Traffic Sign Recognition

Introduction

In an autonomous car driving mode, it is required that vehicle should identify traffic sign posted on the road through images that is captured using camera mounted on the vehicle. The identification process goes through identification pipeline which includes image capture, dividing image area into small patches/windows(using sliding window), and feeding them into a classifier system which in turn returns the identified class.

The goal of the project is to design a classifier that takes small 32x32x3 colored image representing **German traffic sign** and identifies the class, the image belongs to. In the current project, the classifier is designed using non linear neurons and are arranged as Convolution network(CNN). The ConvNet has artificial neurons arranged in multiple layers as a 3-d volume shaped layer, followed by 1-d arrangement of fully connected neurons. In theory, artificial neurons are computational nodes, which take input from previous layer neurons and produce weighted sum of the inputs (linear neuron) and then introduce some non-linearity via different non linear function such as Relu, sigmoid, tanh (non-linear neuron). The implementation is done in python programming language using Tensorflow framework.

Architectural details of CNN used in the project

In general, ConvNet consists of multiple layers that can be categorized as input/output layer, convolution layer, pooling layer and 1-d fully connected layer. Below is the summary of layers used in the design:

1. Input layer neurons: It takes raw pixel value of color image(32x32x3) as inputs and pass them to their output tensor which act as an inputs to next layer.
2. Convolution layer1: 5x5x18 conv filters with stride 1 along all dimension are used to generate 28x28x18 volume neurons from the previous input layer using “VALID” padding method as described in tf.nn.conv2d().
3. Relu layer2: It converts previous layer input volume as non-linear neuron volume of same size and shape
4. Convolution layer3 : 5x5x18 conv filters are used to generate 28x28x18 volume neurons using “SAME” padding. This layer is used to augment, since layer1 depth is small. It also allows to split one large volume layer into small volume layers
5. Relu Layer 4: It converts Layer3 output into non-linear neuron and produces 28x28x18 volume
6. Convolution layer5 : 5x5x18 conv filters with stride 1 along all the dimension are used to generate another augmented 28x28x18 volume neurons.

7. Relu Layer 6: It converts Layer5 output into non-linear neuron and produces 28x28x18 volume
8. Pooling Layer 7: The output of layer 6 is sub-sampled using Max Pool method and it produces 14x14x18 volume
9. Convolution Layer 8: 5x5x28 conv filters were used to produce 10x10x28 volume layer using “VALID” padding
10. Relu Layer 9 : It produces 10x10x28 volume of non linear neurons
11. Convolution Layer 10, Relu Layer 11 , Conv Layer 12, Relu Layer 13 : These layers are used since Layer 8 depth is small. It essentially allows to split one large volume into small volume layers. These layers produce 10x10x28 volume at the end
12. Pooling Layer14: It produces 5x5x28 sub-sampled neurons from previous layer
13. Fully connected layer 15. It arranges the 3d volume of 5x5x28 neurons as 1-d 700 neurons.
14. Fully Connected layer 16: It contains 120 neurons arranged as 1-d neurons. It takes input from “layer 15” and produces non-linear neurons of size 120
15. Fully Connected layer 17: It contains 84 neurons arranged as 1-d neurons, This layer produces non-linear neuron which takes input form “layer 16”
16. Fully Connected output layer 18: It contains 43 neurons arranged as 1-d neurons. This is the final output layer where each neuron acts as a linear neuron providing weighted sum of its input which is used to calculate class probability by feeding it to softmax function

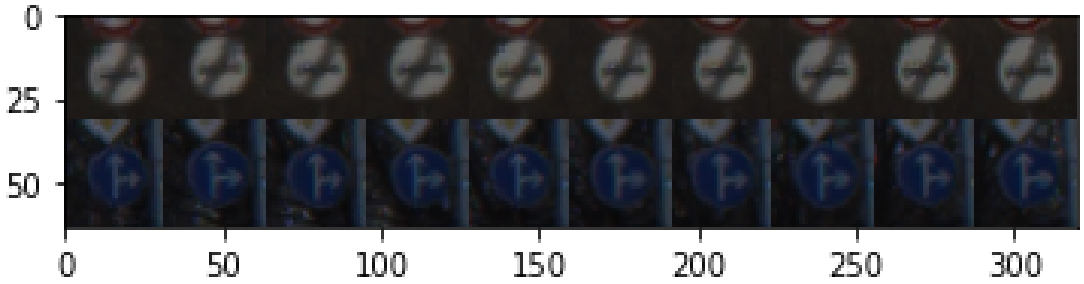
The above architecture is arrived by experimentation. The number of layers, neurons count, learning rate, regularization are considered as hyper parameters of model and they are fine tuned using cross validation method. **Ln[9]** in the code describes the weights and bias variables and their dimensions across layer

Steps Involved

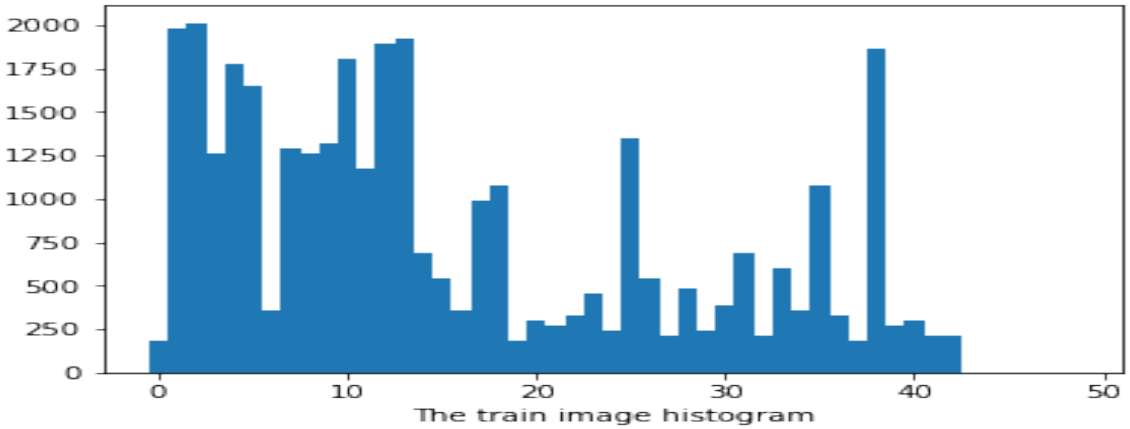
Below are the steps followed while designing the network:

Data Visualization and processing

1. *Data visualization:* It is always useful to visualize the data beforehand. It includes, finding whether training data set includes enough examples from all the classes to be identified. Histogram of the examples's class count is plotted to find the number of training examples available for each class. **Ln[3]** of the code shows some of the example in training data and category/class histogram(plots between example counts and category/class bin)



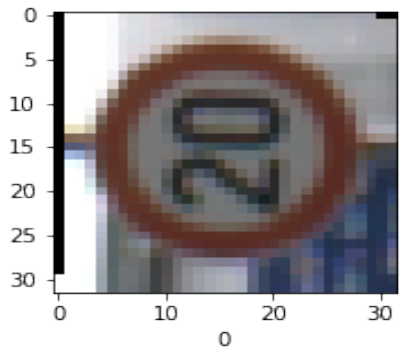
The train image showing #20 sign images

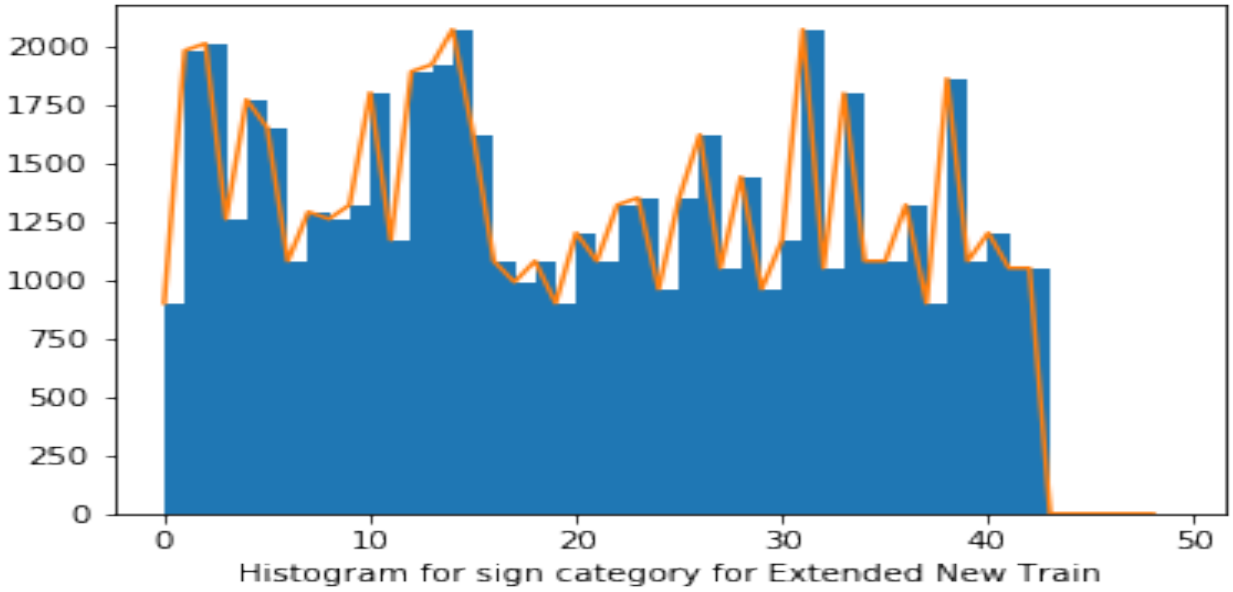


The train image histogram

1. *Data Pre-Processing:*

As we can find from the histogram plot, the training data set is biased or skewed towards some of the classes. Not all classes are equally represented through examples and may result in over-fitting of the model and it will not generalize well. So in order to overcome such problem, more data are collected or data set is augmented by creating new test examples using existing examples from training test vectors. Affine transform can be used to generate new examples. In the current project, rotation method is used to add new examples in data set as shown below. **Ln[4], Ln[5]** of the code show data augmentation steps.





Normalization of training Data: In order to achieve fast convergence to optima, the training example images' pixel value is normalized by subtracting mean of the train data pixel values and diving it to their variance. Though, I believe, it is not required for images at input layer, since each pixel values range from 0-255 considering 8 bit pixel depth per component. Normalization is generally useful when each feature set is represented on different scale **Ln[6]** in the code shows the normalization process.

Model Training

The training and prediction model are implemented in python using Tensorflow framework as shown in **ln[8], ln[9], ln[10], ln[12], ln[13], ln[14]**(shown as ln[] since training was stopped after 20 epochs) of code. Stochastic Gradient Descent(SGD) is used with mini batch size of 128 examples to update the weights and bias vectors. Cross entropy loss function is used as Error/Loss/Cost function along with L2 norm of weights to achieve regularization. Adaptive step update along with momentum (ADAM)approach is used to update weights for every descent. It allows to speed up and smooth

convergence of weight and bias vectors to their optima. Below is flow for training the model:

for epoch in range(#iterations):

    Randomize the training examples

    calculate train accuracy using 12000 train examples

    for batches in [train examples]:

        perform gradient descent/calculate gradient vectors using back propgation

        w(i, j; t+1):= w(i, j; t) – learning_rate * dE/dw

    calculate validation accuracy achieved after training

Due to high computation power requirement, the training is stopped at the end of 20th epocs. The weight and bias vectors are saved after each epoc.

Other details of the training is as below:

1. original training examples: 34,799
2. Total number of training examples used after addition of rotated images: 58077
3. Validation examples = 4410
4. Test examples = 12630
5. learning rate: 0.01
6. regularization parameter lambda = 0.2

Performance evaluation:

Once Model is trained, final accuracy is calculated on original training (without addition of new examples), validation and test data sets. Code is shown in **ln[15].** The code clears the weight and bias vectors and reload trained parameters from one of the best performing saved weights and bias checkpoint file . For example Epoch 18 gives the best validation accuracy of 97%,. Below is the summary of normalized accuracy achieved as shown in **ln[15]**

Train Accuracy = 0.996

Validation Accuracy = 0.970

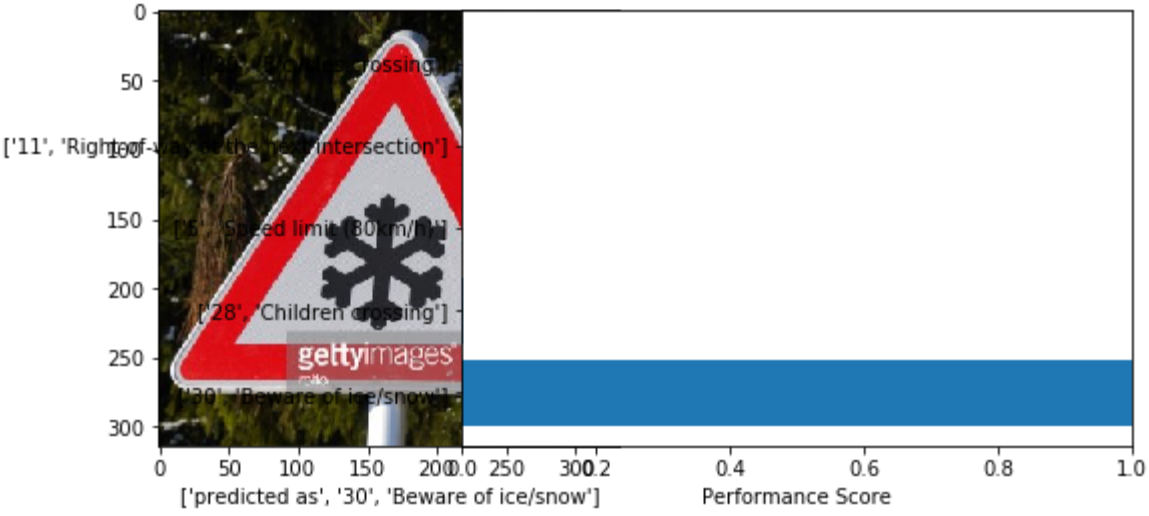
Test Accuracy = 0.950

Above performance shows some amount of overfitting of the model despite the usage of L2 norm weights regularization method

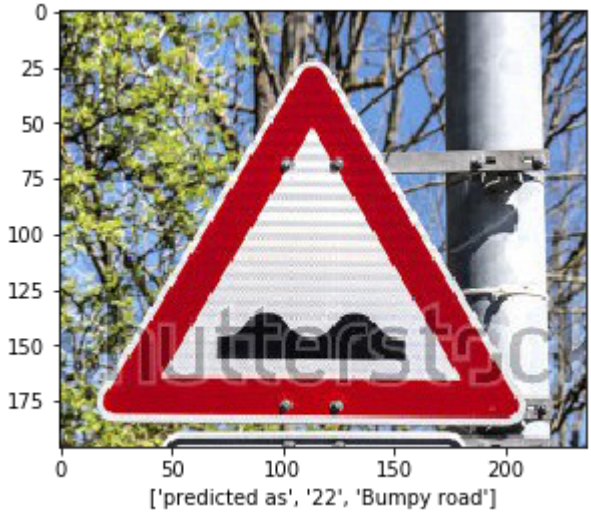
Model performance evaluation on new images:

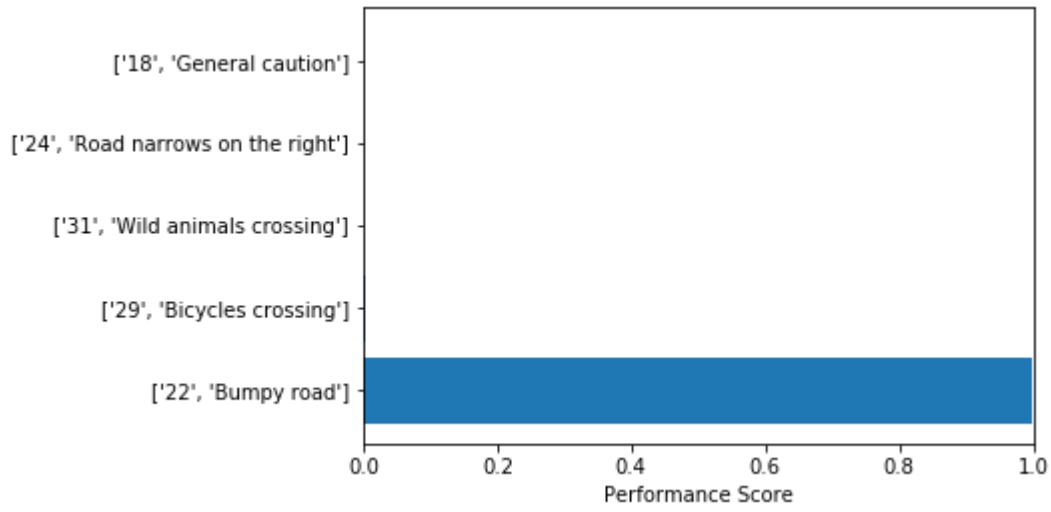
12 new traffic signs are downloaded and they are resized into 32x32x3 for the classification purpose. Original and resized images are shown in console of **ln[18]** of the code. The resized images are fed to classifier and classifier provides top 5 probabilities using Softmax function as shown in **LN[62].** The values are as shown below:

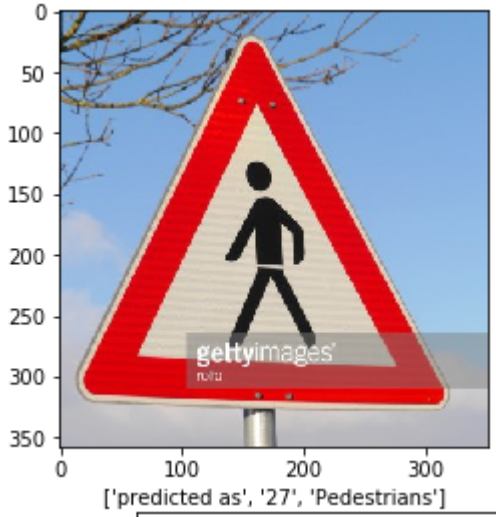
```
Prediction on New Image based on model that was trained using Augmented Data
through Rotation
Top-5 prob: [
 [ 0.99948549  0.00042639  0.0000621   0.00000872  0.00000585]
 [ 0.99707377  0.00216398  0.00058559  0.00010473  0.00007163]
 [ 0.9999845   0.00001539  0.00000014  0.00000001  0.        ]
 [ 0.99728978  0.00178313  0.00069286  0.00009488  0.00006946]
 [ 1.          0.00000001  0.          0.          0.        ]
 [ 0.72159892  0.26265711  0.01503842  0.00050192  0.00007865]
 [ 0.98862177  0.00996354  0.0004697   0.0003697   0.00027786]
 [ 0.99283302  0.0040791   0.00305657  0.00001895  0.0000069 ]
 [ 0.99996173  0.00002183  0.00001402  0.00000144  0.00000043]
 [ 0.24147727  0.22648677  0.12746343  0.08267608  0.05410266]
 [ 0.99950635  0.0002294   0.00011436  0.00006078  0.00003237]]

Top-5 prob indices [
 [30 28  5 11 29]
 [22 29 31 24 18]
 [27 24 18 11  4]
 [14 29 15  0  1]
 [ 1  2 16 32  6]
 [27 28 24 18 20]
 [16 28 30  2  8]
 [ 0  1  5  2 27]
 [28 30 29  1 21]
 [20 31  6  1 32]
 [25  6 21 31  0]]
```

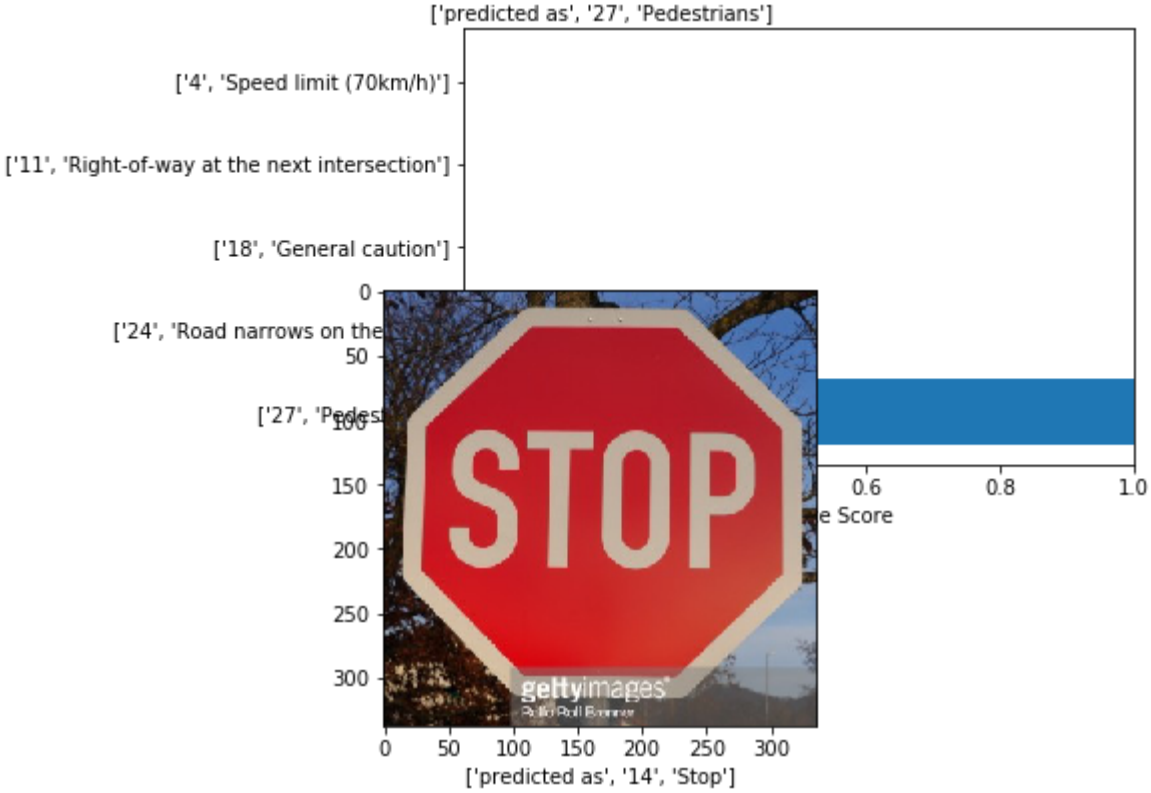


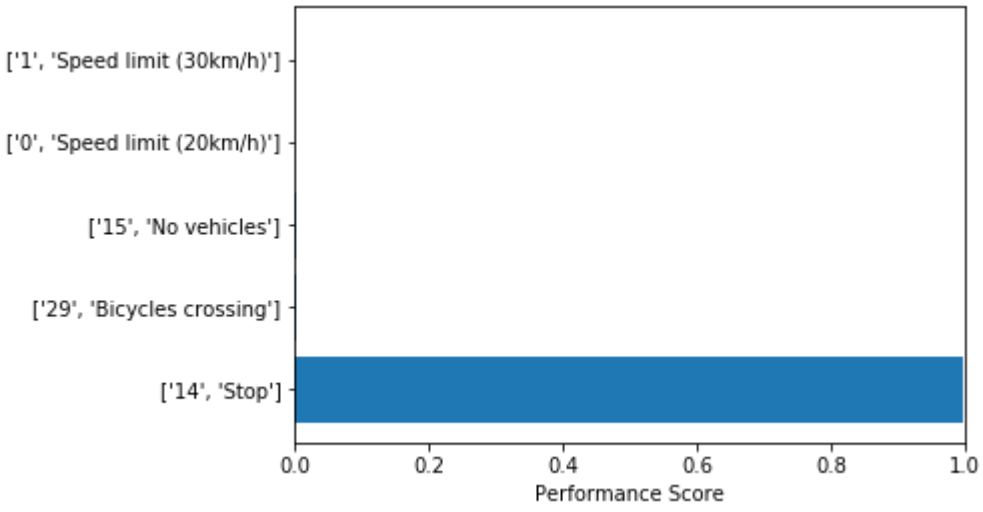
LN[62] console images


0
25
50
75
100
125
150
175
0
50
100
150
200
['predicted as', '22', 'Bumpy road']

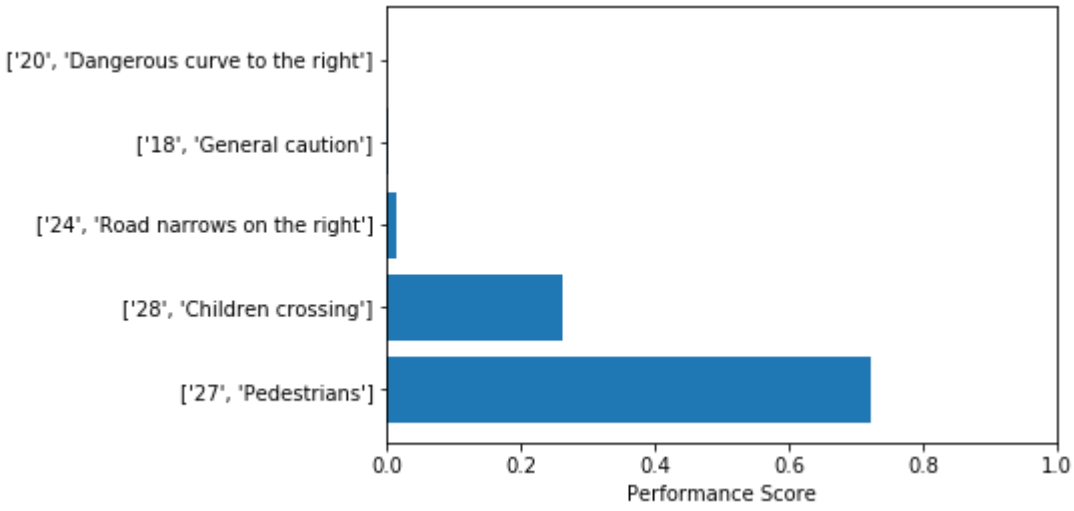

['18', 'General caution']
['24', 'Road narrows on the right']
['31', 'Wild animals crossing']
['29', 'Bicycles crossing']
['22', 'Bumpy road']
0.0
0.2
0.4
0.6
0.8
1.0
Performance Score


0
50
100
150
200
250
300
350
0
100
200
300
['predicted as', '27', 'Pedestrians']


['4', 'Speed limit (70km/h)']
['11', 'Right-of-way at the next intersection']
['18', 'General caution']
['24', 'Road narrows on the
['27', 'Pedes
0.6
0.8
1.0
e Score
0
50
100
150
200
250
300
STOP
0
50
100
150
200
250
300
['predicted as', '14', 'Stop']


['1', 'Speed limit (30km/h)']
['0', 'Speed limit (20km/h)']
['15', 'No vehicles']
['29', 'Bicycles crossing']
['14', 'Stop']
0.0
0.2
0.4
0.6
0.8
1.0
Performance Score


0
25
50
75
100
125
150
175
0
50
100
150
200
250
['predicted as', '27', 'Pedestrians']



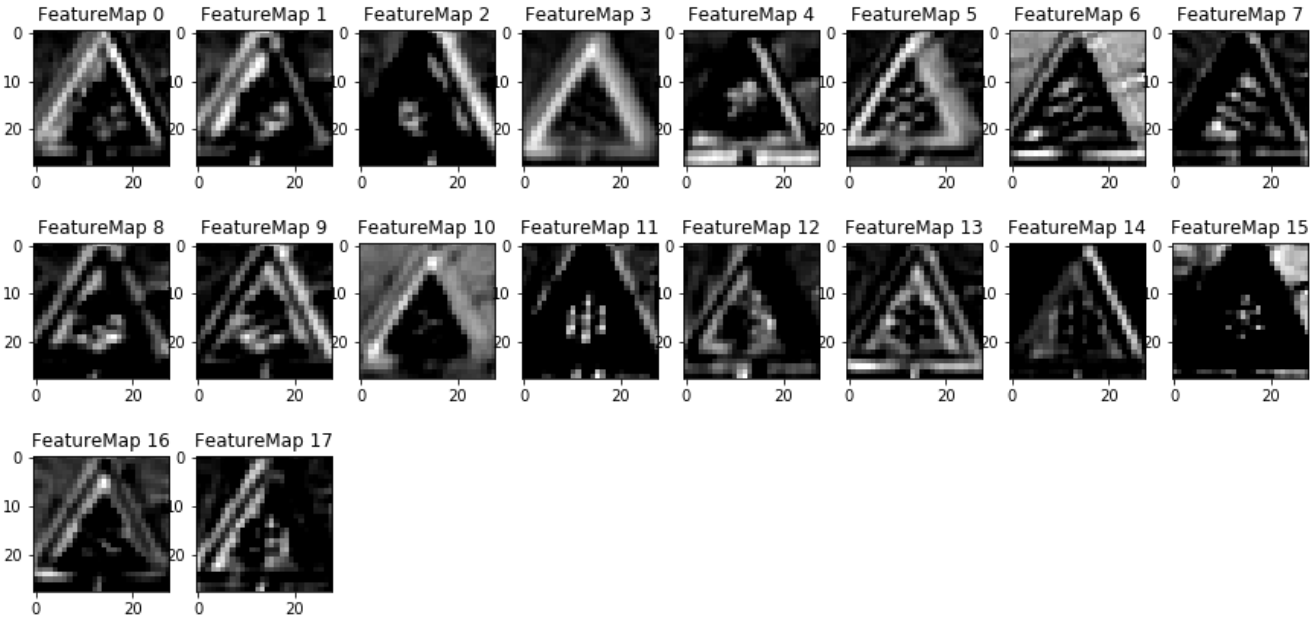
Summary for all the new images and analysis of some failure

traffic sign from new image	Model Prediction	Actual Sign	Possible Reason of Failure	Result
#1	Warning sign for ice/snow	Beware of ice/snow	NA	PASS
#2	Bumpy Road	Bumpy road	NA	PASS
#3	Pedestrian	Pedestrian	NA	PASS
#4	STOP	STOP	NA	PASS
#5	Speed limit 30km/h	Speed limit 30km/h	NA	PASS
#6	Pedestrian	Double curves first to right	The resized image used is sheared, stretched and of bad quality and would have confused the model The training data set should include affine transformed images	FAIL

#8	Vehicle over 3.5 metric tons prohibited	No entry for motor vehicles.	The resized image is of bad quality which makes it appear as heavy weight truck	FAIL
#9	Speed limit (20km/hr)	Speed limit (50km/hr)	This failure happened when the model is trained using training set examples augmented with rotated images for under represented classes. If model is trained without using rotation then it is classified correctly.	FAIL
#10	Children crossing	Children crossing	NA	PASS
#11	Dangerous curve to right	No motor vehicle allowed	The image has area that represents more background than actual sign. This could have been detected using sliding window method	FAIL
#12	Road Work	Road Work	NA	PASS

Internal feature map

Feature map is shown using one of the traffic sign image. Such images show that how each feature set has learned to identify different characteristic of the images. It is shown in **Ln[81**] of the code. Below is example for layer 1



Conclusion

Further improvement in the performance can be achieved by including affine transformed images like different rotation, shear and translation in the training examples. Increasing the number of examples in training set may prevent Variance/Overfit behavior of the model. Other measures like dropout, fine tuning of regularization parameter lambda, can be tried to reduce the overfit. It is also possible that after 100 epoch, the model would have provided better learned parameters.