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

The code for this step is contained in the second and third 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 : 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

  Below is the corresponding output :

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

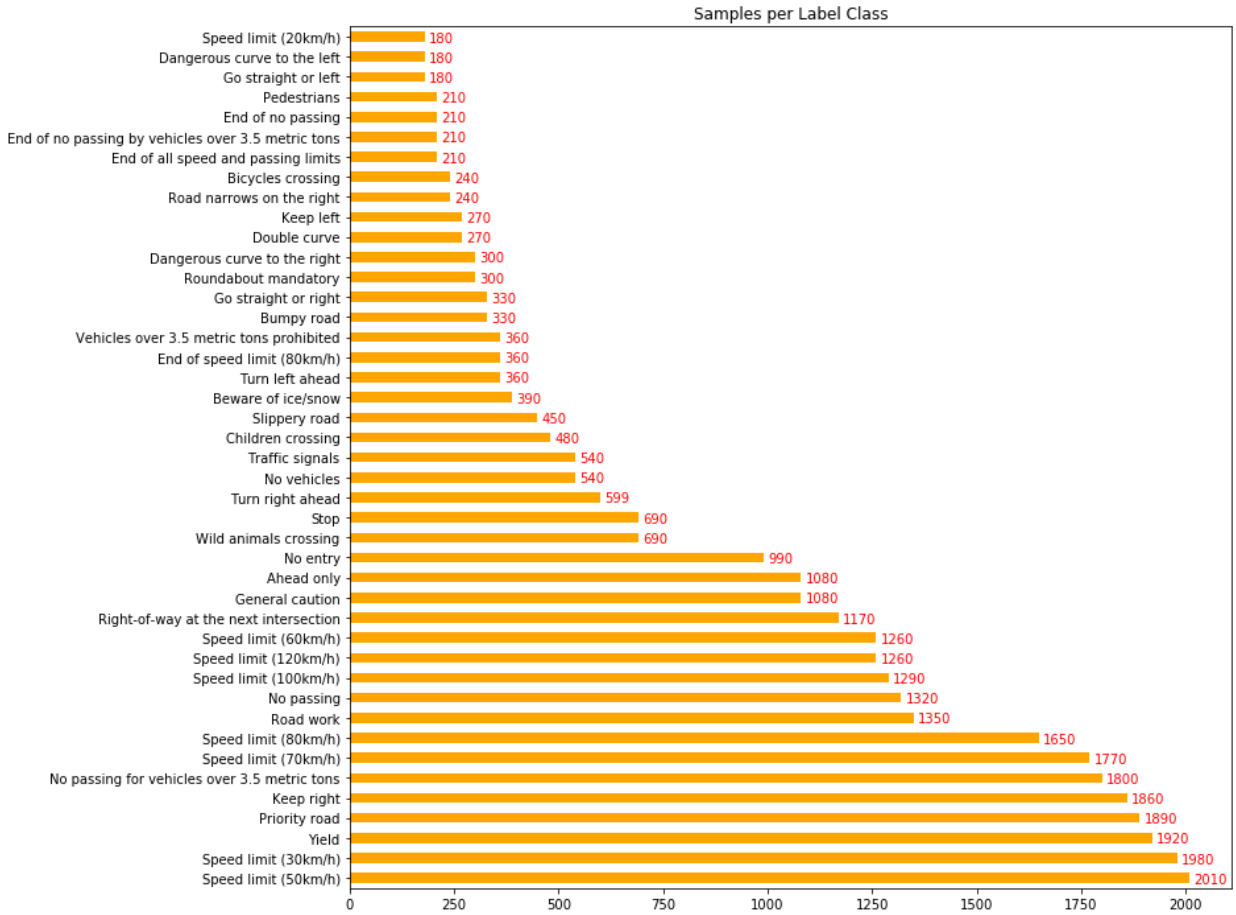
Sample traffic sign display

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

Below are the radom traffic signs printed from training set:

Data Exploration

Here is an exploratory visualization of the data set. It is a bar chart showing how the data distribution with different category of data / sign, the code for this step is contained in the fourth code cell of the IPython notebook. Pandas library used to create label data frame and maplotlib to plot the graph



Design and Test a Model Architecture

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

Image preprocessing done using tensor flow functions, which preprocess the whole batch of images.

1. Brightness : Randomly adjust the image brightness, which improves the brightness of the image, this preprocessing applied to increase the visibility or clarity of the image
2. Contrast : Randomly adjust the image contrast, which improves the contrast of the image, this preprocessing applied to increase the visibility or clarity of the image
3. Gray Scale: As a next step, I decided to convert the images to grayscale because; this is sufficient to find edges of the image and improves the speed of the learning because of less channel to process the data.
4. Image Normalization: As a last step, I normalized the image data using tensorflow API *tf.image.per_image_standardization(image)*, which Linearly scales image to have zero mean and unit norm.

   This operation computes `(x - mean) / adjusted_stddev`, where `mean` is the average of all values in image, and `stddev` is the standard deviation of all values in an image. This helps model is able to learn the real structures instead of dealing with the scale differences

Model Architecture:

Layer	Description
Input	32x32x1 Grayscale image
Convolution 3x3	1x1 stride, valid padding, outputs 28x28x6
RELU	
Local normalization	
Max pooling	2x2 stride, outputs 14x14x6
Convolution 3x3	1x1 stride, valid padding, outputs 10x10x16
RELU	
Local normalization	
Max pooling	2x2 stride, outputs 5x5x16

Flatten	Input = 5x5x16 Output = 400
Fully connected	Input = 400 Output = 120
RELU	
Fully connected	Input = 120 Output = 84
RELU	
Fully connected	Input = 84. Output = 10
Softmax	

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

My final model consisted of the following layers:

Parameters tuned for this model :

Number of Epochs: 30

Learning rate : 0.001

Mean for weight : 0 and standard deviation : 0.1

Batch size : 128

Loss function using L2 Regularization: This is the key method for overfitting problem, I noticed my model is overfitting little and dropout is not effective then I included regularization method and this manage to solve overfitting problem and improve the accuracy on the validation set

Optimization using Adam Optimizer: I selected this optimizer because the method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters.

Model is created incrementally, step taken to finalize the model or solution :

1. Started with using LabNet solution provided and modified according to image size
   Accuracy on validation set achieved 0.87

2. Converted image to grayscale and applied image preprocessing(brightness, contrast) and normalization
   Accuracy on validation set achieved 0.92 and accuracy on training set is 0.99

3. Added dropout with keep_prob:0.5 because difference between accuracy as mentioned in above step, thought overfitting
   However, the result of it found model begin to underfit, validation accuracy is 0.76and training accuracy 0.82
   May be little bit of over fit in step 2 it improves very little bit with dropout prob 0.9 , stopped using dropout by setting it to 1.0 from reading understood since good regularization applied dropout becomes not useful.

4. Then came across this reading(http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf) and understood problem of using RELU which may have unbounded activations and applied local response normalization to normalize RELU activation
5. Instead Dropout showed no effect, used L2 regularization, which reduced gap between Validation and training accuracy, final validation accuracy
6. Validation accuracy is increased to 0.95 and accuracy on training set is 0.99
7. EPOCHS changed from 10 to 20 and Validation accuracy is ~9.6

The code for calculating the accuracy of the model is located in the seventh cell of the Ipython notebook.

My final model results were:

- training set accuracy of : 0.996
- validation set accuracy of : 0.959
- test set accuracy of : 0.94

Test a Model on New Images :

Below are ten German traffic signs that I found on the web: scaled to 32x32 using image editor before loading.

The code for finding predictions of the test images from web is located in the eleventh and twelfth cell of the Ipython notebook.

Predictions performed by restoring the network previously stored during training and running the model on test images from web to perform predictions of the label

As in the below table, Accuracy of the prediction is 100%, all images high probability rightly detected correct images

Below table is the result of the predictions on test images: Total images 10 and prediction accuracy 90%, I tried to cover different category of traffic sign for testing

Image	Prediction
Road Work	Road Work
Slippery Road	Slippery Road
Pedestrians	Pedestrians
Speed limit(30 km/h)	Speed limit(30 km/h)
No Entry	No Entry
Speed limit(70 km/h)	Speed limit(30 km/h)
Turn right ahead	Turn right ahead
Stop	Stop
Road of way at the next intersection	Road of way at the next intersection
Wild Animals Crossing	Wild Animal Crossing

Table : Images vs predictions

Top 5 softmax probabilities :

Finding Top 5 softmax probabilities solution is implemented using tf.nn.top_k , this part of the code implemented in twelfth and plotting result in thirteenth cell.

Analysis of the images prediction accuracy:

Image 1(Roadwork): Prediction correctly detected right label 'Road Work' with high accuracy probability, for other labels probabilities are less than 0.1

Image 2(Slippery): Prediction correctly detected right label 'Road Work' with high accuracy probability, for other labels probabilities are less than 0.1

Image 3(Pedestrians): Prediction correctly detected right label 'Pedestrians with high accuracy probability,

Image 4(Speed Limit 50): Model is performed well here with high prediction accuracy and only one good candidate.

Image 5(No Entry) : Model is performed well here with high prediction accuracy and only one good candidate.

Image 6(Speed Limit 70) : This is the image model prediction is low, Problem could be this image required rotation and need to train more data as very small difference between speed sign images

Image 7(Turn right ahead) : Model is performed average here, probability is not so high and other label probabilities are also high

Image 8(Stop sign): probability for right label 'stop sign' is looks good

Image 9(Right of way at the intersection) : Model is performed well here with high prediction accuracy

Image 10(Wild animals crossing): The model manages to predict but not so good accuracy may because very close to other types of signs

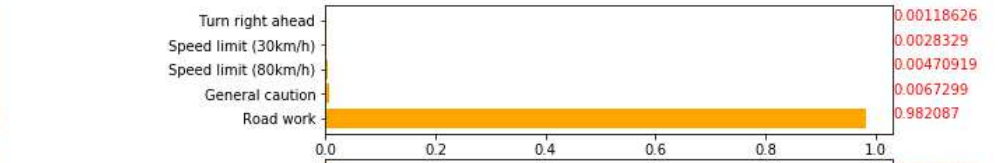
In below image all top predictions per image plotted as bar graph, this is useful to analyze and understand the test image prediction accuracy result

Comparisons of new traffic signs results to predicting on the test set:

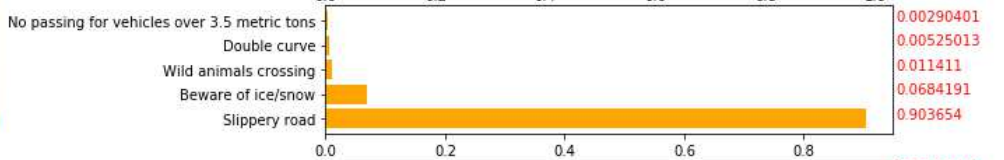
- Accuracy of the Test set is 94% and
- Accuracy of the predictions of new images is 90%

If I analyze the result mentioned in *Table: Images vs predictions*, The prediction is failed for speed sign(70 km/h) and it is predicted as Speed sign 30 km /h. In my point of view it is managed to categorize as speed sign correctly and only text prediction on this sign is failed so required to train model for more texts to able to classify different speed signs correctly, with this note I believe this may not be the result overfitting

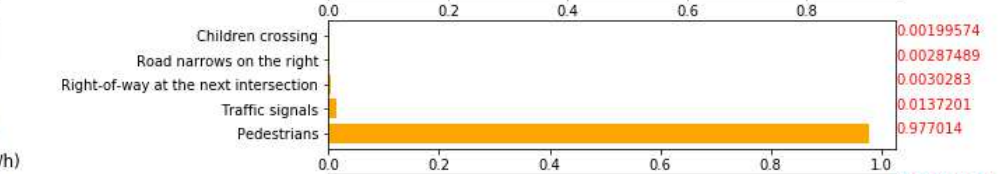
Road work



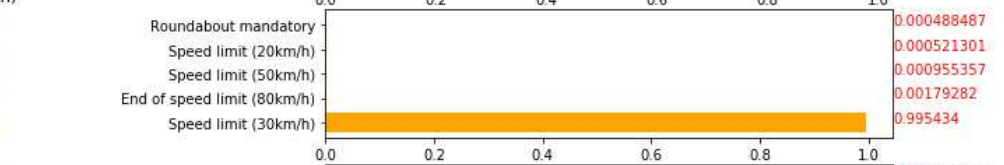
Slippery road



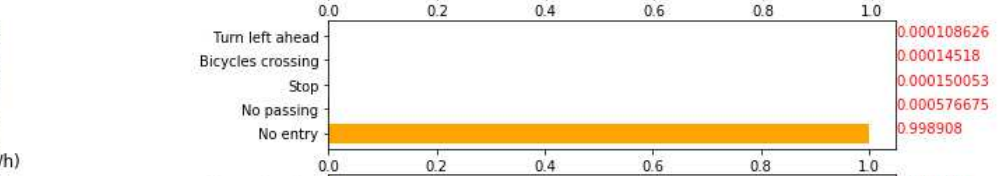
Pedestrians



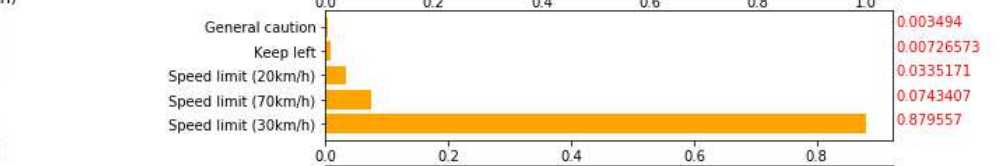
Speed limit (30km/h)



No entry

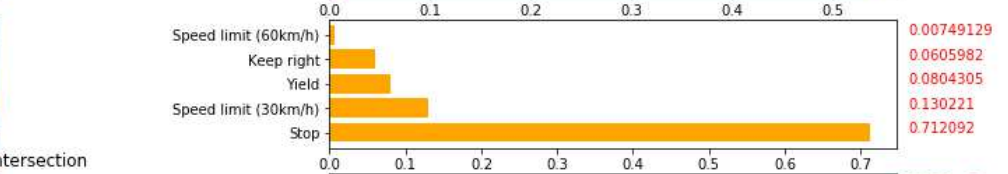


Speed limit (70km/h)

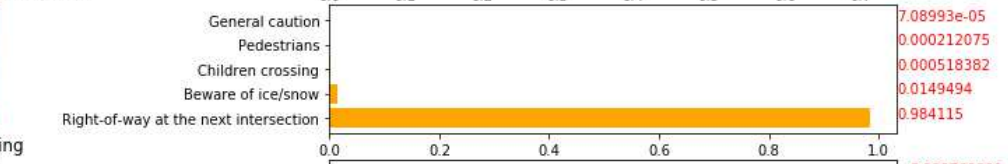


Turn right ahead

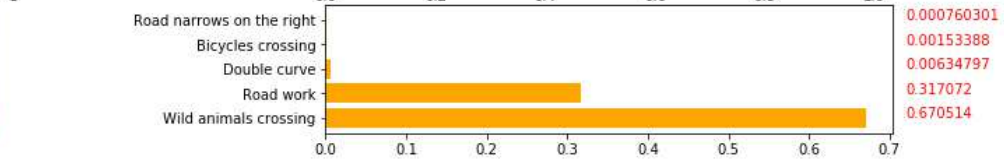
Stop



Right-of-way at the next intersection



Wild animals crossing



Improvements:

1. Image rotation
2. Image augmentation
3. Printing images in each stage of the network
4. Early stop when better peek is achieved