

# Traffic sign Recognition project

## Overview:

In this project I have developed a deep convolutional neural network to train a model so that it can decode signs from natural images by using [German Traffic Sign Dataset](#). Model is also tested on the new traffic sign images downloaded from web.

The steps followed in this project are :

- 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 summary and exploration:

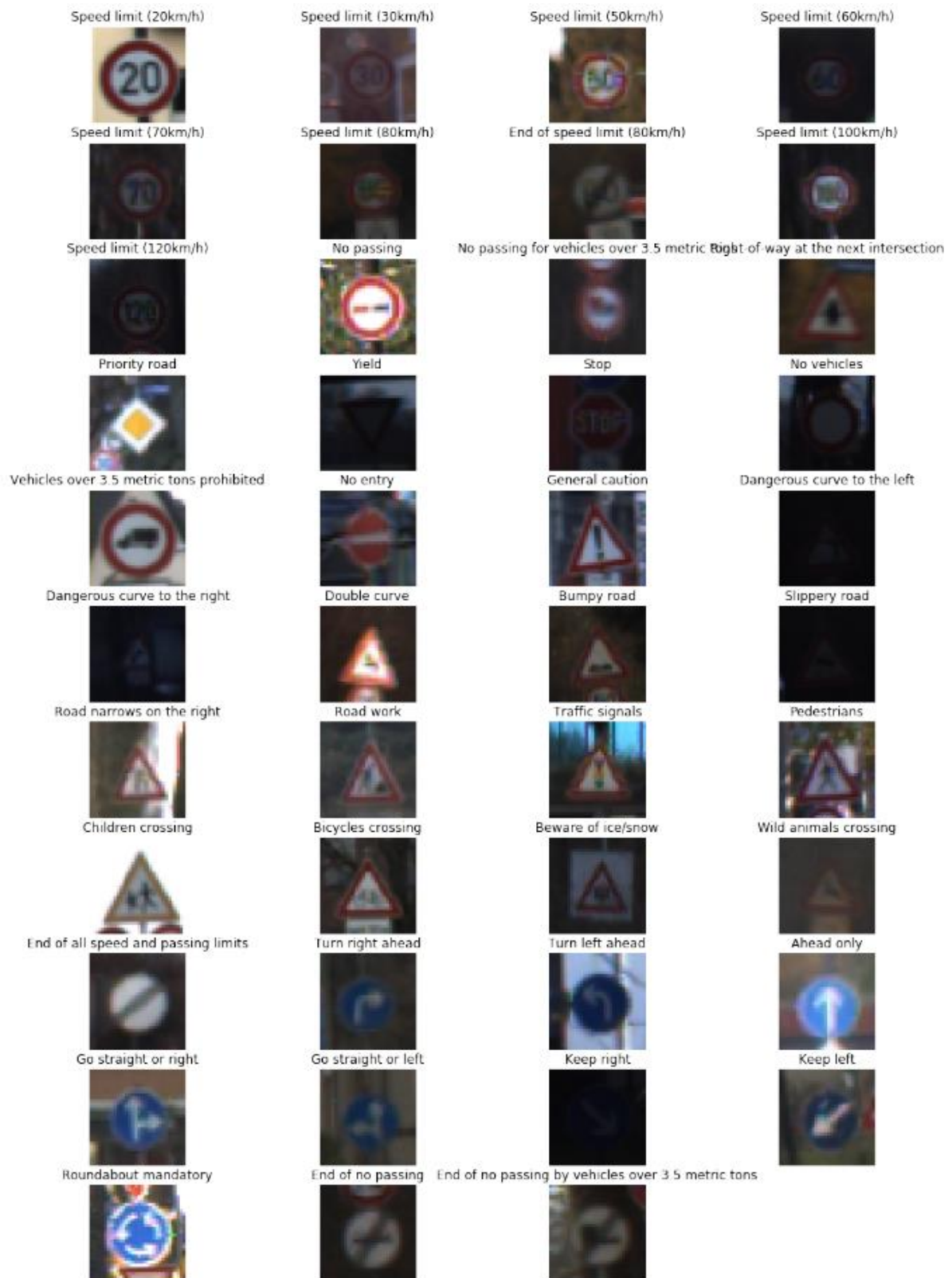
1. Provide a basic summary of the data set using python, numpy and/or pandas

I used the numpy library to calculate summary statistics of the traffic signs data set and result is shown below:

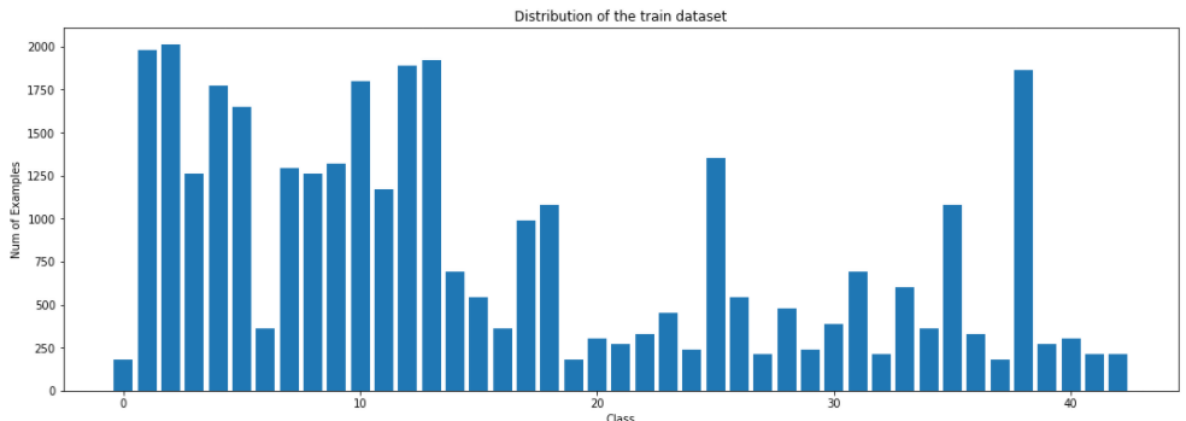
```
Number of training examples = 34799
Number of Validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

2. Include an exploratory visualization of the dataset

Below are the distinct images of German traffic signs visualized.



Distribution of training dataset per class is shown below:



## Design and Test a model architecture:

### 1. Preprocess the data set

Training data is shuffled for better results and the image data is normalized so that the data has mean zero and equal variance. Test and validation image are also normalized and below code is used for image normalization.

```
X_train, y_train = shuffle(X_train, y_train)

def pre_process_image(image):

    image_gray = np.mean(image, axis=3)

    image_gray = np.expand_dims(image_gray, axis=3)

    image_norm = (image_gray - image_gray.mean())/image_gray.std()

    return image_norm

X_train = pre_process_image(X_train)

X_valid = pre_process_image(X_valid)

X_test = pre_process_image(X_test)
```

### 2. Model architecture:

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 Grayscale image
Convolution_1 5x5	1x1 stride, VALID padding, outputs 28X28X6
RELU	
Dropout	Keep probability = 0.7
Max pooling	2x2 stride, outputs 14x14x6
Convolution_2 5x5	1x1 stride, VALID padding, outputs 10x10x16
RELU	
Max pooling	2x2 stride, outputs 5x5x16
Fully connected_0	Output = 400.
Dropout	Keep probability – 0.6
Fully connected_1	Output = 120.
RELU	
Fully connected_2	Output = 84.
RELU	
Dropout	Keep probability = 0.6
Fully connected_3	Output = 43.

To train the model, Initially I used same Lenet Modle as given in class tutorial, but validation accuracy of model was very low so To optimize it I have tried various approach to increase accuracy like adding dropout at fully connected layer, also changing keep probability value, addition of dropout at convolution 1 layer with different keep probability also ADAM optimizer gives better accuracy than SGD optimizer.

To achieve better validation accuracy, I worked out on various hyperparameter like learning rate, epoch's, dropout keep probability and its position in network.

**Following parameter value gives best result for my network:**

- Learning Rate = 0.0009
- Epoch = 70
- batch Size = 100

- Dropout at FC 0 layer = 0.6
- Dropout at FC 2 layer = 0.6
- Dropout at Conv 1 layer = 0.7

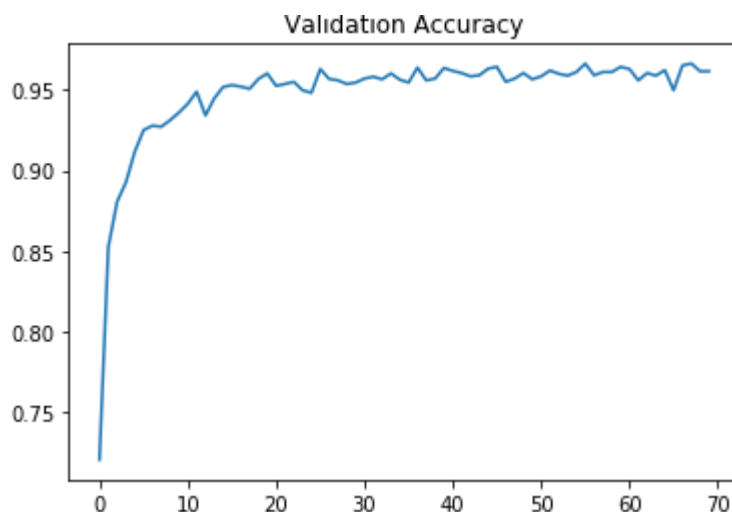
#### Learning from various experiments:

- Learning rate between 0.001 and 0.0009 gives better accuracy.
- Epoch value not fixed, it should be optimized based on model and various experiments.
- Batch Size - Higher batch size train model faster but need higher computation memory.  
Lower batch size takes more time to train model but works on lower computation power.

#### My final model results are:

Train Accuracy = 0.999  
Valid Accuracy = 0.962  
Test Accuracy = 0.954

Below graph shows validation accuracy over number of epochs:




#### Test model on new images:

Below images are taken from web to test our model.

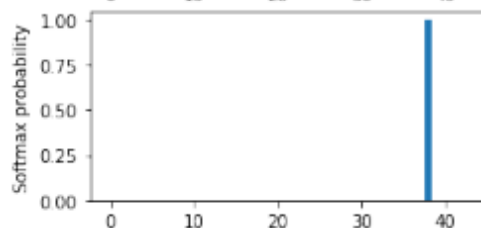
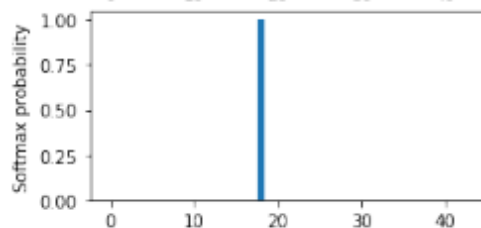
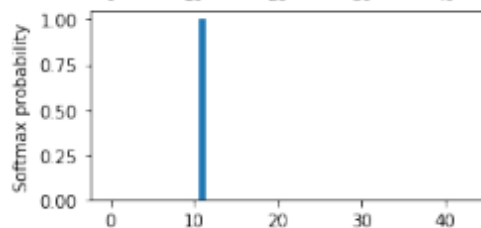
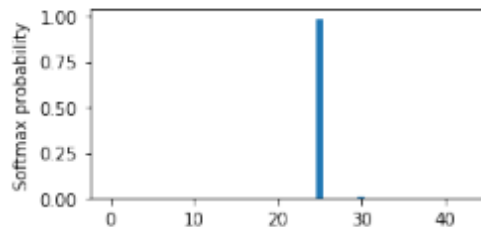
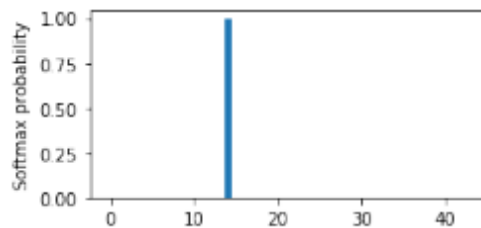
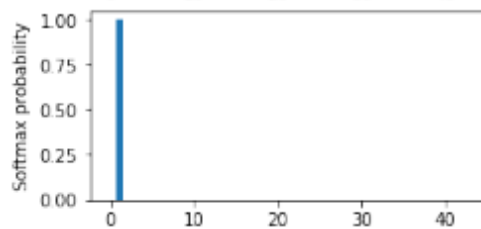
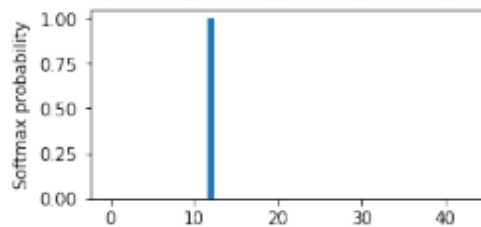
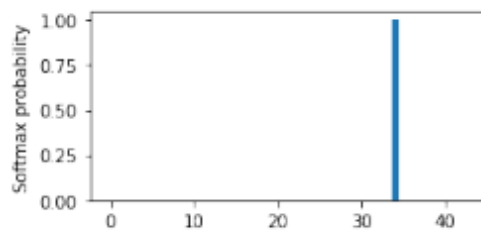
(8, 32, 32, 1)



These images are preprocessed before the model making predictions and below is result of the model's prediction:

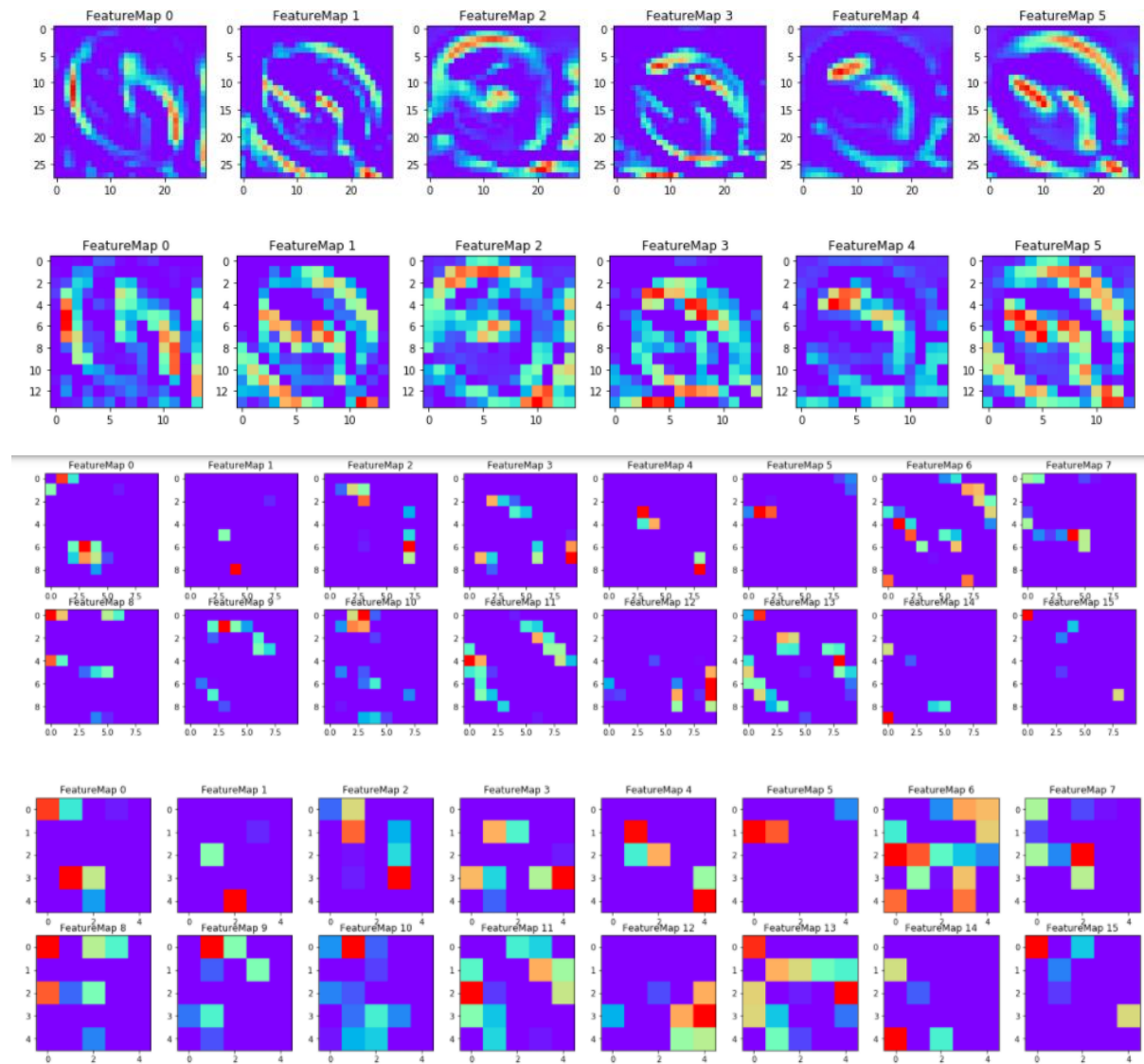
input	top guess: 34 (100%)	2nd guess: 38 (0%)	
			
input	top guess: 12 (100%)	2nd guess: 40 (0%)	
			
input	top guess: 1 (100%)	2nd guess: 2 (0%)	
			
input	top guess: 14 (100%)	2nd guess: 3 (0%)	
			
input	top guess: 25 (99%)	2nd guess: 30 (1%)	
			
input	top guess: 11 (100%)	2nd guess: 30 (0%)	
			
input	top guess: 18 (100%)	2nd guess: 26 (0%)	
			
input	top guess: 38 (100%)	2nd guess: 12 (0%)	
			

The top five softmax probabilities for making predictions on the German traffic sign images found on the web are depicted below:



## Visualize the Neural Network's State with Test Images

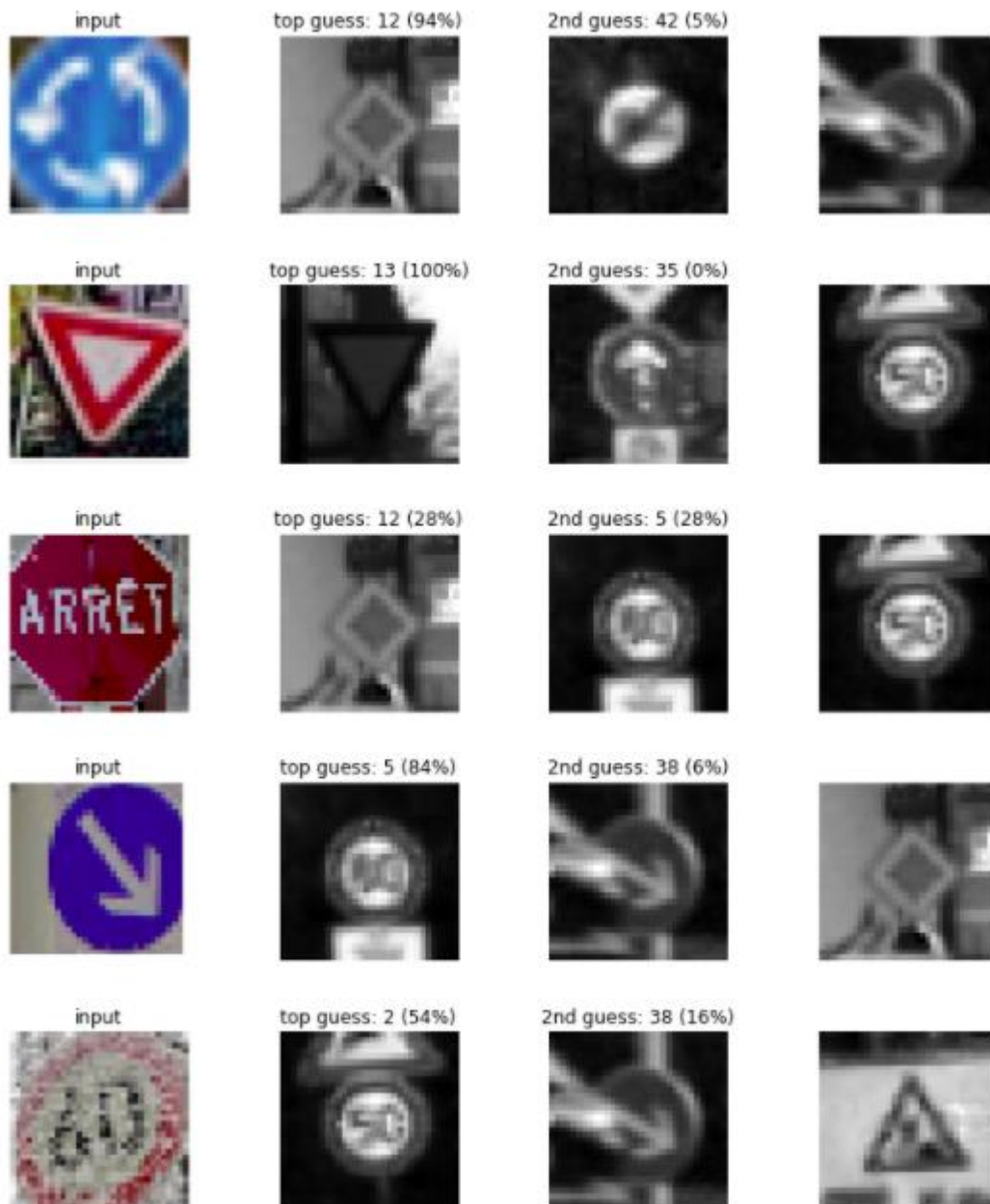
Featuremaps of convolutional layers of the model are shown below for Turn left ahead traffic sign class.



### Discussion :

The model is tested against the new images suggested by reviewer and the models prediction of each image is shown below:





1. “ARRET” : This is a stop sign in French and this image is not in the training sample. So, our model tried classifying it based on shape but that’s not a correct prediction.

Stop



2. Speed limit (60 km/h): This sign is completely covered by snow and our model is able to classify it as 50 km/h but that’s not accurate.

Speed limit (60km/h)



We can train our model with text images of different languages with a particular label to better classify it and also we can apply more augmentation techniques to improve our model like below:

1. Rescaling outward or inward
2. Randomly cropping
3. Zooming
4. Rotating images at some degree
5. Image whitening etc