Traffic Sign Recognition

Build a Traffic Sign Recognition Project

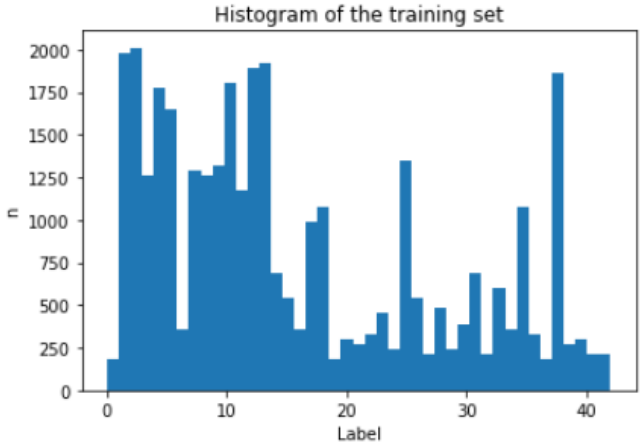
The goals / steps of this project are the following:

- Load the data set (see below for links to the project 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

1. Data Set Summary & Exploration

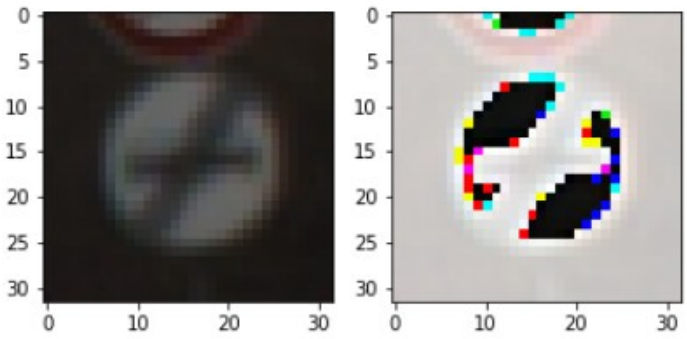
I used the pandas library to calculate summary statistics of the traffic signs data set:

- The size of training set is **34799**
- The size of the validation set is **4410**
- The size of test set is **12630**
- The shape of a traffic sign image is **32x32x3**
- The number of unique classes/labels in the data set is **43**



2. Design and Test a Model Architecture

(1). The raw image is a 32x32x3 image, since the image color contains information about the traffic sign, I did not convert the image to gray scale. For pre-processing, the raw image is normalized, then fed into the CNN. An example of an image before and after normalized image can be found below. Note that we can modify the image to generate additional training data, which will increase the training accuracy, however, I can not import openCV in from AWS instance, so I did not use this approach.



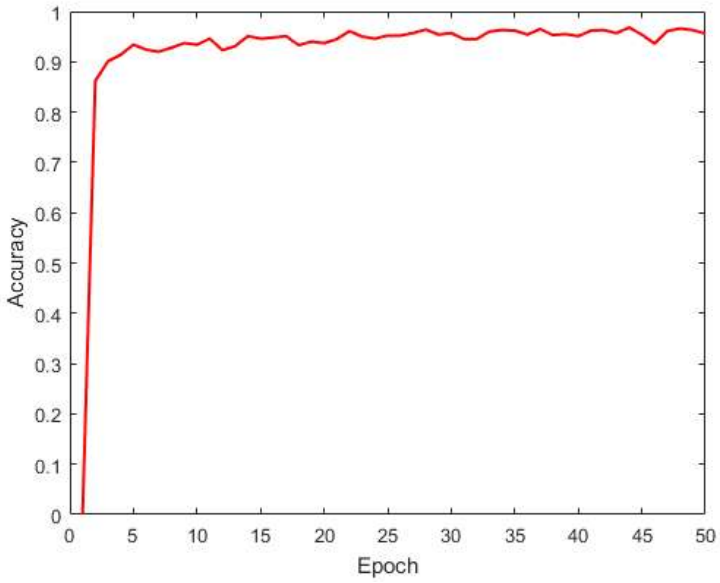
(2). The following table shows the structure of the CNN used in this project

Layers	**Detail**
Input	32x32x3 RGB image
CNN filter: 5x5x3 depth: 40	1x1 stride, valid padding, outputs 28x28x40
RELU	
Max Pooling	2x2 stride, outputs 14x14x40
CNN filter: 5x5x40 depth: 30	1x1 stride, valid padding, outputs 10x10x30
RELU	
Max Pooling	2x2 stride, outputs 5x5x30
Flatten	750
Fully connected	Weights 750x500
RELU & Drop out (0.7)	
Fully connected	Weights 500x300
RELU	
Fully connected	Weights 300x200
RELU	
Fully connected	Weights 200x100

RELU	
Fully connected	Weights 100x80
RELU	
Output	Weights 80x43

(3). The batch size used for training the CNN is 150 images per-batch, and 50 epochs were completed to train the model. The learning rate of the CNN is 0.0009 and the optimizer employed is Adam.

(4). The training set accuracy is around 97% and the validation accuracy is around 96%. I firstly used the LeNet structure but the accuracy is very low. I increased the CNN layer filter depth and used more fully connected layers, and noticed an improvement on the validation accuracy. I also tried to use drop out to avoid model over-fitting, but noticed that the accuracy decreases with drop-out. Thus, I decided to not use the drop-out but terminate the training earlier. The plot below shows the validation accuracy during training.



3. Test a Model on New Images

(1). The following 5 images are the new testing images that are not included in the original data set and were obtained online.

(2). The following table shows the prediction from the trained CNN, the overall accuracy is 80%. However, only 5 images were tested, with more testing images, I believe the accuracy should approach to a value that is close to the validation accuracy .

Image	Prediction
Go straight or left	Go straight or left
Dangerous curve to the left	Ahead only
No passing	No passing
General caution	General caution
Bumpy road	Bumpy road

(3). The following 5 tables shows the top 5 probability of each testing image. The model is very certain to predict the tested image.

Image 1	Probability
37	1
0	0
1	0
2	0
3	0

Image 2	Probability
35	1
0	0
1	0
2	0
3	0

Image 3	Probability
11	1
30	3.8e-18
26	1.7e-22
27	9.7e-22

21	7e-24

Image 4	Probability
18	1
39	1.3e-23
37	1.5e-25
4	3.6e-30
35	1.5e-31

Image 5	Probability
22	0.99
29	3.4e-5
39	2.1e-6
28	1.0e-7
13	9.9e-8