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

Data Set Summary & Exploration

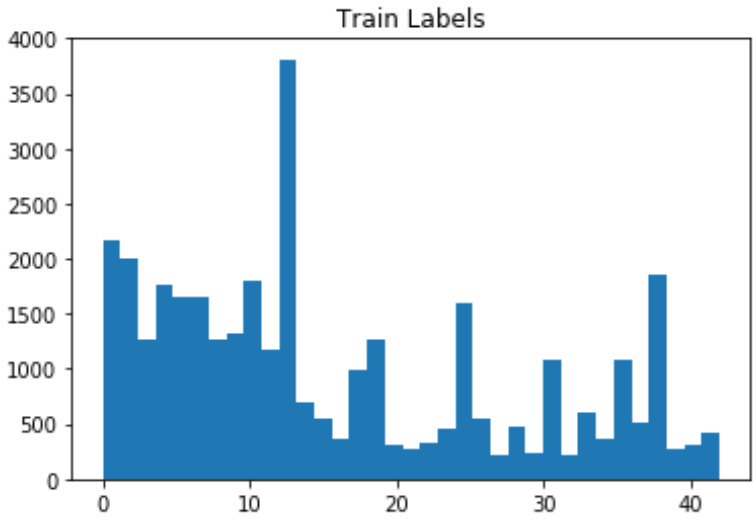
1. Basic summary of the data set

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

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

2. Exploratory visualization of the dataset.

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



Design and Test a Model Architecture

1. Image Preprocessing

As a first step, I decided to convert the images to grayscale because usually color information is not needed.

Next, I normalized the image data to have zero mean and equal variance. For image data, `(pixel - 128)/ 128` is a quick way to approximately normalize and has been used.

I tried subtracting mean and dividing by standard deviation and it did not work well.

2. Model description

The model is very similar to LeNet. I have added an additional conv layer. I have also added drop out after the first two FC layers. Increased the no of filters in each conv layer so that network can learn more patterns/features.

3. Model Architecture

Layer	Description
Input	32x32x1 Gray scale Image
Convolution	5x5 filter with 1x1 stride, valid padding, outputs 28x28x6
RELU	
Max Pool	2x2 Filter with 2x2 stride, outputs 14x14x6
Convolution	5x5 filter with 1x1 stride, valid padding, outputs 10x10x32
RELU	
Max Pool	2x2 Filter with 2x2 stride, outputs 5x5x32
Convolution	3x3 filter with 1x1 stride, valid padding, outputs 5x5x64

RELU	
Flatten	outputs 1600
Fully Connected	Input 1600 and outputs 400
RELU	
Dropout	keep_prob=0.85
Fully Connected	Inputs 400 and outputs 120
RELU	
Dropout	keep_prob=0.85
Fully Connected	Inputs 120 and outputs 43

4. Model training

To train the model, I used the Adam Optimizer and cross entropy. The model ran 5 epochs. I initially chose a batch size of 128 and later reduced it to 64 as I saw better training with 64.

5. Model selection

Tried different architectures. Initially I tried LeNet and found that it was not giving good accuracy. Then I added a conv layer and saw little improvement in accuracy. Then I stated to experiment with different no of filters in conv layers. I also added drop out after every FC layer so that network does not overfit and generalize well as usually there are more no of parameters in FC layer.

6. Discussion

My final model results were:

- training set accuracy of 99.5%
- validation set accuracy of 95.8%
- test set accuracy of 94.6%

The initial architecture tried was classic LeNet. Since LeNet architecture is small and it also had only 10 classes to classify among from, the chosen architecture did quite well for the task it was designed for. But here since we had way a greater number of classes (43), it did not perform well. So, I added one more conv layer in expectation that the network will have the capability to learn more complex features. It seemed to improve the network accuracy a little bit. Then I increased the no of filters in 2nd and 3rd conv layer's once again with the expectation that network will learn more complex features and it did. Then I added drop out layer after first and second fully connected layers. As most of the network parameters are in the FC layers, the network might start overfitting and hence drop out layer was added to avoid that. I experimented with different keep_prob values. Initially I tried a value of 0.85 and it did not give me good results, the training accuracy reached 99% much faster. So, I decreased it to 0.5 and it seemed to work well. I also tried experimenting with the learning rate. I increased the learning rate to 0.1 in the assumption that network might learn faster but that did not seem to be the case. So, I went back to the classic value of 0.001. Also experimented with the batch_size. Initially I used a value of 64 and then decided to go with 128 as that would give a better approximate of dataset.

Test a Model on New Images

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

Model predictions are:

Turn right ahead
Turn left ahead
Go straight or right
End of no passing by vehicles over 3.5 metric tons
Stop

True Predictions are:

Turn right ahead
Turn left ahead
Go straight or right
Stop
Stop

The model was able to classify only four images correctly, which gives an accuracy of 80%.

The top 5 probabilities for each image are as follows:

(1.0, 'Turn right ahead')
(0.0, 'Ahead only')
(0.0, 'Stop')
(0.0, 'Yield')
(0.0, 'Keep left')

(0.50999999, 'Turn left ahead')
(0.37, 'Go straight or right')
(0.029999999, 'Speed limit (120km/h)')
(0.029999999, 'Stop')
(0.02, 'Speed limit (20km/h)')

(0.99000001, 'Go straight or right')
(0.0099999998, 'Road work')
(0.0, 'Traffic signals')
(0.0, 'Priority road')
(0.0, 'Roundabout mandatory')

(0.34999999, 'End of no passing by vehicles over 3.5 metric tons')
(0.22, 'No passing for vehicles over 3.5 metric tons')
(0.2, 'No passing')
(0.090000004, 'Roundabout mandatory')
(0.050000001, 'Speed limit (50km/h)')

(0.99000001, 'Stop')
(0.0, 'Go straight or right')

(0.0, 'Yield')
(0.0, 'Turn right ahead')
(0.0, 'Road work')