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

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

- The size of training set is 39209
- 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. 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 third code cell of the IPython notebook. Here is an exploratory visualization of the data set. It is a bar chart showing (can be seen in IPython notebook how the data is distributed in different classes, Also I randomly choose multiple images from training set and displayed it using pyplot.

Design and Test a Model Architecture

1. Describe how, and identify where in your code, you preprocessed the image data. What techniques 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 gray scale, normalization, etc.

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

As a first step, I decided to convert the images to gray scale because it is easier to find edges in gray-scaled image. Here is an example of a traffic sign image before and after gray scaling. Also I used normalization on gray-scaled image to reduce the effect of illumination differences. Code in fifth code is to display the preprocessed images after converting to gray scale and normalization respectively

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 for augmenting the data is contained in 6th and 7th code cell. I generated new images by applying rotation, translation, sheering on each images. For rotation I applied a uniform range or 30 degrees and translation and sheer range of 5 each. Below 7th code cell is the output of augmented images shown in grid. There are some classes with very few number of training samples in it, this may lead the model to bias towards the classes with high number of images. I augmented data such the every class will have same number of training examples. The class with highest number of training samples has 2250 images in it. I augmented images such that every class has 2250 number of images after applying augmentation.

In the 8th code cell, I'm appending this augmented data to existing training examples. In the 9th code cell, I'm shuffling the training samples and labels.

The code for splitting the data into training and validation sets is contained in the 10th code cell of the IPython notebook. To cross validate my model, I randomly split the training data into a training set and validation set. I did this by using *train_test_split()* function from *sklearn* library. My final training set had 87075 images. My validation set and test set had 9675 and 12630 number of images.

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 13th cell of the ipython notebook. My final model consisted of the following layers:

Layer	Description
Input	32x32x1 Gray Scale, Normalized
Convolution 5x5	1x1 stride, valid padding, outputs 28X28x6
RELU	
Convolution 5x5	1x1 stride, same padding, outputs 28X28x6
RELU	
Max pooling	2x2 stride, valid padding, outputs 14x14x6
Dropout	Keep_prob = 0.5(for training) 1(for testing)
Convolution 5x5	1x1 stride, valid padding, outputs 10X10x16
RELU	
Convolution 5x5	1x1 stride, same padding, outputs 10X10x16
RELU	
Dropout	Keep_prob = 0.5(for training) 1(for testing)
Flatten	Outputs 400
Fully connected	Outputs 120
RELU	
Dropout	Keep_prob = 0.5(for training) 1(for testing)
Fully connected	Outputs 84
logits	Outputs 43

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 17th cell of the ipython notebook. To train the model, I used

- Batch size of 256
- Learning rate of 0.001
- AdamOptimizer
- Keep_prob: 0.5
- Ran optimization for 45 epochs

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 16th cell of the Ipython notebook.

My final model results were:

- training set accuracy of 94.1%
- validation set accuracy of 93.6%
- test set accuracy of 93.8%

If an iterative approach was chosen:

- What was the first architecture that was tried and why was it chosen?
    - At first I have used LeNet architecture that is exercised in the class. Later I have tried adding different activation mechanisms (elus, relus, dropouts), different learning rates and also experimented with different optimizers.
- What were some problems with the initial architecture?
    - In most of the initial architectures the validation accuracy is ~80-85%. But when I apply the model on images that I downloaded from web, model is able to output accuracy of 20-40%.
- How was the architecture adjusted and why was it adjusted? Typical adjustments could include choosing a different model architecture, adding or taking away layers (pooling, dropout, convolution, etc), using an activation function or changing the activation function. One common justification for adjusting an architecture would be due to over fitting or under fitting. A high accuracy on the training set but low accuracy on the validation set indicates over fitting; a low accuracy on both sets indicates under fitting.
    - To overcome the problem of network not performing well on unseen data that is downloaded from web, I added more training data by augmenting the dataset (as explained in earlier section). Now I have ~100K training examples. I feed the data to network. Now the network seems perform worse because the accuracy of validation data is reducing. In fact, the data is under fitting, I added more convolution layers to compensate this.
- Which parameters were tuned? How were they adjusted and why?
    - I tried different combinations of learning rates, batch sizes, epochs. I tried with learning rates = 0.1, 0.01, 0.03, 0.001, batch size = 128, 256, 512, and epoch 20, 45, 50 for this finalized model. But I selected the parameters that gave a high validation accuracy on the data.
    - Sometimes the learning rate is too high that my loss keeps on increasing and sometimes it is too low that loss is reducing very slowly. This is lot of guess and try work, I ran the network every time I chose a different parameter to see how the loos plot looks like and make any changes needed bases on loss curves.

- What are some of the important design choices and why were they chosen? For example, why might a convolution layer work well with this problem? How might a dropout layer help with creating a successful model?
    - These are the layers that I used
        - Convolution: Are good to get the different features of the image, like edges. So convolutions are used in the network to convert the input images into different feature maps
        - RELU: This is added to introduce non-linearity's into the network, this will replace all negative pixels with zero.
        - Pooling: I used max pool to reduce the dimensionality of the feature map, this also will retain the most import information
        - Dropout: Drop out activations are added to make sure that network doesn't over fit with the training data.
        - Fully Connected Layer: These layers are added at the end of the network to feed the features from previous layer for classifying in to classes.

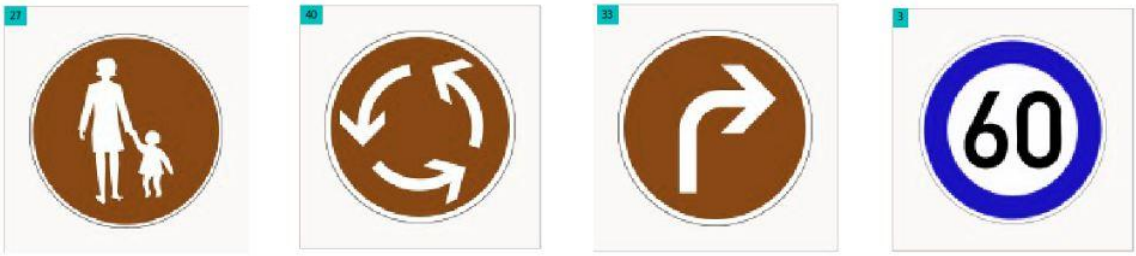
If a well-known architecture was chosen:

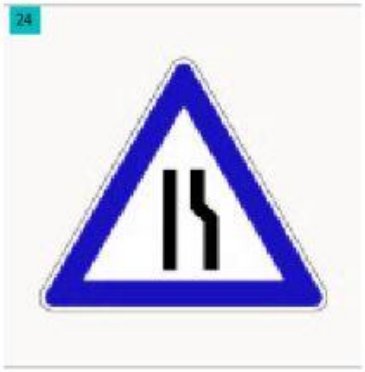
- What architecture was chosen?
    - I used LeNet architecture, but slightly modified to add more layers to it (convolutions, dropouts etc.)
- Why did you believe it would be relevant to the traffic sign application?
    - This architecture is for finding the handwritten digits from the images, in both the cases the network is working on same type of inputs and trying to classify them into multiple classes
- How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?
    - These 3 accuracies are about in the same range suggesting that model is pretty much generalized to classify unseen test data as it is classifying seen train data.

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:





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 23rd cell of the Ipython notebook.

Here are the results of the prediction:

Image	Prediction
Pedestrians	Turn left ahead
Roundabout mandatory	Roundabout mandatory
Turn right ahead	Turn right ahead
Speed limit (60km/h)	Speed limit (60km/h)
Road narrows on the right	Road narrows on the right

The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%. This compares favorably to the accuracy on the test set of 93.8%

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 25th cell of the Ipython notebook.

For the first image, the model is sure that this is a Turn left ahead sign (probability of 0.9), and the actual image does contain a pedestrian sign.

The top five soft max probabilities were

Probability	Prediction
.99	Turn left ahead
0.00018	Ahead only
0.00010	Go straight or right
0.000003	Keep right
0.0	Traffic signals

For the second image, the model is pretty sure that this is a Roundabout mandatory sign (probability of 0.9), and the actual image does contain a Roundabout mandatory sign.

The top five soft max probabilities were

Probability	Prediction
.99	Roundabout mandatory
.00017	Priority road
.00015	Keep right
.000013	Speed limit (100km/h)
.000011	Speed limit (120km/h)

For the Third image, the model is sure that this is a Turn right ahead sign(probability of 1.0), and the actual image does contain a Turn right ahead sign.

The top five soft max probabilities were

Probability	Prediction
1.0	Turn right ahead
.00000002	Ahead only
.0000000008	No passing for vehicles over 3.5 metric tons
.0000000003	Turn left ahead
.00000000002	Stop

For the fourth image, the model is sure that this is Speed limit (60km/h) sign (probability of 0.9), and the actual image does contain a Speed limit (60km/h) sign.
The top five soft max probabilities were

Probability	Prediction
.99	Speed limit (60km/h)
.001	Speed limit (80km/h)
.00005	Speed limit (50km/h)
.000000005	Speed limit (30km/h)
.000000002	No passing for vehicles over 3.5 metric tons

For the fifth image, the model is sure that this is a Road narrows on the right sign (probability of 0.9), and the actual image does contain a Road narrows on the right sign.
The top five soft max probabilities were

Probability	Prediction
.99	Road narrows on the right
.0000014	Double curve
.00000069	Pedestrians
.000000023	Children crossing
.0000000077	General caution