# Traffic Sign Classifier

In the self-driving car once you are able to detect the lanes using OpenCv libraries, it is important to move next step as traffic sign classification and object detections. So presenting the step by step approaches taken to get the desired results.


Dataset used for algorithm implementation is from German Traffic Sign Benchmarks.
(http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset)

Below are the steps performed in this project:

- Load the dataset
- Visualize the dataset and check length of train, test, validate datasets.
- Design ,train and test the model architecture (LeNet-5)
- Use the model to make predictions on new images (Random 5 images from German Traffic Sign Dataset)
- Analyze the softmax probabilities of the new images.
- Summarize the results with a written report

## Step 1 : Dataset Summary

Load the dataset as per the data available and store it into X_train, y_train ,X_valid, y_valid, X_test, y_test wrt features and labels.

Also read **signnames.csv** file and store the data as dictionary wrt SignName.

**Note:**

Pickle module accepts any python object and convert it into string representation and dumps it into file by using dump function, this process is called pickling. While retrieving original python objects from stored string representations is called unpickling.

The pickled data is a dictionary with 4 key/value pairs:

- `'features'` is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).
- `'labels'` is a 1D array containing the label/class id of the traffic sign. The file `signnames.csv` contains id -> name mappings for each id.
- `'sizes'` is a list containing tuples, (width, height) representing the original width and height the image.
- `'coords'` is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image. **THESE COORDINATES ASSUME THE ORIGINAL IMAGE. THE PICKLED DATA CONTAINS RESIZED VERSIONS (32 by 32) OF THESE IMAGES**

Also check Number of training examples, testing examples, validation examples, and unique classes/labels in dataset using numpy function.

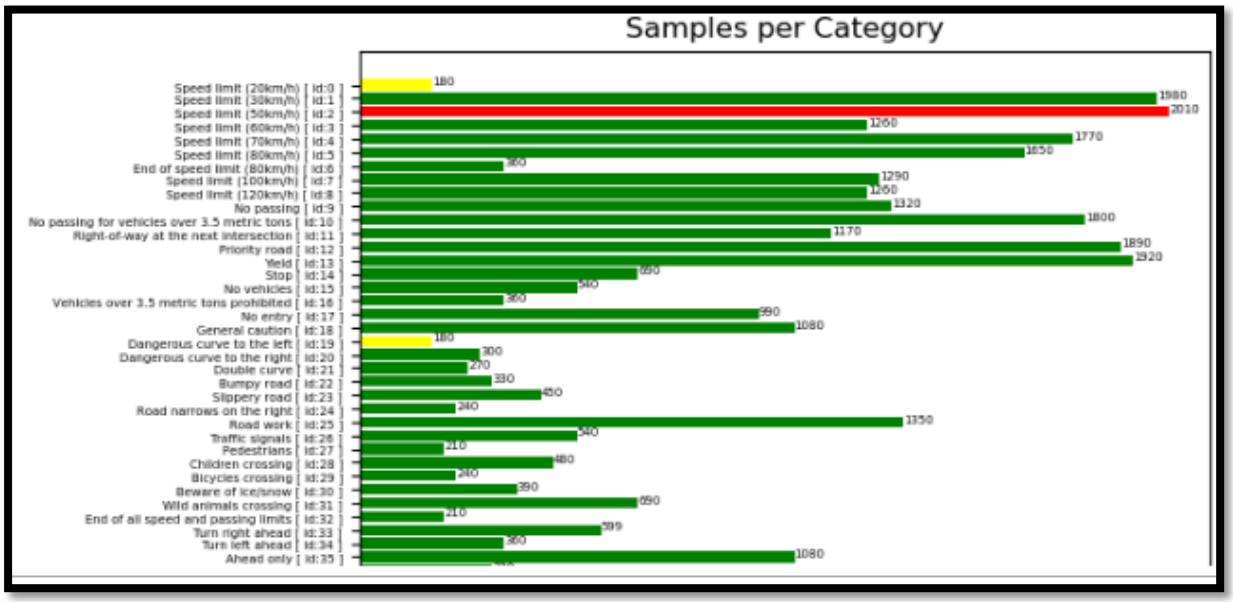
There are 43 unique traffic signs in the dataset so classes = 43

Each image in the dataset is 32 pixels x 32 pixels x 3 Channels (one each for RGB)

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

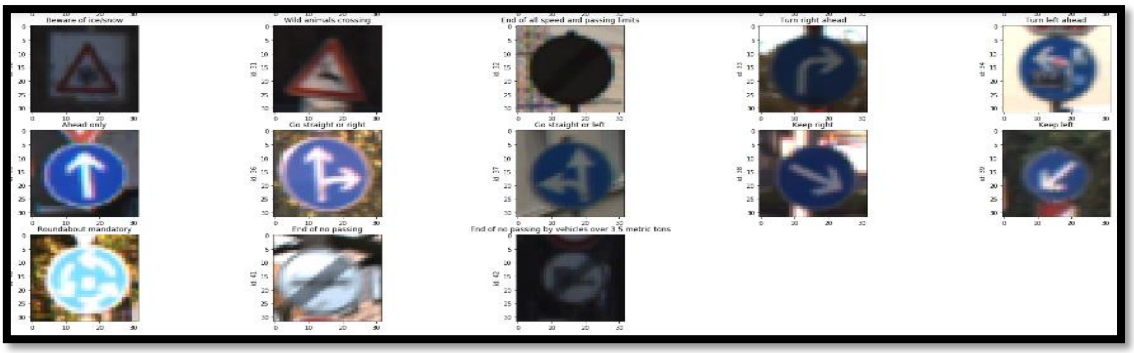
## Step 2 : Data exploration visualization

Get samples per category and category name from the dataset and plot how many samples are present in each category using matplotlib library.



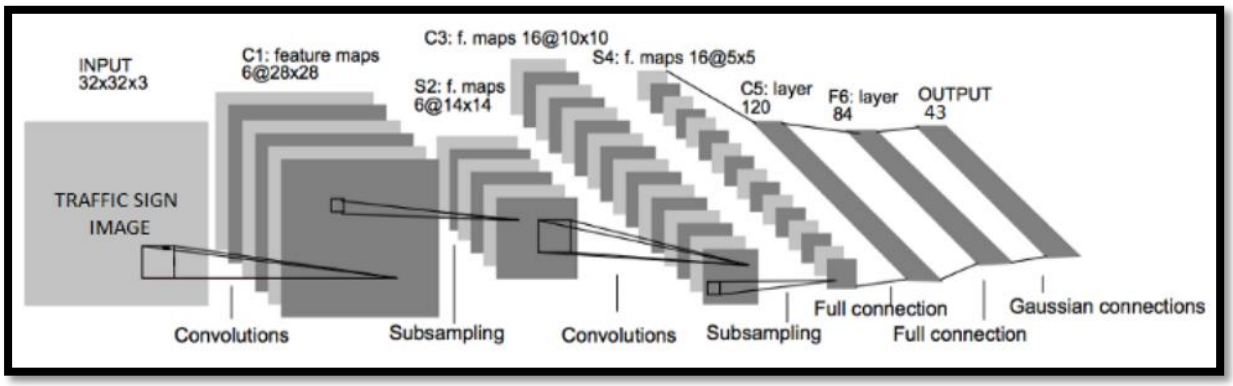

## Random Image from Each Category

Output a sample image from each category



## Step 3 : Design and Test a Model Architecture

As per the inputs in the LeNet-5 implementations - the image data should be normalized so that the data has mean zero and equal variance. For image data, (pixel - 128)/ 128 is a quick way to approximately normalize the data.




LeNet Model (Source: Yann LeCun)

Below are the steps performed before model architecture:

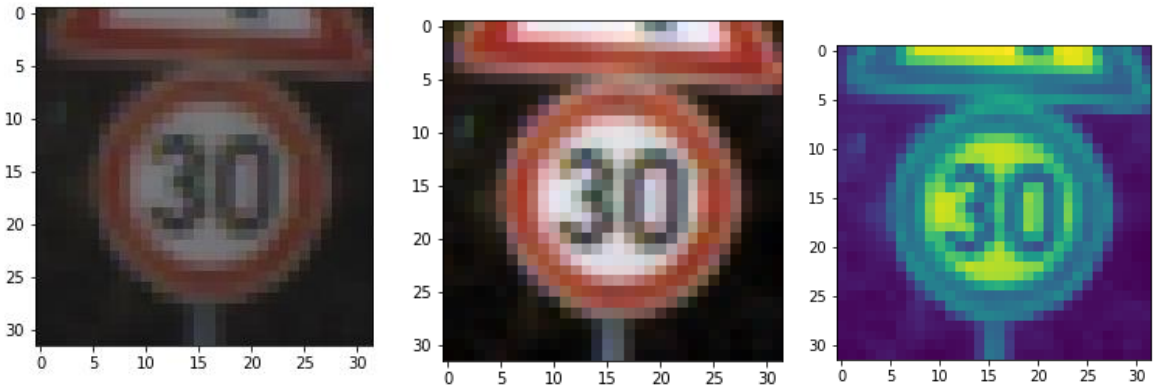
- Shuffle the data to avoid over fitting and getting it processed sequentially.
- Transform initial set of images so that they are ready to be fed to neural network

- Normalize image
- Convert RGB image to gray scale

- Make number of samples in each category equal
  - The data set has different number of samples for each category.
  - Function will transform the data set in a way that each category will contain the number of samples equal to maximum samples per category from the initial set.
  - This will provide an equal probability to meet traffic sign of each category during the training process.

```
print(image_shape_prep)

(32, 32, 1)
```



**Original Dataset Image| Normalized Image | GrayScaled Image**

- Define the model architecture:
  - LeNet-5 architecture is used.

- Steps inside LeNet-5 architecture :
  - Layer 1: Convolutional. Input = 32x32xchannels. Output = 28x28x6.
  - Layer 1: Activation.
  - Layer 1: Pooling. Input = 28x28x6. Output = 14x14x6.
  - Layer 2: Convolutional. Output = 10x10x16.
  - Layer 2: Activation.
  - Layer 2: Flatten. Input = 5x5x16. Output = 400.
  - Layer 3: Fully Connected. Input = 400. Output = 120.
  - Layer 3: Activation
  - Layer 4: Fully Connected. Input = 120. Output = 84.
  - Layer 4: Activation.
  - Layer 5: Fully Connected. Input = 84. Output = 10.

## Step 4: Train, Validate and Test the Model

A validation set is used to assess how well the model is performing. A low accuracy on the training and validation sets imply underfitting. A high accuracy on the training set but low accuracy on the validation set implies overfitting.

- Set placeholders for input images and output labels

- Set hyperparameters for training processes as learning rate(0.008), epochs(30), batch size(128), cross_entropy and so on.
- Evaluate the model
- Train the model ( Remember to shuffle the training dataset at start of each epochs to avoid overfitting)

```
Training...

EPOCH 1 ...
Train Accuracy = 0.891
Validation Accuracy = 0.815

EPOCH 2 ...
Train Accuracy = 0.967
Validation Accuracy = 0.868

EPOCH 3 ...
Train Accuracy = 0.977
Validation Accuracy = 0.869

EPOCH 4 ...
Train Accuracy = 0.989
Validation Accuracy = 0.917

EPOCH 5 ...
Train Accuracy = 0.993
Validation Accuracy = 0.923

EPOCH 6 ...
Train Accuracy = 0.994
Validation Accuracy = 0.931

EPOCH 7 ...
Train Accuracy = 0.996
Validation Accuracy = 0.926
```

```
Validation Accuracy = 0.952

EPOCH 27 ...
Train Accuracy = 0.999
Validation Accuracy = 0.956

EPOCH 28 ...
Train Accuracy = 0.999
Validation Accuracy = 0.958

EPOCH 29 ...
Train Accuracy = 1.000
Validation Accuracy = 0.966

EPOCH 30 ...
Train Accuracy = 0.999
Validation Accuracy = 0.959

Model saved
```

- Evaluate Trained Model Using Test Samples

```
INFO:tensorflow:Restoring parameters from ./model.ckpt
Test Accuracy = 0.929
```

When re-trained the same dataset, accuracy is:

```
INFO:tensorflow:Restoring parameters from ./model.ckpt
Test Accuracy = 0.940
```

## Step 5: Test a Model on New Images

Downloaded five pictures of German traffic signs from the web and used on model to predict the traffic sign type.

**Results are:**


Predict the Sign Type for Each Image

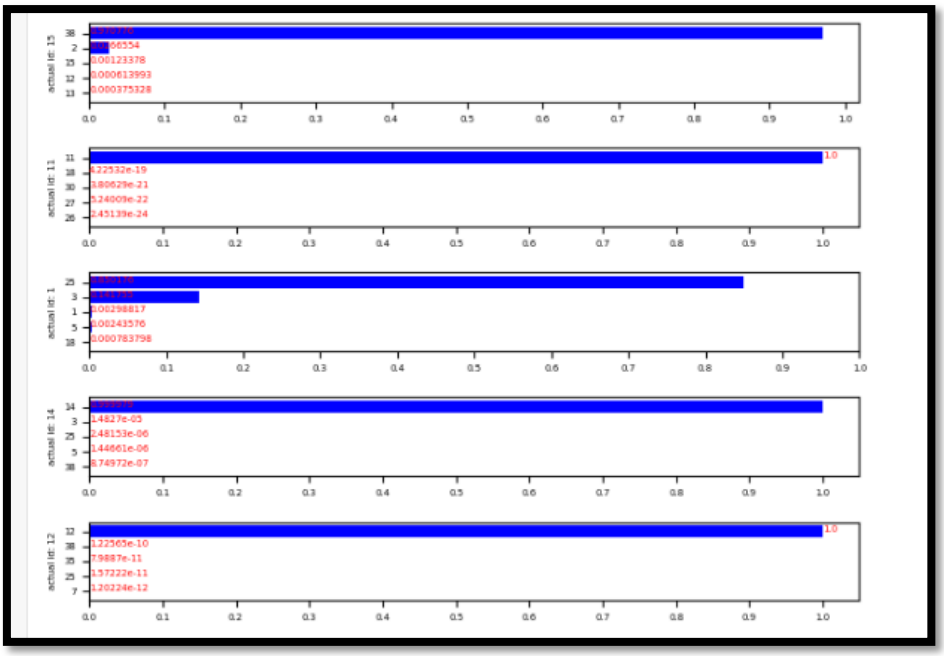
```
INFO:tensorflow:Restoring parameters from ./model.ckpt
-----------------------------------------------------------------------------------------
|                  PREDICTED                  |                  ACTUAL                  |
-----------------------------------------------------------------------------------------
| 38              Keep right                  | 15              No vehicles              |
| 11  Right-of-way at the next intersection   | 11  Right-of-way at the next intersection |
| 25              Road work                   | 1            Speed limit (30km/h)        |
| 14                 Stop                     | 14                 Stop                     |
| 12             Priority road                | 12             Priority road               |
-----------------------------------------------------------------------------------------
```

**Analyze Performance**

```
INFO:tensorflow:Restoring parameters from ./model.ckpt
Accuracy = 0.600
```

**Output Top 5 Softmax Probabilities For Each Image Found on the Web**

Plotted the histogram of top 5 softmax probabilities for each image:


actual id: 15
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.00123378
0.000613993
0.000375328
actual id: 11
1.0
4.22532e-19
3.80629e-21
5.24009e-22
2.45139e-24
actual id: 1
0.00298817
0.00243576
0.000783798
actual id: 14
1.4827e-05
2.48153e-06
1.44661e-06
8.74972e-07
actual id: 12
1.0
1.22565e-10
7.9887e-11
1.57222e-11
1.20224e-12