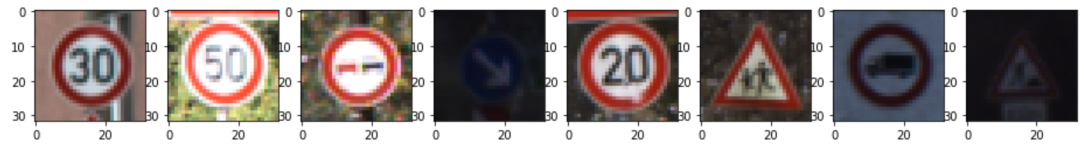
Project German Traffic Sign Recognition

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

Load the data set

Here is a list of randomly picked images loaded for training.

```
Image 3148, ClassId:1
Image 32725, ClassId:2
Image 11052, ClassId:9
Image 15365, ClassId:38
Image 10104, ClassId:0
Image 27295, ClassId:28
Image 5066, ClassId:16
Image 34507, ClassId:25
```



[image1]: Speed limit (30km/h)
[image2]: Speed limit (50km/h)
[image3]: No passing
[image4]: Go straight or right
[image5]: Speed limit (20km/h)
[image6]: Children crossing
[image7]: Vehicles over 3.5 metric tons prohibited
[image8]: Road work

I have attached my notebook CarND-Traffic-Sign-Classifier-Project/blob/master/
Traffic_Sign_Classifier7.ipynb as well.

Data Set Summary & Exploration

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

I calculated 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.

Fig. 1 shows an exploratory visualization of the data set.

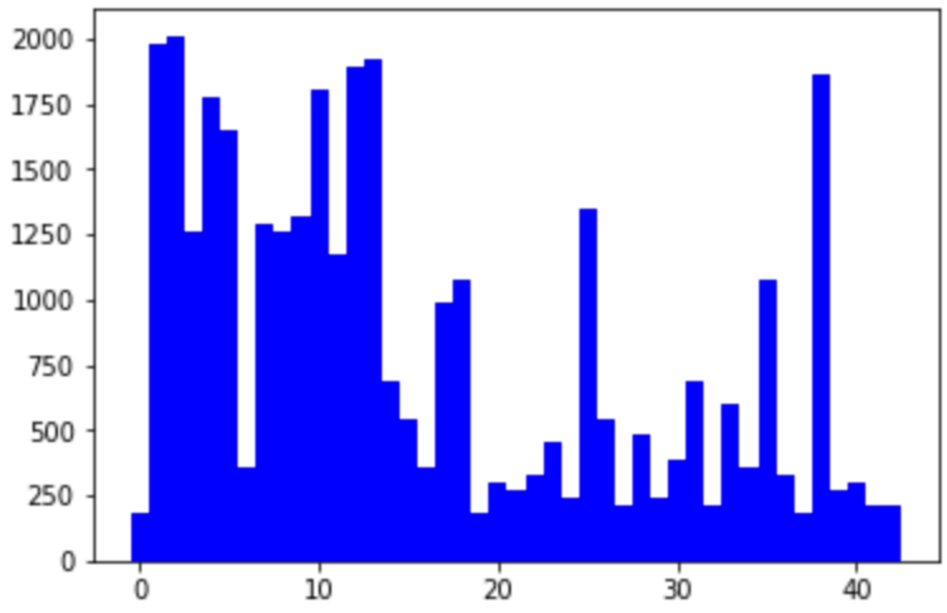


Fig. 1 Distribution of training data in classes.

One problem from the training dataset is unbalanced in classes with mean of 809 and sigma of 619. Another problem is the validation data set. The provided validation data set has only four pairs of images and labels.

We can more training data with balanced set and split some of them as validation data set.

Design and Test a Model Architecture

Pre-process Data
As the first step, I decided to convert the images to grayscale for simplicity of calculation.
I also normalized images between 0.1 and 0.9 to avoid potential boundary problem with 0.0 and 1.0. Fig. 2 and Fig. 3 show traffic sign images and after gray-scaling.

The next objective is to increase training data set with balanced distribution in all 43 classes. To increase training data set.

1. **Make brighter images**

   I used OpenCV API to add brightness with intensity and a random number to images.
   image1 = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
   random_bright = intensity + np.random.uniform()

```
        image1[:,:,2] = image1[:,:,2]*random_bright
        image1 = cv2.cvtColor(image1,cv2.COLOR_HSV2BGR)
```

2. Flip Images

Not every image can be flipped. Table 1 shows images that can be flipped horizontally as the same image (symmetrical left and right). Table 2 shows images that can be flipped vertically (symmetrical top and bottom). Table 3 shows images that can be flipped both horizontally and vertically. Table 4 shows images that can be flipped horizontally to a different sign. flip_extend() is a utility function to handle the flip for images and labels.

3. Using Rotation

Rotation uses Transform from Sci-Kit Image to rotate images. Rotate_img2 is a utility function to perform the task with intensity and a random number.

```
def rotate_img2(x, intensity = 0.75):
        delta = 30. * intensity # scale using augmentation intensity
        d = random.uniform(-delta, delta)
        r_img = rotate(x.copy(), d, mode = 'edge')
        return r_img
```

Fig. 4 shows several rotated images.

4. Using Projection

Project uses Transform from Sci-Kit Image to do ProjectiveTransform images. apply_projection_transform is a utility to apply transform images. Fig. 5 shows projection Images on 20 km/hr signs.

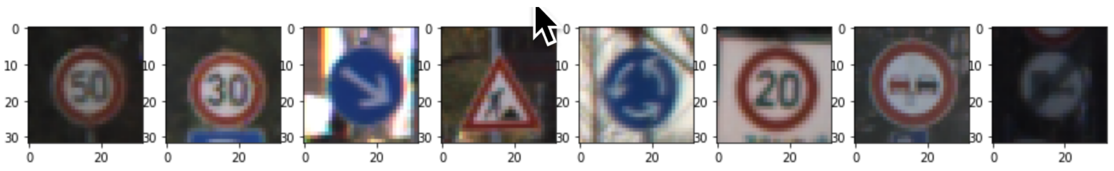


Fig. 2   Training Images

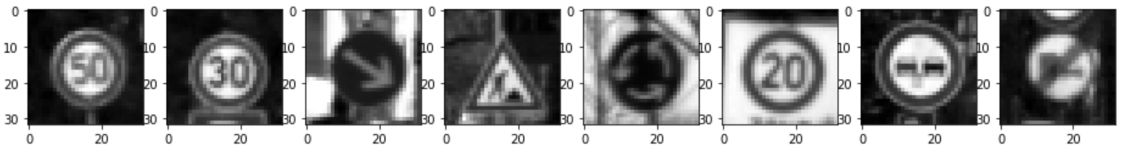


Fig. 3 Training Images Converted to Grayscale

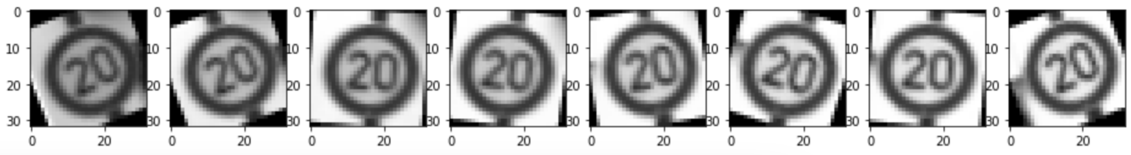


Fig. 4 Rotate Images to Increase Training Data Set

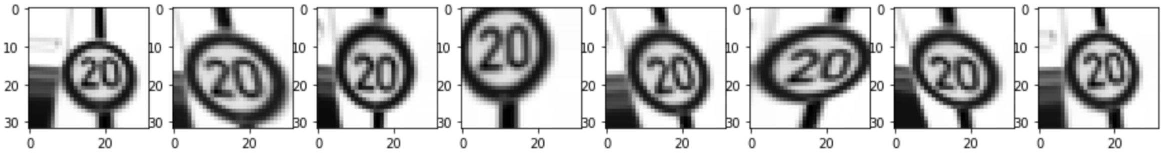
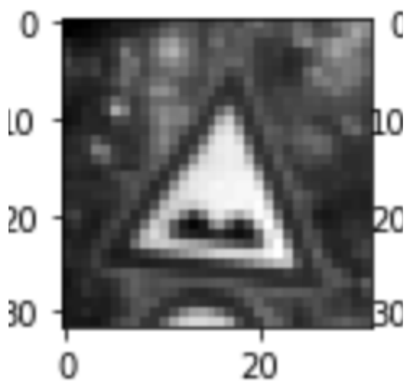
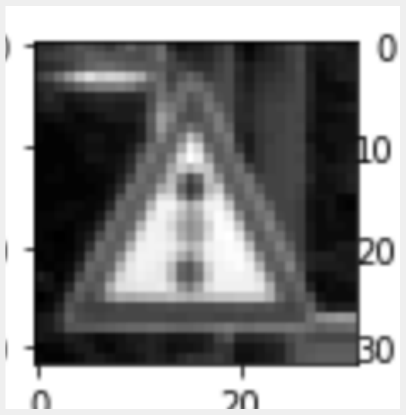
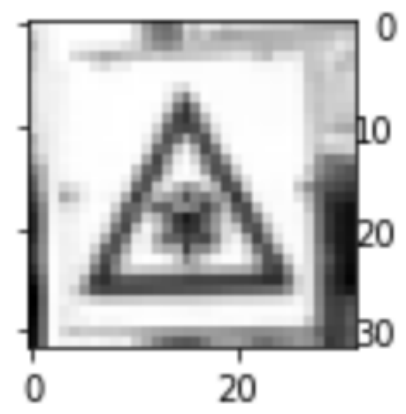


Fig. 5 Project Images to Increase Training Data Set

	11	Right-of-way at the next intersection
	13	Yield
	18	General caution

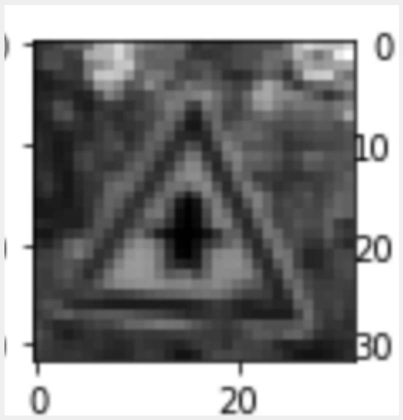
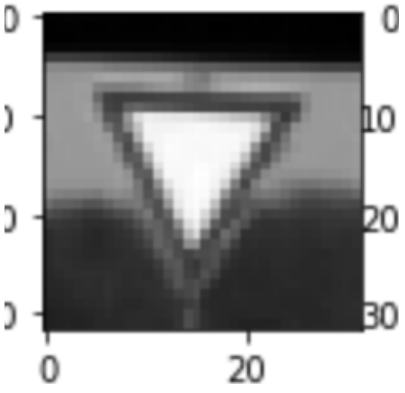
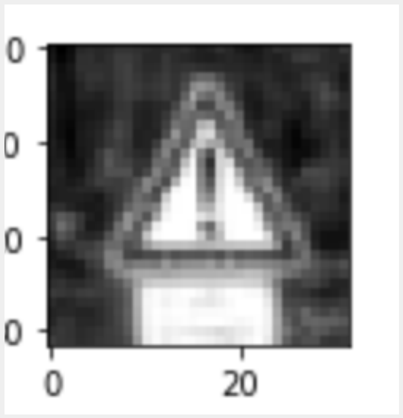
	22	Bumpy road
	26	Traffic signals
	30	Beware of ice/snow

Table 1 Image that can flip horizontally

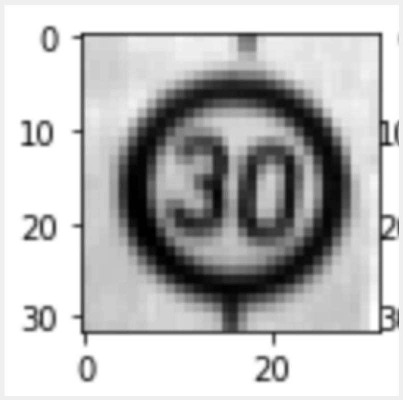
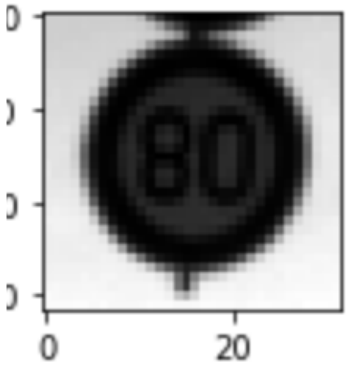
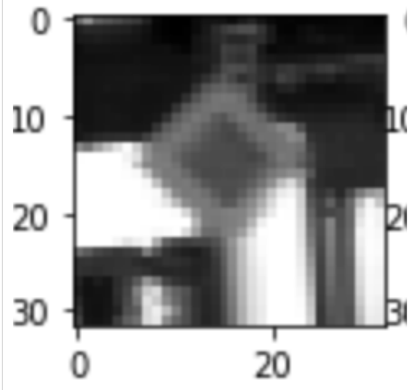
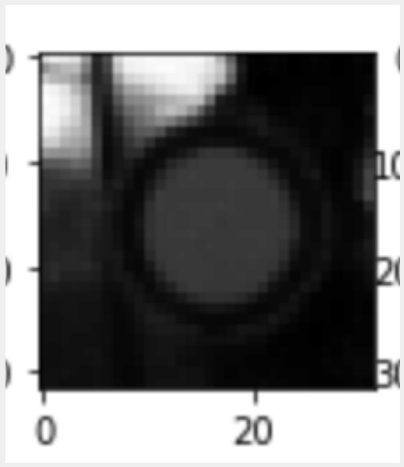
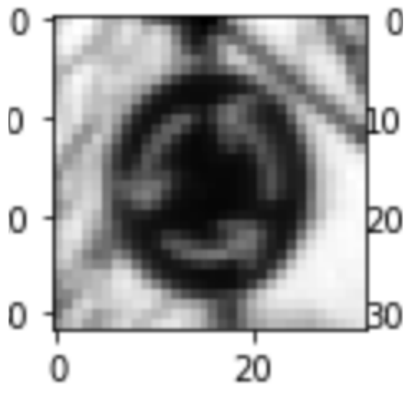
	1	Speed limit (30km/h)
	5	Speed limit (80km/h)

Table 2 Image that can flip vertically

	12	Priority road
	15	No vehicles
	17	No entry

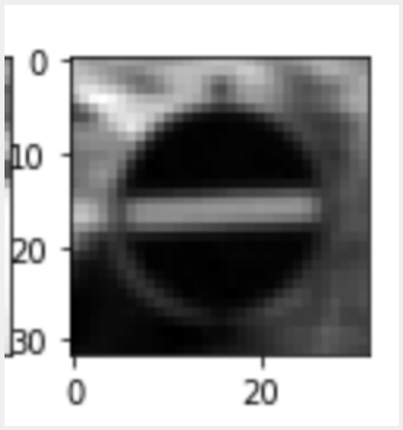
	40	Roundabout mandatory

Table 3 Images can flip both horizontally and vertically

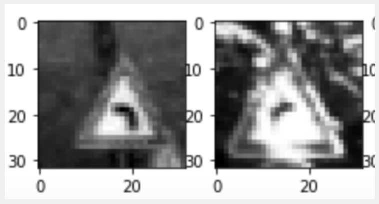
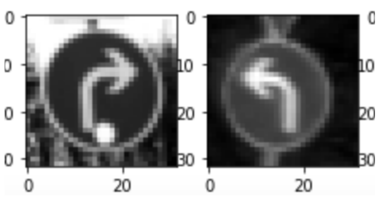
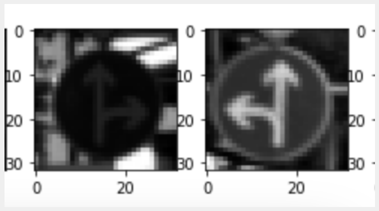
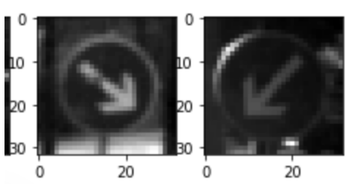
	[19, 20]	19,Dangerous curve to the left 20,Dangerous curve to the right
	[33, 34]	33,Turn right ahead 34,Turn left ahead
	[36, 37]	36,Go straight or right 37,Go straight or left
	[38, 39]	38,Keep right 39,Keep left

Table 4 Images that can flip horizontally to a different sign

I used utility function from sklearn.model_selection to split 10% training data for validation. X_train2,X_valid2,y_train2,y_valid2=train_test_split(X_train2,y_train2,test_size=0.1) by randomly shuffle. By generating more training data helps the model. With technics like rotation, projection, and brightening which cost little in computationn can help generalize model in training if the quality of image is good.

* The size of training set is 383261 (34799).
* The size of validation set is 42585.
* The size of test set is 12630.

The training data set have been increased by 11 times.

The code for my final model is located in the seventh cell of the ipython notebook under LeNet(x) function. Basically, I modified to from the lecture. I changed max_pool to avg_pool, which performed better in my testing. I added dropout to direct connected layers. I also added an additional layer in direct connected layer which helped accuracy in the downloaded images.

My final model consisted of the following layers:

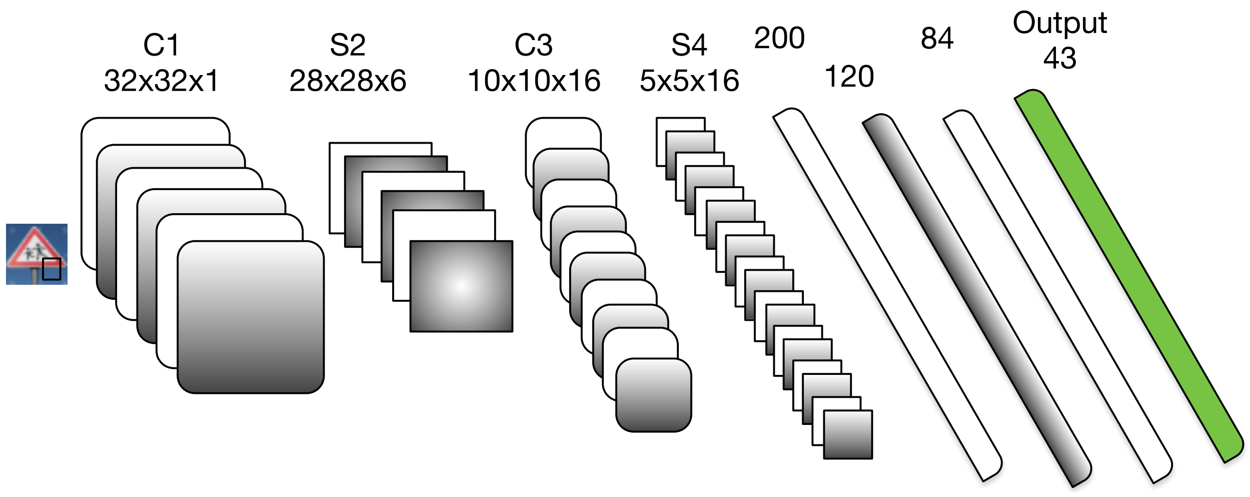


Fig. 6 LeNet Architecture

The code for training the model is located in the eighth cell of the ipython notebook.
Here are the hyper-parameters that I used to train model.
EPOCHS = 100
BATCH_SIZE = 128
mu = 0.0
sigma = 0.1
rate = 0.001

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

My final model results were:
* validation set accuracy of 0.929
* test set accuracy of 0.958

The well known architecture that introduced by the class is a good architecture which has almost 90% accuracy with Epoch 20 and training rate 0.001. I made some mistakes in the process. First, I used the provided validation data set which are only four images. I also used different sizes of images downloaded from web and ran into weird problems. OpenCV uses BGR instead of RGB although it doesn't seem to matter that much in grayscale. Number of training data, balanced data set and quality of training data are important. I used brightening, rotation, flip, and projection to enlarge dataset. I ran into out of memory when I generated lots of training data and cut back. Some images have very low visibility. This is one area that can be improved upon. When I changed the directly connected layers from 400->200->84->43 to 400 -> 120 -> 84 -> 43, it seems to perform better. Later on, I added one more layer to make it 400->200->120->84-43, it improves accuracy with web images. I used dropout to prevent from overfitting. I used L2 regularization (not included) but doesn't seem to help.

However, as the validation set accuracy is only 0.929 while validation set accuracy is 0.958, the model seems to be overfitting. The paper about Traffic Signs Color Detection and Segmentation in Poor Light Conditions is a good reference for future improvement.

Test a Model on New Images

Table 5 shows five German traffic signs that I found on the web.

Image	Id & description	Predication

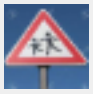
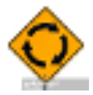
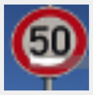
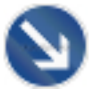
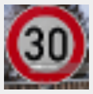
	28,Children crossing	1.0
	40,Roundabout mandatory	0.0005
	2,Speed limit (50km/h)	0.793
	38,Keep right	0.458
	1,Speed limit (30km/h)	0.924

Table 5 Test Images Download from Web

The second image might be difficult to classify because there are several signs with similar pattern. As we use grayscale, the yellow background and black circle sign don't help recognition. I intentionally use a brush to wipe out some watermark word on the image. It's good it's predicted correctly.

The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%. The image of roundabout mandatory is difficult to recognize due to quality of training images. Besides, the background and foreground color can't help due to grayscale is used in recognition.

Fig. 7 shows top five softmax probabilities for the predictions on the German traffic sign images found on the web.

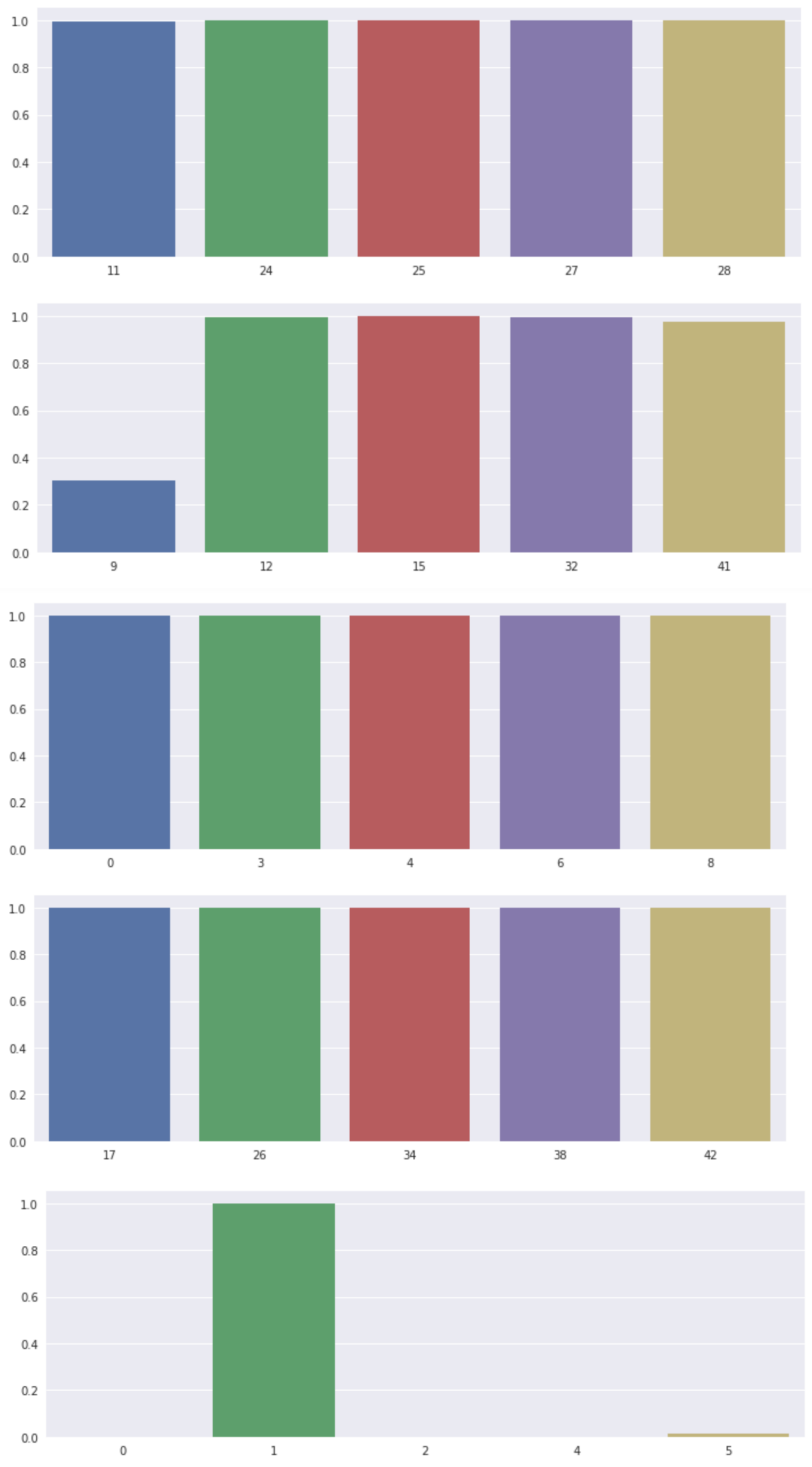


Fig. 7   Top five softmax probabilities for the predictions on Web German traffic sign images

Later on, when we learn some more in vision recognition, I probably can detect these signs better. Improving train data quality and quantity definitely will help as well.

Fig. 8 shows the images in convolution of layer 1 and layer 2. The first two layers are more on edge and line detection.

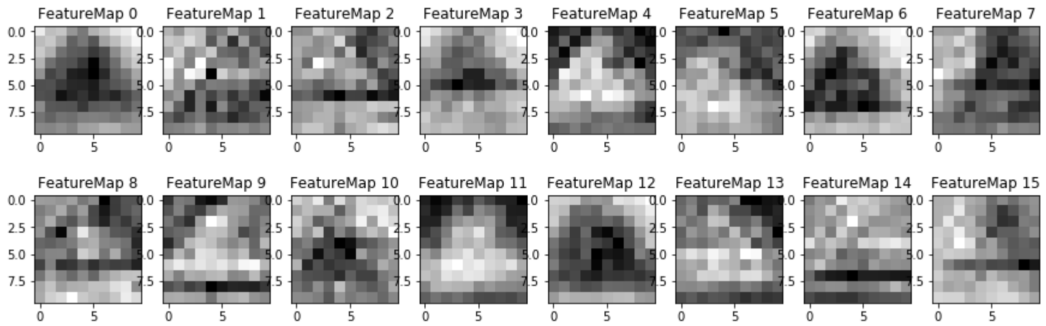


Fig. 8 Images outputFeatureMap for Layer 1 and Layer 2 for Child Crossing Sign