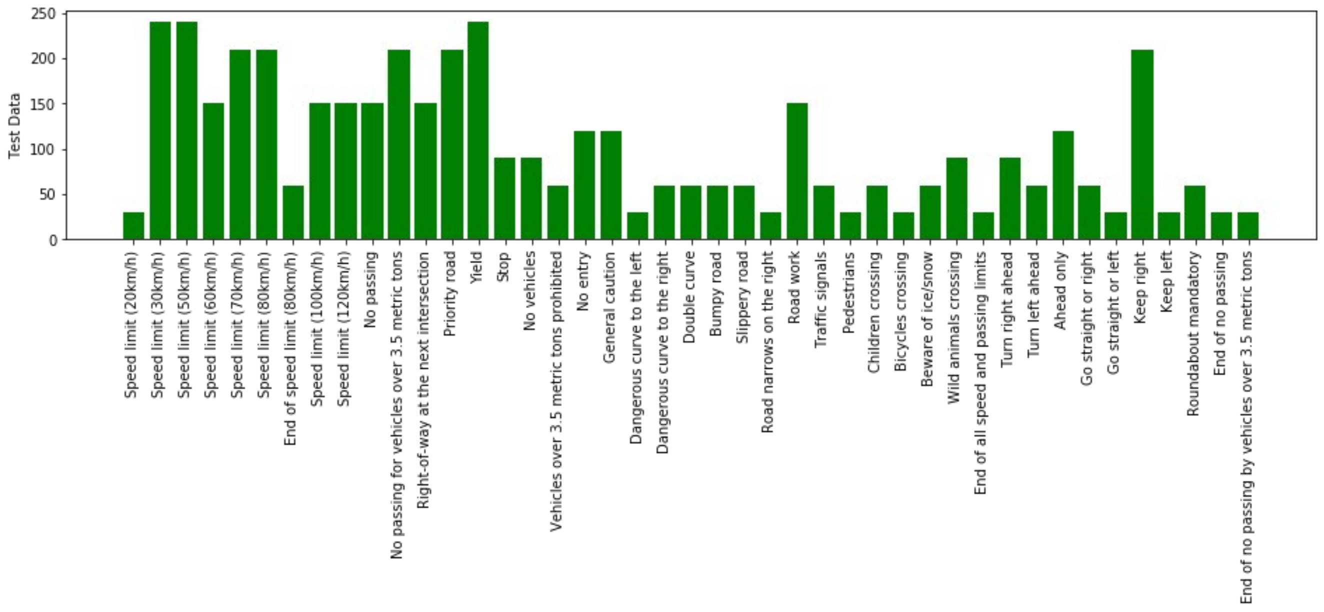
- The size of training set is 34799
- The size of test set is 12630
- The shape of a traffic sign image is (32, 32, 3)
- The number of unique classes/labels in the data set is 43

The code for this step is contained in the second code cell of the IPython notebook.

2. Here is an exploratory visualization of the data set. It is random image from the training set and the number of labels per set.







The code for this step is contained in the third , forth and fifth code cell of the IPython notebook.

Design and Test a Model Architecture

1. As a first step, I decided to convert the images to grayscale because i consider only using information about geometry in the image instead of color information. A blue or red dog is stills a dog. Also in this way i won't have to add extra CNN layers to process de color information. Then i used histogram equalization in order to improve the contrast of the images. This method was recommended and i tested it obtaining good results. The images of the data set need it image processing to obtain a better result or

to add CNN layers to correct the imperfections of the images. Finally i normalize the image in order to the recommendations in the lessons of Neural networks and i made a reshape of the image image = image.reshape((32,32,1)).

The code for this step is contained in the sixth code cell of the IPython notebook.

2. My final model consisted of the following layers:

Layer	**Description**
Input	32x32x1 Gray scale image
Convolution 5x5	1x1 stride, valid padding, outputs 28x28x6
RELU	
Max pooling	2x2 kernel, 2x2 stride, outputs 14x14x6
Convolution 5x5	1x1 stride, valid padding, outputs 10x10x16
RELU	
Max pooling	2x2 kernel, 2x2 stride, outputs 5x5x16
Flatten	
Fully connected	400 input, output = 120.
RELU	
Fully connected	400 input, output = 120.

The code for my final model is located in the cell number eight of the ipython notebook.

3. To train the model, I used the LeNet configuration because i consider it a good option to start and to use it in a more complicated example.

Then i apply the concepts we saw in the Lesson of Neural Networks. I need it to calculate softmax and compare the result of the probability vector with the actual vector of desires outputs. Then I choose the same optimizer we saw in class for the task. The following lines are the main code i use it.

logits = LeNet(x)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=one_hot_y)
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate = rate)
training_operation = optimizer.minimize(loss_operation)

I choose a standard batch size of 128 and a epochs of 30, 50 and 50. I left the epics number of 50.
The learning rate of 0.001 was useful to me. I prove an standard deviation of sigma = 0.1 and sigma = 0.01. I stayed with sigma = 0.1 for the speed.

The code for training the model is located in the cells nine to twelve of the ipython notebook.

4. First i try to use only the same LeNet structure with the modification of increasing the input channels to three. But the performance was less than 0.91. I realize that i will need to change or increase the number of layers. I didn’t want to change the number of layers because i consider the LeNet to be able of the classification task so i increase the number of inputs and outputs in the layer but the validation accuracy didn’t improve very well.

Then I change the color space of the image to YUV in accordance with paper and i normalize the Y channel but the result weren’t satisfactory.

Finally i decided to change to Gray scale considering that the color information were unnecessary because a signal in green or red is the same signal. I improve the contrast of the image due to the poor quality of the set and finally i did the normalization of the image information. This allows me to fix the layers and the inputs of the LeNet.

I choose the LeNet structure because i realize due to the number of channels in the image it would be a good idea to increase the number of layers. But i wanted to pre-process the image information before modify the number of layers.

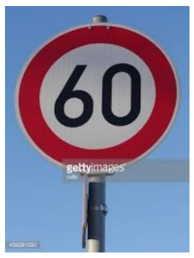
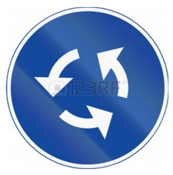
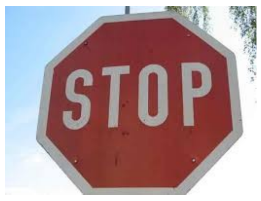
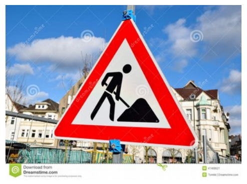
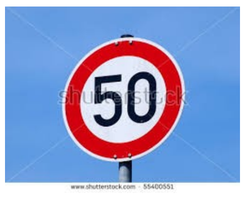
The code for calculating the accuracy of the model is located in the cell number twelve of the Ipython notebook.

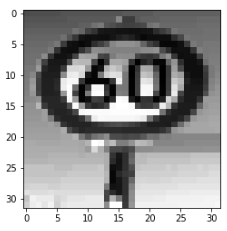
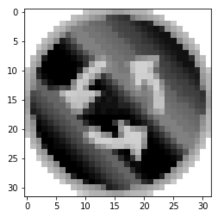
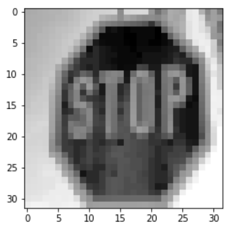
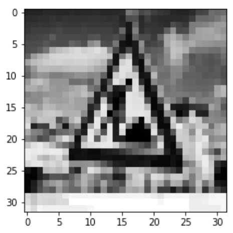
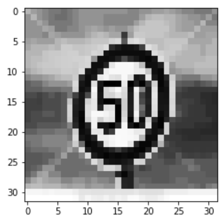
My final model results were:

- training set accuracy of 0.953
- validation set accuracy of 0.934

Test a Model on New Images

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

All the images have a water frame and can be considered a noise insert to the image. I consider changing the images but i believe it was a good idea to try in this way and see what happens.

2. Here are the results of the prediction:

Image	**Prediction**
3	9
40	40
14	13
25	30
2	2

The model was able to correctly guess 1 of the 5 traffic signs, which gives an accuracy of 20%. Sometimes the model was able to correctly guess 2 of 5 signals, which gives an accuracy of 40%.

This is a poor accuracy in contrast with the 0.93 of the validation accuracy. I adjudge this to the water frame.

3. For almost all the images the certainty was absolute. Only ones and zeros for the vector of softmax probabilities. But for the image one the mayor probability was in id 9 and then the probability was in id 35. ID 9: 0.65806568. ID 35: 0.34193435.