Traffic Sign Recognition

## Writeup

Build a Traffic Sign Recognition Project

### Data Set Summary & Exploration

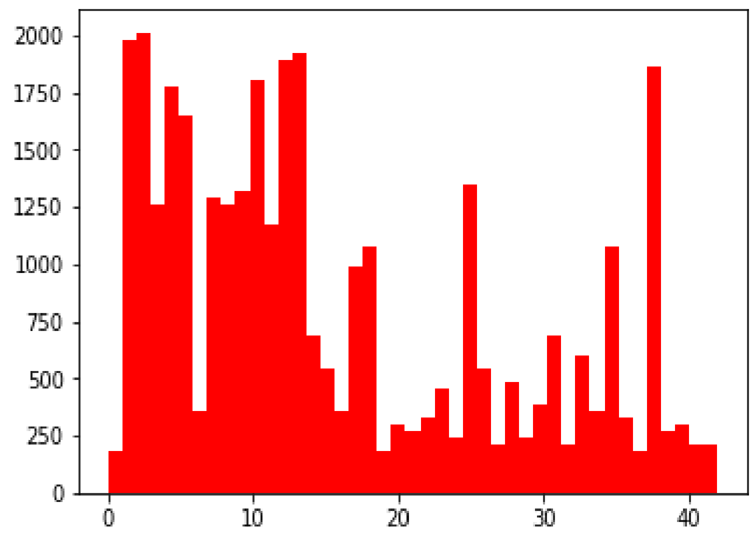
#### 1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

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

Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of unique classes / labels = 43

#### 2. Include an exploratory visualization of the dataset.

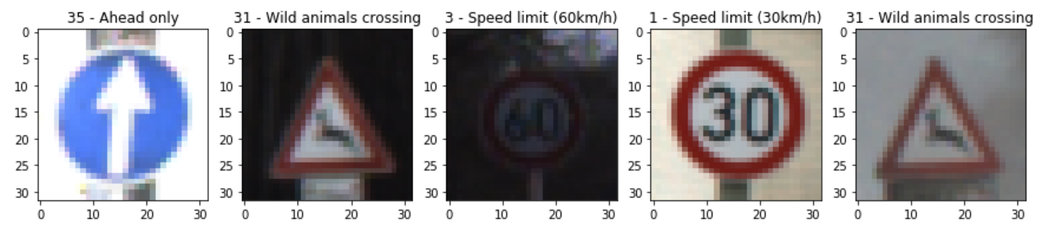
Here is an exploratory visualization of the data set. It is a histogram chart showing how the data is spread across the classes available in the training data set.



I have also displayed 5 random images that are in our training set.

```
####Visualizing 5 random images from the training set####

 5 Random images with their sign names:
```



### Design and Test a Model Architecture

#### 1. Describe how 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 grayscale, normalization, etc. (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, and provide example images of the additional data. Then describe the characteristics of the augmented training set like number of images in the set, number of images for each class, etc.)

As a first step, I decided to convert the images to grayscale because gray scaling makes the neural network faster and it focus more on what is important. There are multiple options to convert the image to grayscale like opencv function, tensor flow function and also to divide the rgb values by 3 to arrive at the image. But I chose the below method as it captures the gray shade more precisely.

```
0.2989 *img[:,:,:,0] +  0.5870*img[:,:,:,1] + 0.1140*img[:,:,:,2]
```

I normalized the image data so that the data has zero mean and equal variance. I have used the following method for normalization.

`(pixel - 128)/ 128`

I have used the np.resize method to maintain the lost dimension during the above preprocessing steps.

```
Shape of Training data before Preprocessing and : (34799, 32, 32, 3)
Shape of Testing data before Preprocessing and : (12630, 32, 32, 3)
Shape of Validation data before Preprocessing and : (4410, 32, 32, 3)
Shape of Training data after Preprocessing and : (34799, 32, 32, 1)
Shape of Testing data after Preprocessing and : (12630, 32, 32, 1)
Shape of Validation data after Preprocessing and : (4410, 32, 32, 1)
```

I decided to generate additional data because the data available to train the neural network for certain classes was very minimal. It was clearly shown in the histogram earlier.

As a result of the above scenario, the neural network can be able to identify only the classes that have more data and won't be able to identify other classes clearly.

So, one way to rectify the problem is to augment certain data sets to the original training set that was given to me.

To add more data to the data set, I used the data available in the data set given for each class and rotated them to arrive at the new image sets.

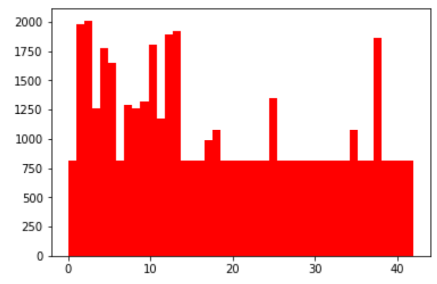
My motive is to evenly spread the data. So I decided to increase the no of images in the classes that was lacking them. I chose to increase all the classes that was lacking to the mean of images of all the classes before augmentation.

We have augmented the data by rotating the available data to make each class to have at least the mean 809. I have used scipy's ndimage.rotate method. I have chosen the rotation degree as below.

```
# Angles to be used to rotate images in additional data made
angles = [-5, 5, -15, 15, -25, 25]
```

Find below the augmented classes.

```
         Displaying the graph which plots the distribution of the 43 classess in the training data sets post augmentation
Out[37]: (array([  809.,  1980.,  2010.,  1260.,  1770.,  1650.,   809.,  1290.,
                 1260.,  1320.,  1800.,  1170.,  1890.,  1920.,   809.,   809.,
                  809.,   990.,  1080.,   809.,   809.,   809.,   809.,   809.,
                  809.,  1350.,   809.,   809.,   809.,   809.,   809.,   809.,
                  809.,   809.,   809.,  1080.,   809.,   809.,  1860.,   809.,
                  809.,   809.,   809.]),
          array([  0.        ,   0.97674419,   1.95348837,   2.93023256,
                  3.90697674,   4.88372093,   5.86046512,   6.8372093 ,
                  7.81395349,   8.79069767,   9.76744186,  10.74418605,
                 11.72093023,  12.69767442,  13.6744186 ,  14.65116279,
                 15.62790698,  16.60465116,  17.58139535,  18.55813953,
                 19.53488372,  20.51162791,  21.48837209,  22.46511628,
                 23.44186047,  24.41860465,  25.39534884,  26.37209302,
                 27.34883721,  28.3255814 ,  29.30232558,  30.27906977,
                 31.25581395,  32.23255814,  33.20930233,  34.18604651,
                 35.1627907 ,  36.13953488,  37.11627907,  38.09302326,
                 39.06976744,  40.04651163,  41.02325581,  42.        ]),
          <a list of 43 Patch objects>)
```



#### 2. Describe 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.

My final model consisted of the following layers:

2 Convolutional layers and followed by 3 fully connected layers.
I have used the LeNet architecture. I have chosen the mean to be 0 and standard deviation to be 0.1.

I have chosen Relu as my activation function and done max pooling as well in each of the convolutional layer.

I have used drop out functions to eliminate over fitting. I have used 25% of drop out and 30 % of drop out in two fully connected layers.

I have used one hot encoding to labels.

I have used the tensor flow functions wherever possible. I used the softmax function to calculate the cross entropy and find the cost of loss and to reduce it ultimately as we train the neural network.

I have attached below the details of the architecture.

Layers	Functions Applied	I/P Dimensions	O/P Dimensions
Convolutional Layer 1	convolution	32x32x1	28x28x6
	relu		
	max pooling	28x28x6	14x14x6
Convolutional Layer 1	convolution	14x14x6	10x10x6
	relu		
	max pooling	10x10x6	5x5x16
Convert to 1D	flatten	5x5x16	1x400
Fully Connected Layer 1	matmul	1x400	1x120
	relu		
	drop out		
Fully Connected Layer 2	matmul	1x120	1x84
	relu		
	drop out		
Fully Connected Layer 3	matmul	1x84	1x43

#### 3. Describe how 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.

I used the AdamOptimizer from within TensorFLow to optimize, which seemed to do better than a Gradient Descent Optimizer.

As I discussed earlier, I have chosen a 0 mean and 0.1 standard deviation and I stuck to the batch size of 128 as it worked well for me from the beginning.

Initially I have chosen 20 epochs with a high learning rate of 0.005 and drop out probabilities of 0.5 for 2 fully connected layers. Which didn't work well as expected.

Decreasing the learning rate further and keeping the epochs same seems to make the efficiency drop a bit.

So finally I stuck on to 50 epochs and a learning rate of 0.0009 and drop out probabilities of 0.75 and 0.8.

#### 4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. 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.

I have chosen an iterative approach as I am not sure of what would be the best values for hyper parameters straight away.

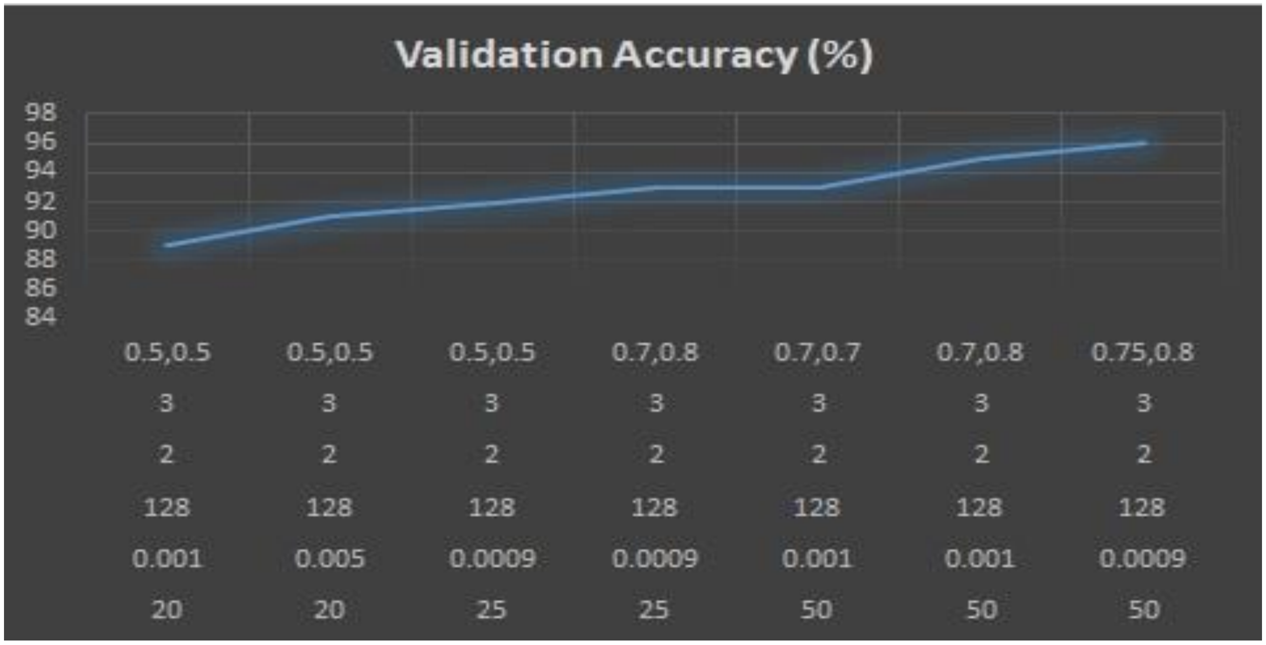
I believe there is no one particular value that will fit for a hyper parameter in all the cases.

We need to make sure that in the course of training, I should choose the hyper parameters in a way that avoids both over fitting and under fitting.

To elaborate further, I have made the below table and the following graph to illustrate how I have chosen my model iteratively.

EPOCHS	Learning Rate	Batch Size	No of Convolution Layers	No of Fully Connected Layers	Dropout Probabilities	Validation Accuracy (%)

20	0.001	128	2	3	0.5,0.5	89
20	0.005	128	2	3	0.5,0.5	91
25	0.0009	128	2	3	0.5,0.5	92
25	0.0009	128	2	3	0.7,0.8	93
50	0.001	128	2	3	0.7,0.7	93
50	0.001	128	2	3	0.7,0.8	95
50	0.0009	128	2	3	0.75,0.8	96



In the end, the model gave the validation accuracy around 96% and test accuracy of 93.90% as shown below.

```
EPOCH 49 ...
Validation Accuracy = 0.960

EPOCH 50 ...
Validation Accuracy = 0.958

Model saved
```

```
In [45]: with tf.Session() as sess:
             saver.restore(sess, tf.train.latest_checkpoint('.'))

             test_accuracy = evaluate(X_test, y_test)
             print("Test Accuracy = {:.3f}".format(test_accuracy))
```

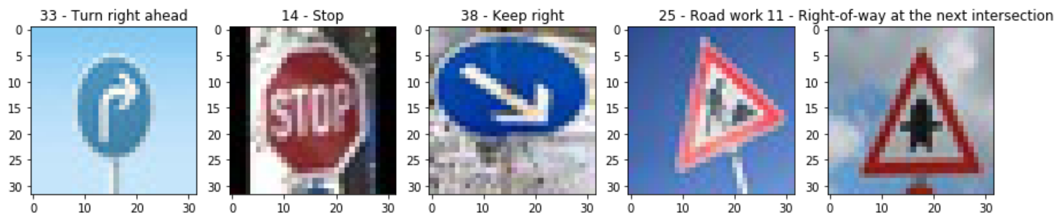
```
Test Accuracy = 0.939
```

### 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:

```
The following are the images that I am going to process with the trained model
```



I think, it's very difficult find road work as many signs use the similar outer structure. And I am pretty sure that stop sign will be found easily due to the shape of it.

And also, Right of way at the next intersection sign should do well as it is a unique sign.

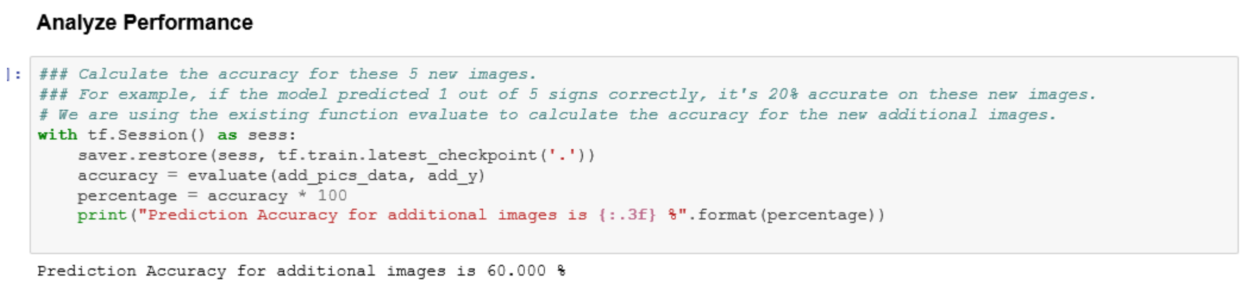
And turn right ahead should also be easy as it is clearly visible.

One potential difficulty would be the low resolution of the image data.

And other would be the similarities with the other traffic signs.

#### 2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. 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 model was able to correctly guess 3 of the 5 traffic signs, which gives an accuracy of 60%.



Analyze Performance

```
]: ### Calculate the accuracy for these 5 new images.
   ### For example, if the model predicted 1 out of 5 signs correctly, it's 20% accurate on these new images.
   # We are using the existing function evaluate to calculate the accuracy for the new additional images.
   with tf.Session() as sess:
       saver.restore(sess, tf.train.latest_checkpoint('.'))
       accuracy = evaluate(add_pics_data, add_y)
       percentage = accuracy * 100
       print("Prediction Accuracy for additional images is {:.3f} %".format(percentage))
```

```
Prediction Accuracy for additional images is 60.000 %
```

The predictions of the model is shown below for each of the image with their top 5 probabilities:

Input Image	Predicted Image	Probability
Turn right ahead		
	Turn right ahead	95.51
	Speed limit (100km/h)	4.36
	Beware of ice/snow	0.122
	Speed limit (120km/h)	0.0002
	Ahead only	0.0001
Stop		
	Stop	100
	Turn left ahead	0
	No entry	0
	Turn right	0
	Keep right	0
Keep right		
	Slippery road	99.9878
	Dangerous curve to the left	0.0105
	No passing for vehicles over 3.5 metric tons	0.0012
	Dangerous curve to the right	0.0004
	Turn left ahead	0
Road work		
	Go straight or left	52.81
	Roundabout mandatory	46.53
	Keep left	0.304
	No passing	0.296
	Dangerous curve to the right	0.038
Right-of-way at the next intersection		
	Right-of-way at the next intersection	100
	End of speed limit (80km/h)	0
	Beware of ice/snow	0
	Children crossing	0

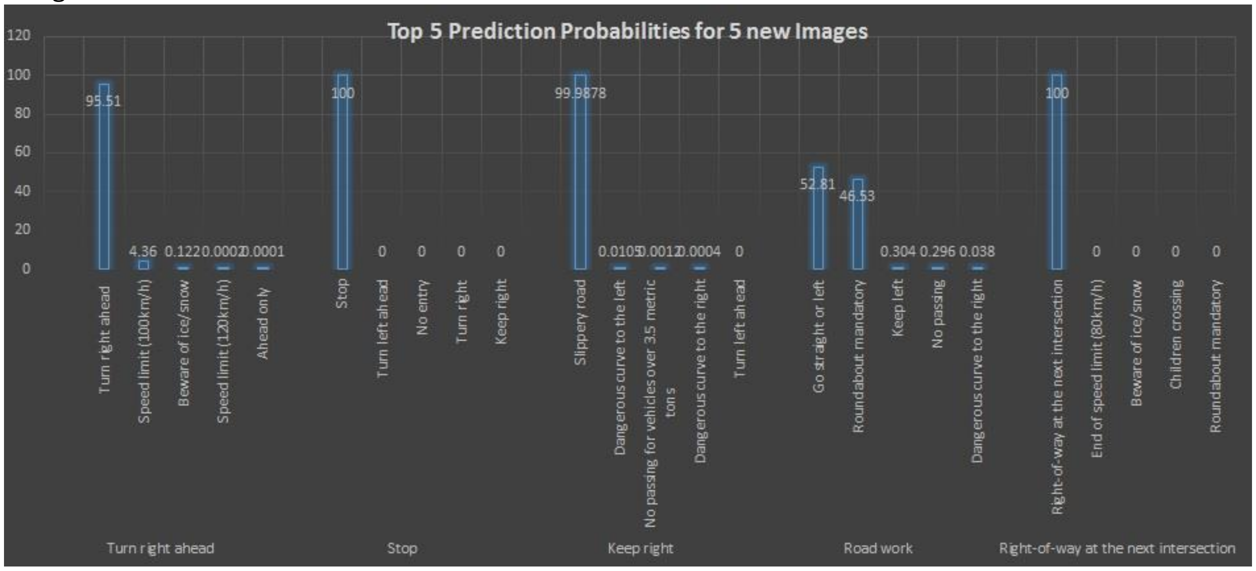
	Roundabout mandatory	0

This is lower than the accuracy given by the test images which is around 93 %. This is due to the similarities of the images with other images and resolution.

#### 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. 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)

As I expected, the model was able to find 3 of the images correctly. Stop, Right-of-way at the next intersection and Turn right ahead.

Find below the graph that shows the top 5 probabilities of the model prediction for each of the 5 images.



As I discussed earlier model wasn't able to find the other two images correctly due to resolution and similarities between the signs.

The other possible reason could be that the train images for these 2 signs are 180 and 240 for keep right and road work respectively. The model might work well if we give more images for these classes as well. And also tweaking some of the parameters that I have given and training the model would also be helpful to achieve the accuracy that I have got for test images.