SELF DRIVING CAR NANODEGREE – TERM 1 / PROJECT2 – Traffic Signs classifier

ADITHYA RANGA
March 2016 – COHORT STUDENT

The goals / steps of this project are the following:

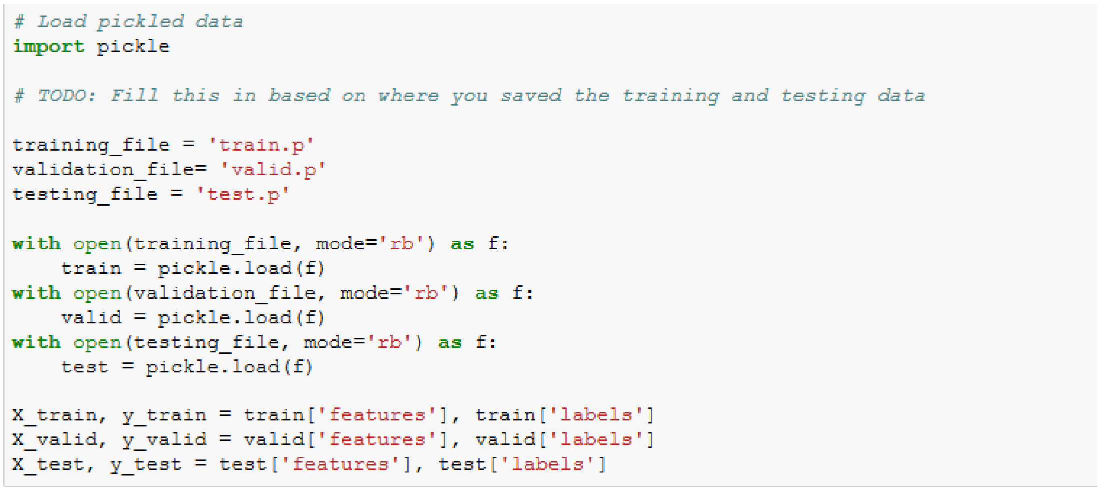
- Load the data set (see below for links to the project data set)
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

Section1: Data Summary and Visualization:

Part - 1
Basic Summary of the Data Set Used for building this Traffic Sign Classifier Pipeline:

The data primarily is of Traffic signs from Germany and the pickled version of this data is presented to work with. The original data set was downloaded from the following link: http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

The data primarily consists of Training, validation and test data sets separately. The following first section of the code loads and reads the datasets:



```python
# Load pickled data
import pickle

# TODO: Fill this in based on where you saved the training and testing data

training_file = 'train.p'
validation_file= 'valid.p'
testing_file = 'test.p'

with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)

X_train, y_train = train['features'], train['labels']
X_valid, y_valid = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']
```

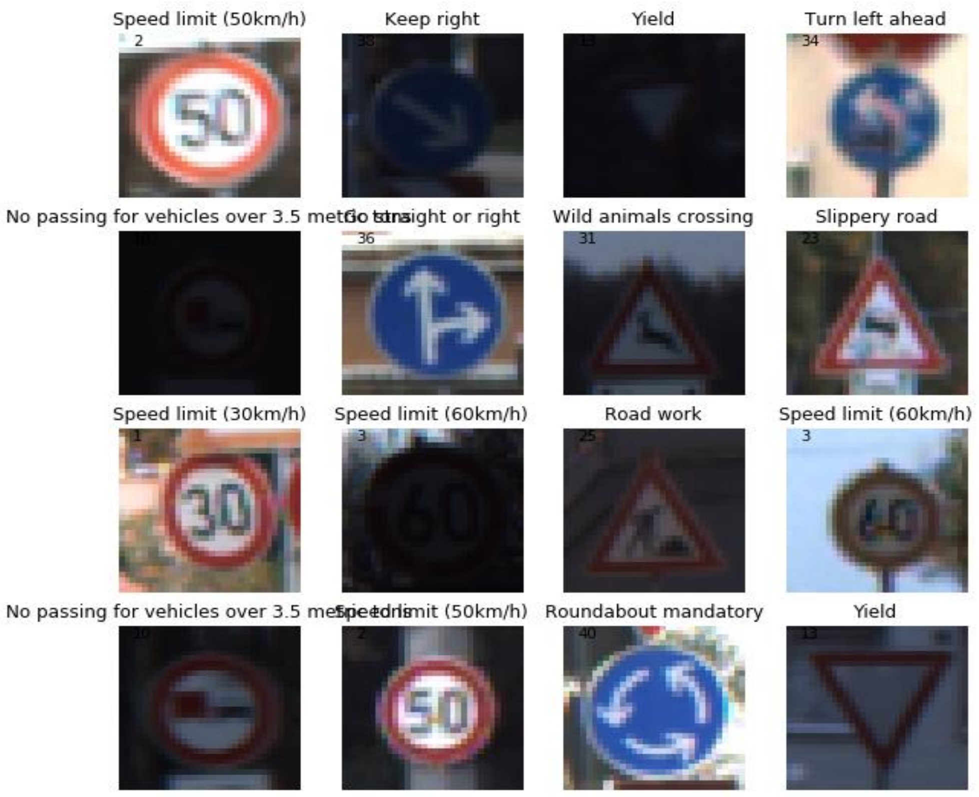
Using Numpy and pandas – the basic summary of the dataset is presented as showing in the following code section

```
           ClassId,SignName
0   0,Speed limit (20km/h)
1   1,Speed limit (30km/h)
2   2,Speed limit (50km/h)
3   3,Speed limit (60km/h)
4   4,Speed limit (70km/h)
Number of training examples = 34799
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

The size of the training set is about ~35k images and all the images are in RGB format as shown in one of the random images. Image from the training set to the right is converted to gray scale. Hence the original training, validation and test sets have shape (32, 32, 3). The total number of classes from the dataset is 43.

Part – 2
Visualization / Distribution of Training, validation and test Data:

Sample Visualization of Training Data and the corresponding labels:

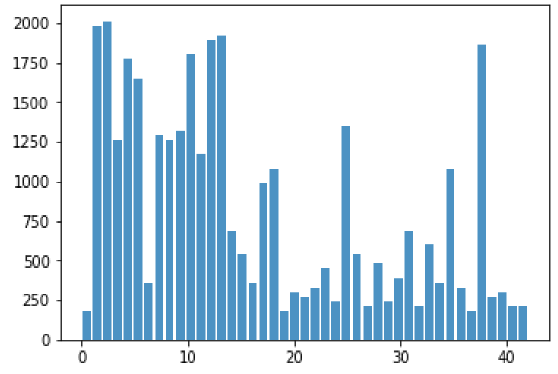


This is randomly printed from the training Dataset.

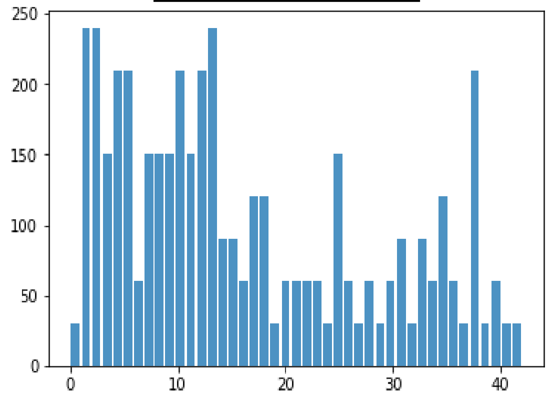
<u>Data Distribution:</u>

The bar charts below shows the distribution of training, test and validation data sets.

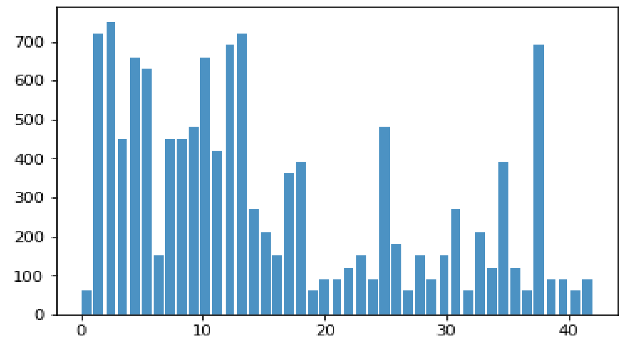
TRAINING DATA



VALIDATION DATA



TEST DATA



The Bar plots show the distribution of training, validation and test sets for each class. It is clear from the plots that the plots that some of the classes have dense population while others are sparsely distributed in the sets.

<u>SECTION – 2: Designing / testing Model Architecture:</u>

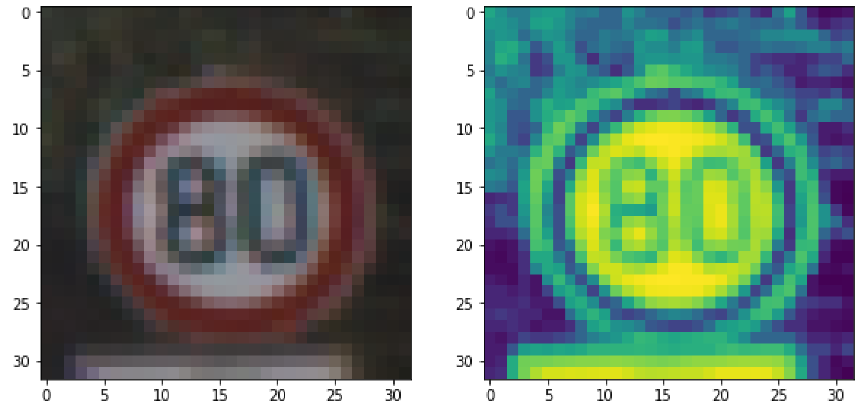
As visualized and explained – data set for traffic signs consists of color images and importantly the distribution of some of the classes is very poor. Considering these points I did write some logic to condition the existing data and then generate additional data from the existing sets and augment them such that the distribution of each class is more even.

Part – 1:

Data Preprocessing:

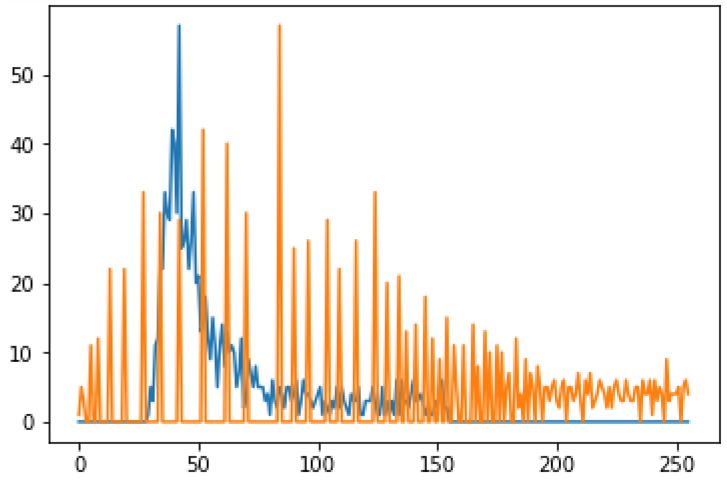
In this section I did implement the following logic:

Grayscale conversion: Converted the existing training data set from colored images to grayscale. This reduces the dimension of the data from (32, 32, 3) to (32, 32, 1). This provided me with some better performance during training as the numbers of input features are reduced.



```
Originaldata: (32, 32, 3)  - Processeddata: (32, 32, 1)
```

Histogram Equalization: I did perform histogram equalization on the input training data set. After carefully reading through the logic behind enhancing the contrast of the images – I implemented this logic. The contribution towards accuracy improvement on my validation set was not much through this logic but though the idea seemed to be the right thing to do.

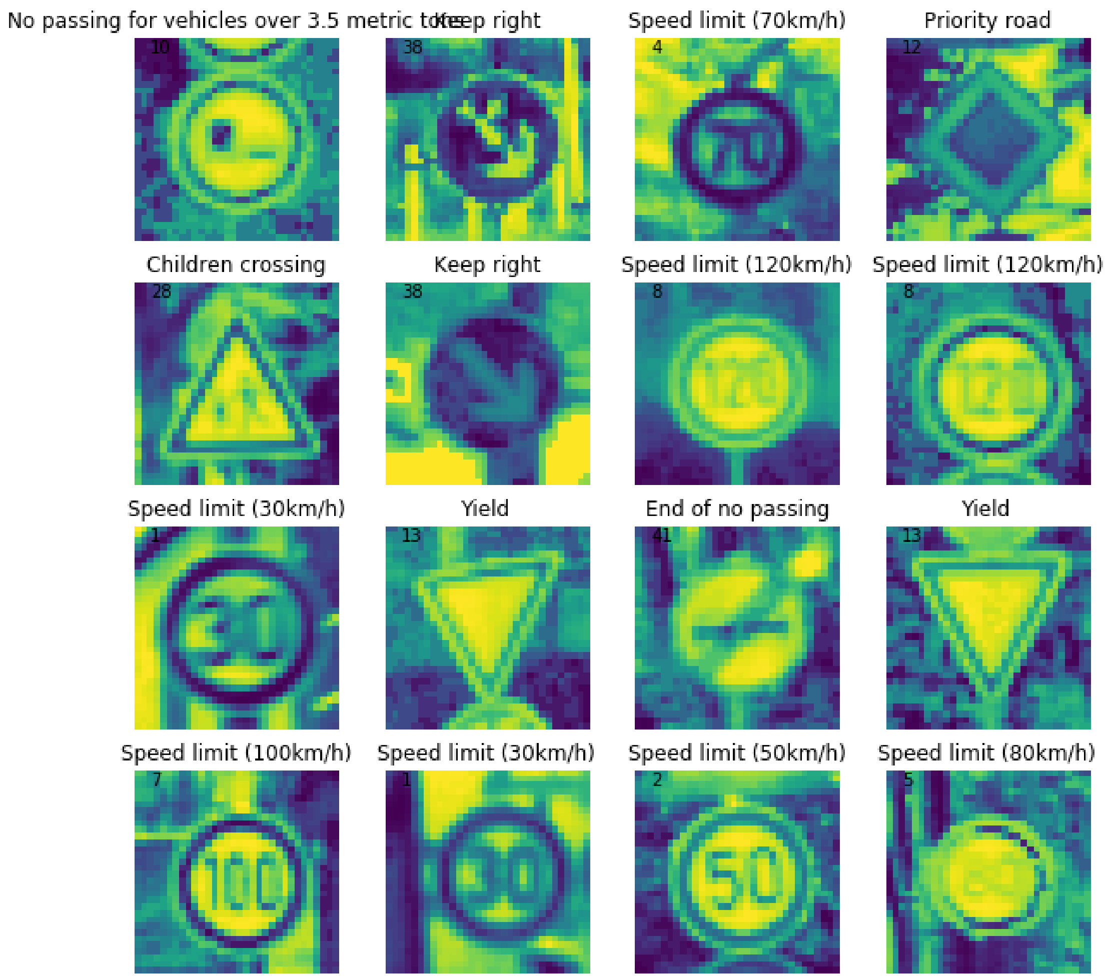


This above histogram plot shows the actual input image from our data set on the blue line and how locally concentrated where the image pixel values. After performing equalization the histogram is shown in the orange line with more evenly distributed pixel intensities. There is an equal chance in some cases this procedure could increase the contrast on background noise and hence should be carefully applied.

Normalization: The input images have pixel values in the range of [0 255]. Feature scaling is one such procedure which would normalize all the raw input data so that all the inputs are scaled. This would allow the gradient descent optimization search to converge faster likewise.
There are many suggested ways to perform this normalization, for this pipeline I implemented the standardization procedure where each input feature is scaled to have zero mean and unit variance.

$$`feature' = \frac{feature - mean(feature)}{std(feature)}$$

Training samples after grayscale conversion and standardization are as follows:



I tried normalizing the images using min max scaling and comparatively standardization have better validation and test set accuracy.

Part – 2:

Data Augmentation:

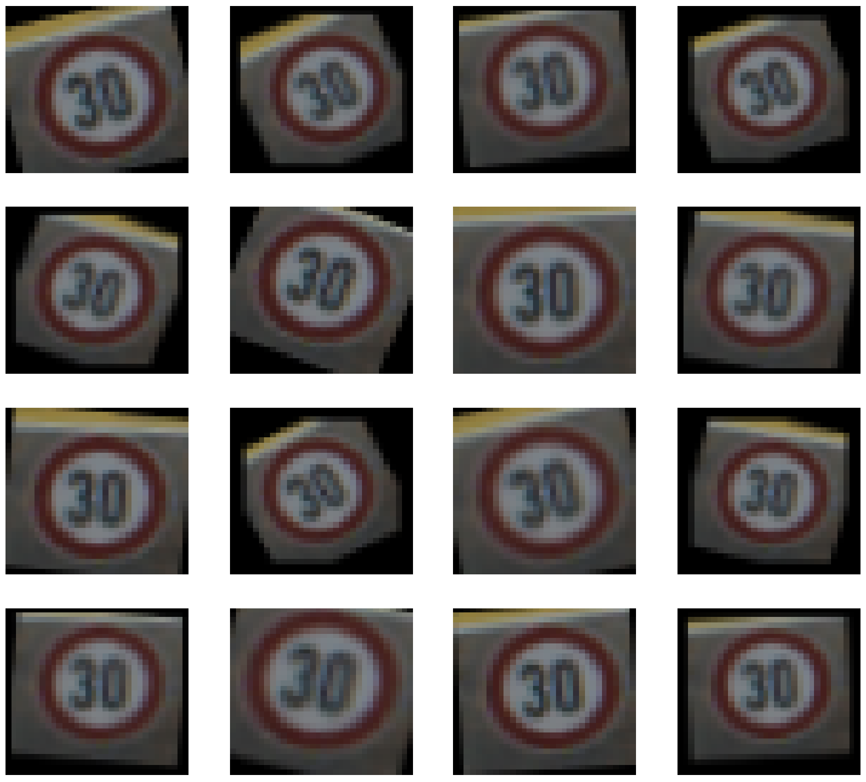
As shown from the dataset we have very weak distribution for some of the classes in the data and hence we need some additional data for these classes for better training out network and likewise additional data for validation set would also be helpful.

Data augmentation techniques as described in
http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf
Is implemented in my logic.

Three techniques used for data augmentation are:

- Translating image data set randomly in the range of (-2, 2) pixel positions
- Rotating the images from (-30, 30) range randomly
- Doing perspective transformation / zooming in in the range of (-2, 2) position values.

Please refer to the code for function **'datatransform (x,y)'** which does this transformation / jittering of the input data set.

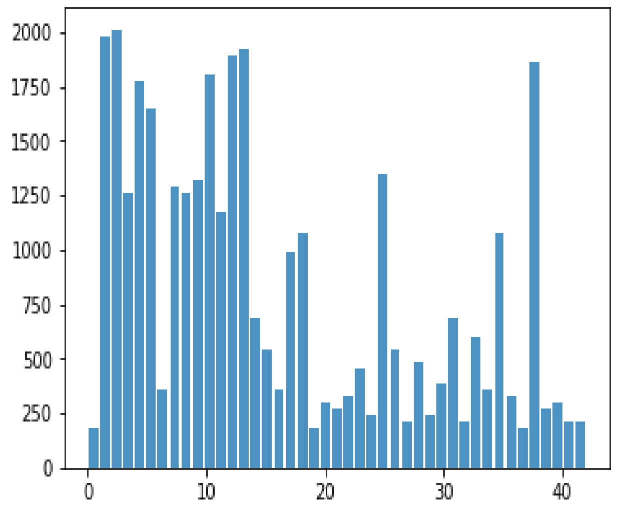


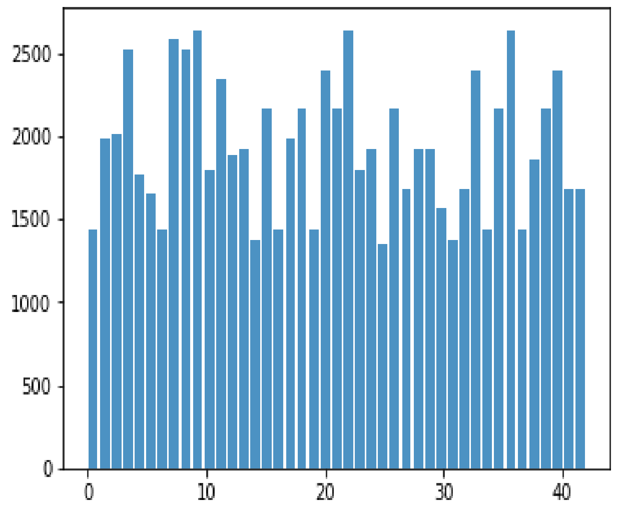
The logic in the code defined in the function “**dataaugment”** generates fake data by calling the transformation function as shown. This function generates fake data and append to the original training data sets.

Also the logic refers to the distribution of each class from the original training data set to generate fake data while ensuring that all the classes in the augmented data are evenly distributed.

Distribution of each class in the augmented dataset is shown in the second bar graph.

```
Actualtrainingdata:(34799, 32, 32, 3) - AugmentedData:(83486, 32, 32, 3)
```





The generated fake / augmented data is split off – to separate out the validation and test data.

MODEL ARCHITECTURE:

The final architecture used for my training was a 6 layer modified LeNet architecture as follows:

Layers	Type	Specs
Layer1	Convolution	Filter = 5 x 5 x 48
	Activation	ReLU
	Max Pooling	2 x 2
Layer2	Convolution	Filter = 5 x 5 x 144
	Activation	ReLU
	Max Pooling	2 x 2
	FLATTEN	Flatten – output 3600
Layer3	Fully Connected	FC 1800
	Activation	ReLU
	Dropout	0.5
Layer4	Fully Connected	FC 500
	Activation	ReLU
Layer5	Fully Connected	FC 500
	Activation	ReLU
	Dropout	0.5
Layer6	Fully Connected	FC 43

My approach started with the well know Lenet model architecture and there on made adjustments and improvements in an iterative manner.

- The original LeNet architecture during training and on validation sets resulted in a lower accuracy of about 85%.
- I increased the filter depth in convolution layers initially and saw an improvement in the validation and test accuracy of about 92% while using the data from the pre conditioning logic as described in section 2.

- Then as I added more nodes on the fully connected layer the validation accuracy improved to about 93.5 % along with an additional fully connected layer.
- To avoid overfitting the dropout layer was introduced in the FC layer of the model and this provided great improvement in the validation accuracy to about 96%.

SECTION 3: TRAINING:

HYPERPARAMETERS:

The following Hyper parameters have been used for training:

- Learning rate: 0.001
- EPOCHS: 20 (Considering the training time on CPU)
- Dropout probability = 0.5
- Batch Size = 128

For the training purpose I use the Adam optimizer logic from the tensorflow module which adaptively adjusts the learning rate.

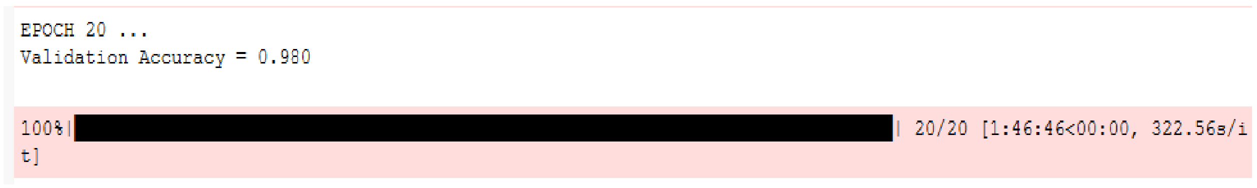
Batch size and Learning rate used in the final logic worked out really well and hence I fixed upon those.
For training the model mostly the normal tensorflow version on CPU was used and hence I restricted the EPOCHS to 20 considering the time for each iteration and load on the processor.
I did try out the g2x instance on AWS but even though the CUDA version of tensorflow was active and running the time for each iteration was still slow.

RESULTS/JOB LOGS:

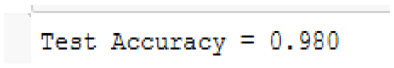
- The original LeNet architecture with the original data set converged at a validation accuracy of about ~ 86 %.

- Final Model structure with data conditioning (Section 2): By adding the dropout layer and the additional fully connected layer validation accuracy of the model turned out to be about ~ 95.6% for the original dataset.

- Final Model Structure with Augmented Data: Using the augmented Data I could achieve a training accuracy of 100%, validation accuracy of about 98% and test data accuracy of 98%. (Please refer to the Jupyter notebook for the code).



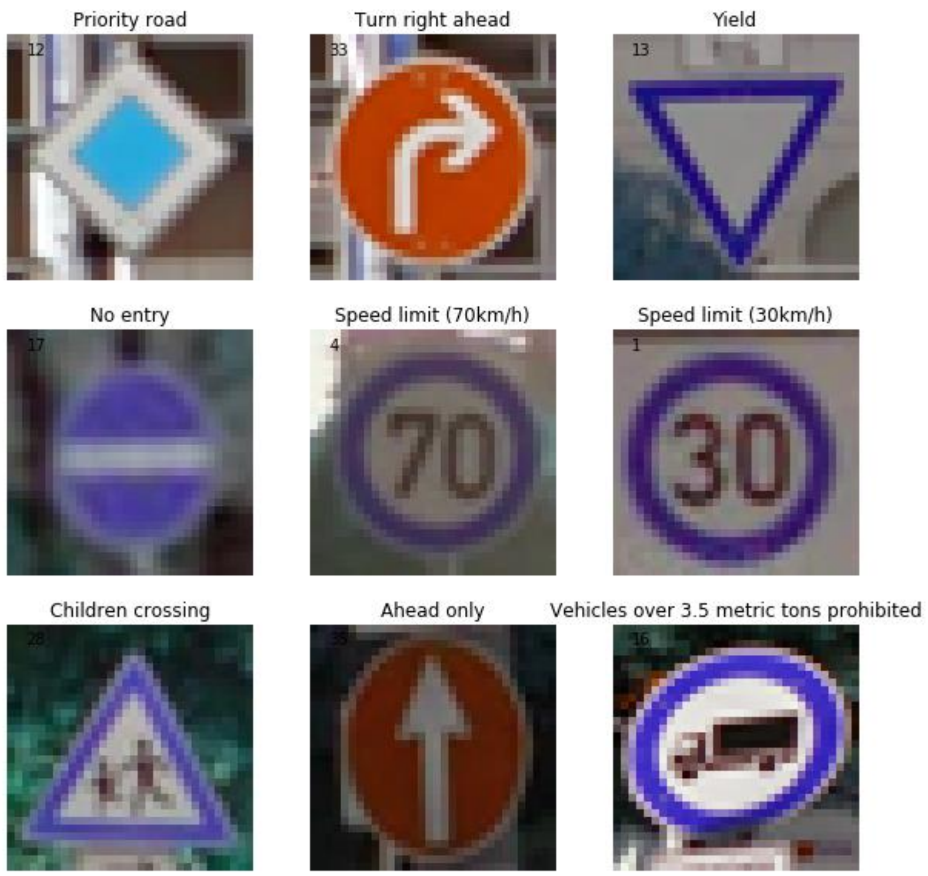
```
EPOCH 20 ...
Validation Accuracy = 0.980

100%|████████████████████████████████████████| 20/20 [1:46:46<00:00, 322.56s/i
t]
```



```
Test Accuracy = 0.980
```

SECTION 4:
Test the Model on New Images:

The test images for German traffic signs I found from web and the class peers in Slack are as follows:



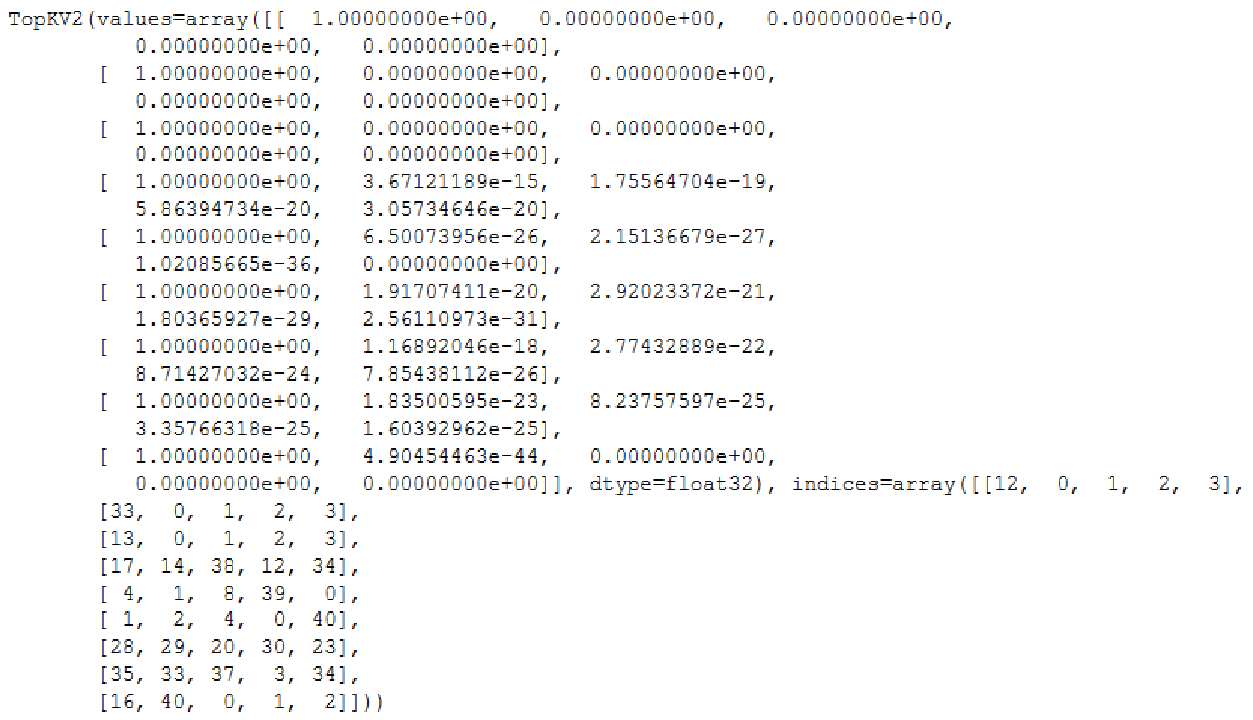
I used these 9 Test images to test the model architecture.

- These test images were processing to convert to grayscale and equalize the contrast and standardize the input features.
- The saved model in "./mymodel/Lenetaugmodel" was restored and the predictions for each image were calculated.

<u>RESULTS:</u>

- Achieved a Test accuracy of 100%.

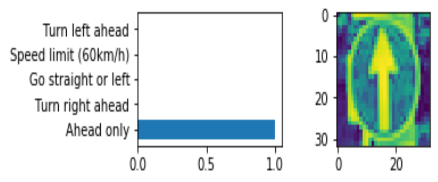
- The following are the top 5 softmax probabilities for test images:

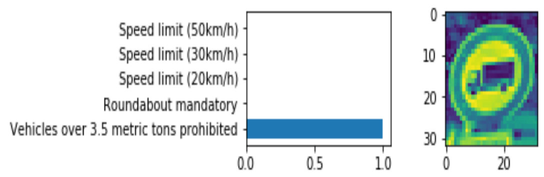


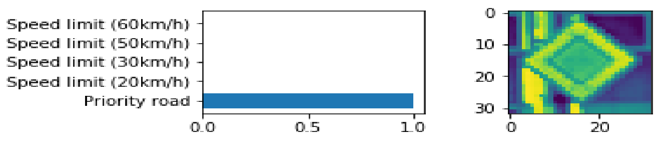
```
TopKV2(values=array([[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00,   0.00000000e+00],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00,   0.00000000e+00],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00,   0.00000000e+00],
       [  1.00000000e+00,   3.67121189e-15,   1.75564704e-19,
          5.86394734e-20,   3.05734646e-20],
       [  1.00000000e+00,   6.50073956e-26,   2.15136679e-27,
          1.02085665e-36,   0.00000000e+00],
       [  1.00000000e+00,   1.91707411e-20,   2.92023372e-21,
          1.80365927e-29,   2.56110973e-31],
       [  1.00000000e+00,   1.16892046e-18,   2.77432889e-22,
          8.71427032e-24,   7.85438112e-26],
       [  1.00000000e+00,   1.83500595e-23,   8.23757597e-25,
          3.35766318e-25,   1.60392962e-25],
       [  1.00000000e+00,   4.90454463e-44,   0.00000000e+00,
          0.00000000e+00,   0.00000000e+00]], dtype=float32), indices=array([[12,  0,  1,  2,  3],
       [33,  0,  1,  2,  3],
       [13,  0,  1,  2,  3],
       [17, 14, 38, 12, 34],
       [ 4,  1,  8, 39,  0],
       [ 1,  2,  4,  0, 40],
       [28, 29, 20, 30, 23],
       [35, 33, 37,  3, 34],
       [16, 40,  0,  1,  2]]))
```

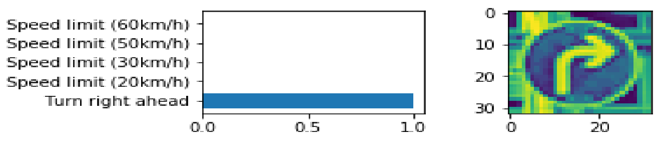
The predictions turned out to be exactly matching the input test images as shown in the top 5 softmax values along with the labels.

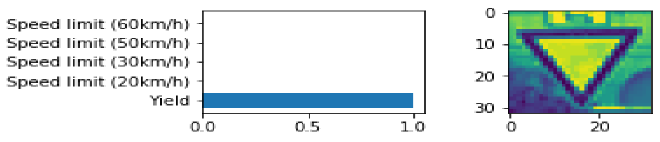
The bar graph below shows the visual representation of the predictions along with the images.

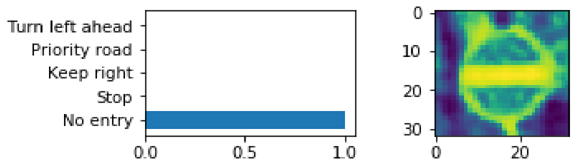

Turn left ahead
Speed limit (60km/h)
Go straight or left
Turn right ahead
Ahead only
0.0
0.5
1.0
0
10
20
30
0
20

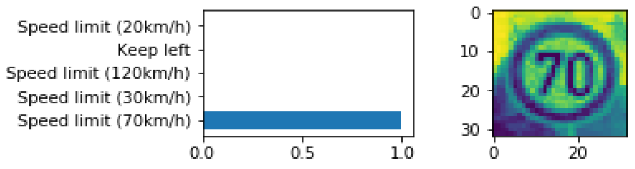

Speed limit (50km/h)
Speed limit (30km/h)
Speed limit (20km/h)
Roundabout mandatory
Vehicles over 3.5 metric tons prohibited
0.0
0.5
1.0
0
10
20
30
0
20

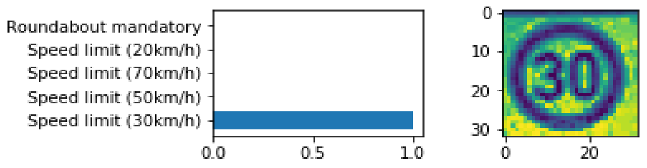

Speed limit (60km/h)
Speed limit (50km/h)
Speed limit (30km/h)
Speed limit (20km/h)
Priority road
0.0
0.5
1.0
0
10
20
30
0
20

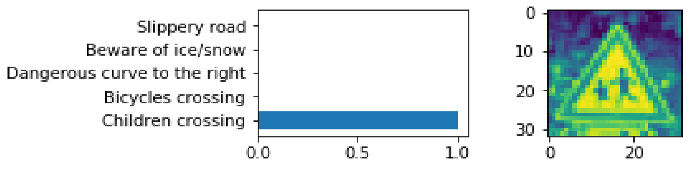

Speed limit (60km/h)
Speed limit (50km/h)
Speed limit (30km/h)
Speed limit (20km/h)
Turn right ahead
0.0
0.5
1.0
0
10
20
30
0
20


Speed limit (60km/h)
Speed limit (50km/h)
Speed limit (30km/h)
Speed limit (20km/h)
Yield
0.0
0.5
1.0
0
10
20
30
0
20


Turn left ahead
Priority road
Keep right
Stop
No entry
0.0
0.5
1.0
0
10
20
30
0
20


Speed limit (20km/h)
Keep left
Speed limit (120km/h)
Speed limit (30km/h)
Speed limit (70km/h)
0.0
0.5
1.0
0
10
20
30
0
20


Roundabout mandatory
Speed limit (20km/h)
Speed limit (70km/h)
Speed limit (50km/h)
Speed limit (30km/h)
0.0
0.5
1.0
0
10
20
30
0
20


Slippery road
Beware of ice/snow
Dangerous curve to the right
Bicycles crossing
Children crossing
0.0
0.5
1.0
0
10
20
30
0
20