Traffic Sign Recognition

Build a Traffic Sign Recognition Project

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

All Code is in file Traffic_Sign_Classifier.ipynb

Data Set Summary & Exploration

1. Provide a basic summary of the data set and identify where in your code the summary was done. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

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

I used the pandas 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.

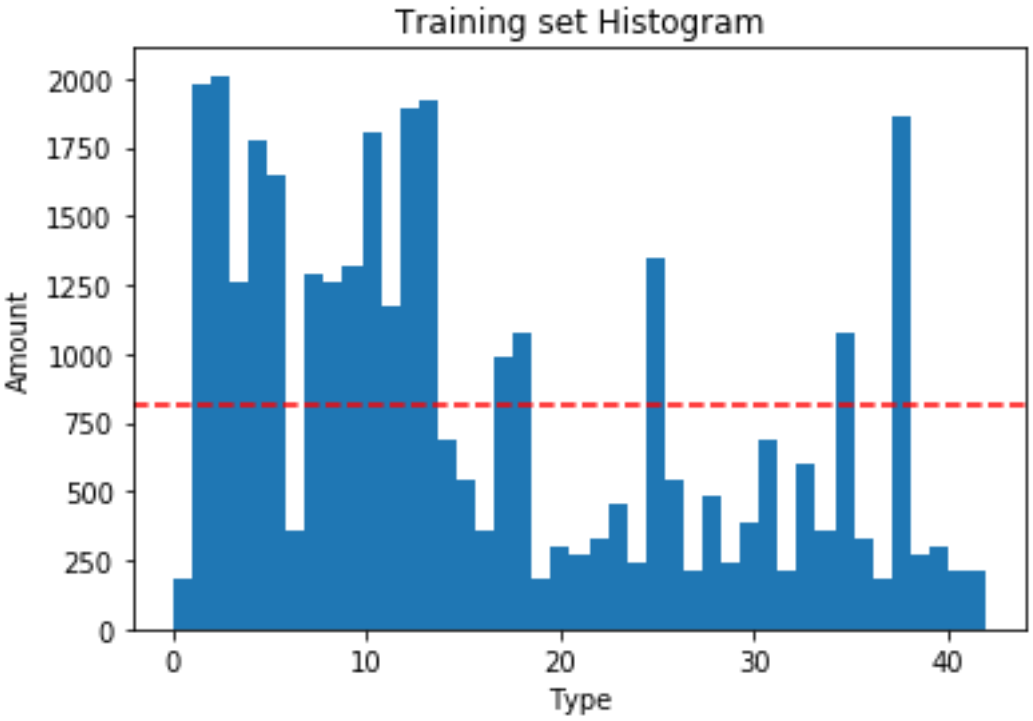
The shape of a traffic sign image is (32, 32, 3).

The number of unique classes/labels in the data set is 43.

2. Include an exploratory visualization of the dataset and identify where the code is in your code file.

The code for this step is contained in the 4th code cell of the IPython notebook.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data...



Design and Test a Model Architecture

1. Describe how, and identify where in your code, you preprocessed the image data. What tecniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc.

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

I decided to pre-process the data by only normalizing it. To make the image data consistent with weights and to get data in the form of zero mean and equal variance.

Also, C(NN)s learn by continually adding gradient error vectors (multiplied by a learning rate) computed from backpropagation to various weight matrices throughout the network as training examples are passed through.

The thing to notice here is the "multiplied by a learning rate".

If we didn't scale our input training vectors, the ranges of our distributions of feature values would likely be different for each feature, and thus the learning rate would cause corrections in each dimension that would differ (proportionally speaking) from one another. We might be over compensating a correction in one weight dimension while undercompensating in another.

This is non-ideal as we might find ourselves in a oscillating (unable to center onto a better maxima in cost (weights) space) state or in a slow moving (traveling too slowly to get to a better maxima) state.

It is of course possible to have a per-weight learning rate, but it's yet more hyperparameters to introduce into an already complicated network that we'd also have to optimize to find. Generally learning rates are scalars.

Thus, we normalize images before using them as input into NN (or any gradient based) algorithm.

2. Describe how, and identify where in your code, you set up training, validation and testing data. How much data was in each set? Explain what techniques were used to split the data into these sets. (OPTIONAL: As described in the "Stand Out Suggestions" part of the rubric, if you generated additional data for training, describe why you decided to generate additional data, how you generated the data, identify where in your code, and provide example images of the additional data)

The code is already splitted in training, validation and testing set.

My final training set had 34799 number of images. My validation set and test set had 4410 and 12630 number of images.

I didn't augment the data. I tweaked the hyperparameters to increase the accuracy.

3. Describe, and identify where in your code, what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.

The code for my final model is located in the 8th code cell of the ipython notebook.

My final model consisted of the following layers:

Layer	Description
Input	32X32X3 Normalized image
Convolution 5X5	1X1 stride, VALID padding, 28X28X64 output
RELU	
Max Pooling	2X2 stride, outputs 14X14X64
Convolution 5X5	1X1 stride, VALID padding, 10X10X32 output
RELU	
Max pooling	2X2 stride, outputs 5X5X32
Fully Connected	800 Input, 512 output
RELU	
Fully connected	512 input, 256 output
RELU	
Fully connected	256 inputs, 128 outputs
RELU	
Fully connected	128 inputs, 43 outputs
Softmax	

4. Describe how, and identify where in your code, you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

The code for training the model is located in the 10th code cell of the ipython notebook.

To train the model, I used an AdamOptimizer so that I need to only tune the learning rate. I tuned my learning rate to 0.001. I tuned batch size to 128 image/epoch. I will run the network for 30 epoches.

5. Describe the approach taken for finding a solution. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think the architecture is suitable for the current problem.

The code for calculating the accuracy of the model is located in the 11th cell of the IPython notebook.

My final model results were:

Validation set accuracy of: - 95.3 %

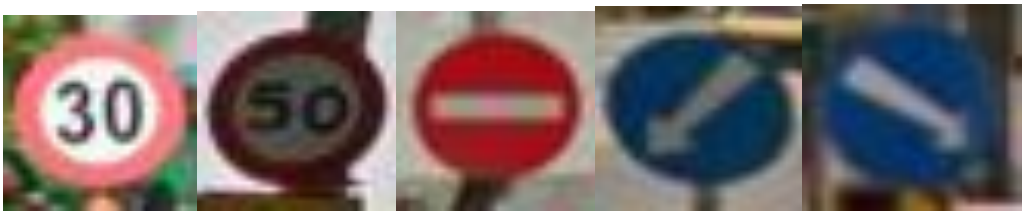
Test set accuracy of: - 93.3%

I started with LeNet, the validation accuracy was not above 90% even after running and tweaking the model architecture many times. After many trial & error and tweaking hyper parameters, I got the result. The goal was keep model small and iterate fast i.e. reduce model complexity. I followed the intuition lesson. Like the first convolution layer starts with detecting objects like lines etc. Then, as the data is passed ahead, more complex objects are formed. That's what I followed. I'm very happy with the result I got.

Test a 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 think except first, remaining all images can be difficult to predict as they have low brightness. I think second image will be difficult to predict as the brightness is low and also the no. 5 in image seems similar to 8.

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. Identify where in your code predictions were made. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).

The code for making predictions on my final model is located in the 15th code cell of the IPython notebook.

I think it will be difficult to recognize traffic sign in second image, as the 50 in image is very much similar to 80.

Here are the results of the prediction:

Image	Prediction
Keep left	Keep left
Speed limit (50km/h)	Speed limit (80km/h)
No entry	No entry
Speed limit (30km/h)	Speed limit (30km/h)
Keep right	Keep right

The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%.

The accuracy on captured images is only 80% and on test data is 93%. But, the images here are only 5. I would be too early to conclude that the model is overfitting or underfitting.

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 and identify where in your code softmax probabilities were outputted. 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 17th cell of the Ipython notebook

For the first image, the model is very sure that this is a Keep left sign (probability of 1.0), and the image does contain a keep left sign. The top five soft max probabilities were

Probability	Prediction
100 %	Keep left
0%	Turn right ahead
0%	Speed limit (70km/h)
0%	Speed limit (30km/h)
0%	Roundabout mandatory

For the second image, the model is very sure that this is a Speed limit (80km/h) sign (probability of 1.0), and the image contains a Speed limit (50km/h) sign. The top five soft max probabilities were

Probability	Prediction
100 %	Speed limit (80km/h)
0%	Speed limit (50km/h)
0%	Speed limit (70km/h)
0%	Keep left
0%	Speed limit (100km/h)

For the third image, the model is very sure that this is a No entry sign (probability of 1.0), and the image does contains a no entry sign. The top five soft max probabilities were

Probability	Prediction
100 %	No entry
0%	Speed limit (20km/h)
0%	Speed limit (30km/h)
0%	Speed limit (50km/h)
0%	Speed limit (60km/h)

For the fourth image, the model is very sure that this is a Speed limit (30km/h) sign (probability of 1.0), and the image does contains a Speed limit (30km/h) sign. The top five soft max probabilities were

Probability	Prediction
100 %	Speed limit (30km/h)
0%	Speed limit (70km/h)
0%	Speed limit (50km/h)
0%	Road narrows on the right
0%	End of speed limit (80km/h)

For the fifth image, the model is very sure that this is a keep right sign (probability of 1.0), and the image does contains a keep right sign. The top five soft max probabilities were

Probability	Prediction
100 %	Keep right
0%	Turn left ahead
0%	Slippery road
0%	Speed limit (20km/h)
0%	Speed limit (30km/h)