Traffic Sign Recognition

Build a Traffic Sign Recognition Project

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

C:\work\AI\jupyter_notebook\udacity\project\CarND-Traffic-Sign-Classifier-Project-master\examples

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. You can use this template as a guide for writing the report. The submission includes the project code.

You're reading it! and here is a link to my project code

Data Set Summary & Exploration

1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

I used the pandas library to calculate summary statistics of the traffic signs data set:

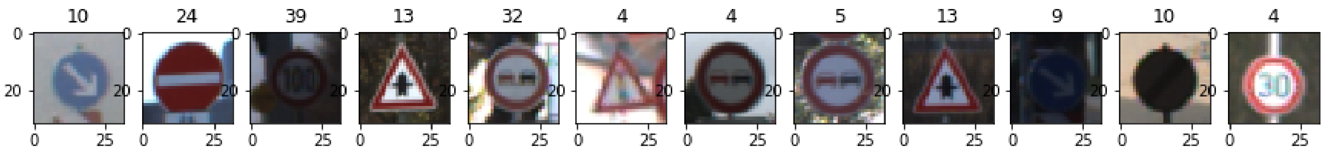
- The size of training set is 34799
- The size of the validation set is 4410
- The size of test set is 12630
- The shape of a traffic sign image is (32, 32, 3)
- The number of unique classes/labels in the data set is 43

2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data ...

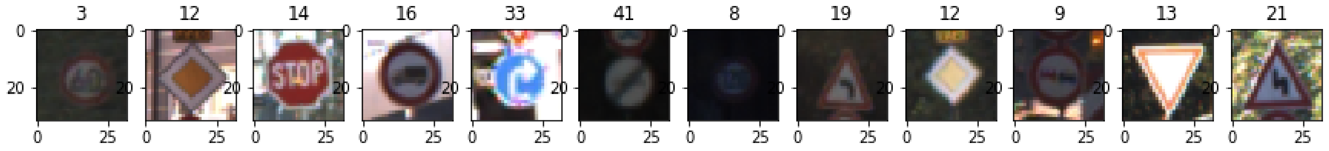
Training Samples

```
Training images: indices  [19980, 12438, 4639, 6736, 22911, 20954, 7568, 5393, 12288, 8869, 33981, 17120]
```



Validation Samples

```
Validation images: indices  [19980, 12438, 4639, 6736, 22911, 20954, 7568, 5393, 12288, 8869, 33981, 17120]
```



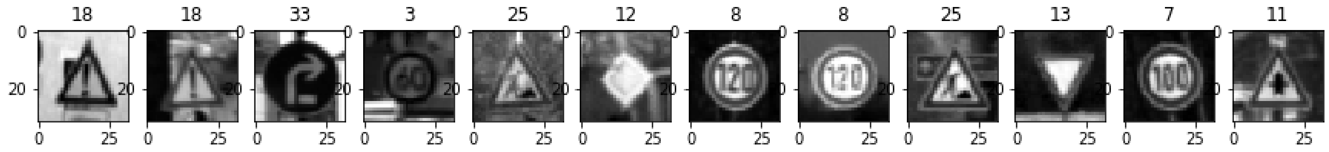
Design and Test a Model Architecture

1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc. (OPTIONAL: As described in the "Stand Out Suggestions" part of the rubric, if you generated additional data for training, describe why you decided to generate additional data, how you generated the data, and provide example images of the additional data. Then describe the characteristics of the augmented training set like number of images in the set, number of images for each class, etc.)

As a first step, I decided to convert the images to grayscale because ...

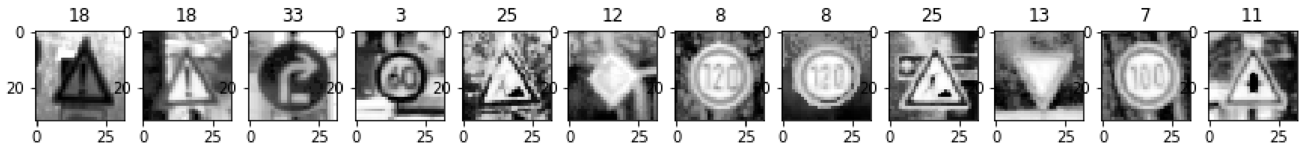
Here is an example of a traffic sign image before and after grayscaling.

```
Training gray images: indices  [29395   241 26586 21972 27379  8653 15524 15692 25444  2162  1414 12236]
```



As a last step, I normalized the image data using both cv2.euqlalizeHist() and (pixel-128)/128 to do normalize, the output samples as below:

```
Training equalizeHist+(pixel - 128)/128 images: indices  [29395   241 26586 21972 27379  8653 15524 15692 25444  2162  1414 12236]
```



I have compared below eight different normalize methods:

Method 1~5 have train set of 34799, validation set of 4410, and test set of 12630.

Method 1. (MeanStd, Traffic*Sign*Classifier*norm*mean-std.ipynb)

```
def normalize(img):
    img = img - np.mean(img)
    img = img / np.std(img)
    return img

X_train_norm = normalize(X_train[:,])
X_valid_norm = normalize(X_valid[:,])
X_test_norm = normalize(X_test[:,])
```

After normalized, the data sets are: Train set shape is (34799, 32, 32, 3), valid set shape is (4410, 32, 32, 3), and test set shape is (12630, 32, 32, 3).

Method 2. (normalize128, Traffic*Sign*Classifier*norm*128.ipynb)

```
def normalize128(image):
    #img = (image - 128)/128  equal img = image/128 - 1
    img = np.divide(image, 128)
    image = np.subtract(img,1)
    return image

X_train_norm = normalize128(X_train[:,])
X_valid_norm = normalize128(X_valid[:,])
X_test_norm = normalize128(X_test[:,])
```

After normalized, the data sets are: Train set shape is (34799, 32, 32, 3), valid set shape is (4410, 32, 32, 3), and test set shape is (12630, 32, 32, 3).

Method 3. (EqualizeHist, Traffic*Sign*Classifier*norm*equalizeHist.ipynb)

```
def to_gray(image):
    return cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# Normalizing function
def normalize_image(image):
    img = np.divide(image, 128)
    image = np.subtract(img,1)
    return image

# Histogram equalization based on open-cv builtin library
def hist_equalize(image):
    return cv2.equalizeHist(image)

X_train_norm = normalize_image(hist_equalize(to_gray(X_train_norm)))
X_valid_norm = normalize_image(hist_equalize(to_gray(X_valid_norm)))
X_test_norm = normalize_image(hist_equalize(to_gray(X_test_norm)))
```

After normalized, the data sets are: Train set shape is (34799, 32, 32, 1), valid set shape is (4410, 32, 32, 1), and test set shape is (12630, 32, 32, 1).

So, the train image changed from 32x32x3 to 32x32x1, and we need to redefine the img_channel from 3 to 1 in train model.

Method 4. (AgumentBrightness, Traffic*Sign*Classifier*norm*mean-std_augment-brightness.ipynb)

Method 4 have doubled the train set (from 34799 to 69598), validation set keep of 4410, and test set keep of 12630.

```
def normalize(img):
    img = img - np.mean(img)
    img = img / np.std(img)
    return img

X_train_norm = normalize(X_train[:,])
X_valid_norm = normalize(X_valid[:,])
X_test_norm = normalize(X_test[:,])

def **augment_brightness_camera_images**(image):
    image1 = cv2.cvtColor(image,cv2.COLOR_RGB2HSV)
    random_bright = .25+np.random.uniform()
    image1[:,:,2] = image1[:,:,2]*random_bright
    image1 = cv2.cvtColor(image1,cv2.COLOR_HSV2RGB)
    return image1

for i in range(len(X_train_gen)):
    X_train_gen[i] = augment_brightness_camera_images(X_train_gen[i])
X_train_gen_norm = normalize(X_train_gen[:,])

X_train_norm = np.concatenate((X_train_norm, X_train_gen_norm), axis=0)
y_train = np.concatenate((y_train, y_train_gen), axis=0)
```

After normalized, the data sets are: Train set shape is (69598, 32, 32, 3), valid set shape is (4410, 32, 32, 3), and test set shape is (12630, 32, 32, 3).

Method 5. (normalize128 + AugmentBrightness, Traffic*Sign*Classifier*norm*128_augment-brightness.ipynb)

```
def normalize128(image):
    #img = (image - 128)/128  equal img = image/128 - 1
    img = np.divide(image, 128)
    image = np.subtract(img,1)
    return image

X_train_norm = normalize(X_train[:,])
X_valid_norm = normalize(X_valid[:,])
X_test_norm = normalize(X_test[:,])

def **augment_brightness_camera_images**(image):
    image1 = cv2.cvtColor(image,cv2.COLOR_RGB2HSV)
    random_bright = .25+np.random.uniform()
    image1[:,:,2] = image1[:,:,2]*random_bright
    image1 = cv2.cvtColor(image1,cv2.COLOR_HSV2RGB)
    return image1

for i in range(len(X_train_gen)):
    X_train_gen[i] = augment_brightness_camera_images(X_train_gen[i])
X_train_gen_norm = normalize(X_train_gen[:,])

X_train_norm = np.concatenate((X_train_norm, X_train_gen_norm), axis=0)
y_train = np.concatenate((y_train, y_train_gen), axis=0)
```

After normalized, the data sets are: Train set shape is (69598, 32, 32, 3), valid set shape is (4410, 32, 32, 3), and test set shape is (12630, 32, 32, 3).

Split train set to train set and validation set

In Method 6-8, The valid set will be split from train set. So before normalization, the train set is 27839, validation set is 6960, and test set is 12630.

Method 6. (TrainSplit + mean-std, Traffic*Sign*Classifier*train-split*mean-std.ipynb)

```
def normalize(img):
    img = img - np.mean(img)
    img = img / np.std(img)
    return img

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=0)

X_train_norm = normalize(X_train[:,])
X_valid_norm = normalize(X_valid[:,])
X_test_norm = normalize(X_test[:,])
```

After normalized, the data sets are: Train set shape is (27839, 32, 32, 3), valid set shape is (6960, 32, 32, 3), and test set shape is (12630, 32, 32, 3).

Method 7. (TrainSplit + equalizeHist, Traffic*Sign*Classifier*train-split*equalizeHist.ipynb)

```
def to_gray(image):
    return cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# Normalizing function
def normalize_image(image):
    img = np.divide(image, 128)
    image = np.subtract(img,1)
    return image

# Histogram equalization based on open-cv builtin library
def hist_equalize(image):
    return cv2.equalizeHist(image)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=0)
```

```
X_train_norm = normalize_image(hist_equalize(to_gray(X_train_norm)))
X_valid_norm = normalize_image(hist_equalize(to_gray(X_valid_norm)))
X_test_norm = normalize_image(hist_equalize(to_gray(X_test_norm)))
```

After normalized, the data sets are: Train set shape is (27839, 32, 32, 1), valid set shape is (6960, 32, 32, 1), and test set shape is (12630, 32, 32, 1).

Method 8. (TrainSplit + AgumentBrightness, Traffic*Sign*Classifier*train-split*augment-brightness.ipynb)

Method 8 have doubled the train set (from 34799 to 69598), validation set keep of 4410, and test set keep of 12630.

```
def normalize(img):
    img = img - np.mean(img)
    img = img / np.std(img)
    return img

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=0)

X_train_norm = normalize(X_train[:,])
X_valid_norm = normalize(X_valid[:,])
X_test_norm = normalize(X_test[:,])

def **augment_brightness_camera_images**(image):
    image1 = cv2.cvtColor(image,cv2.COLOR_RGB2HSV)
    random_bright = .25+np.random.uniform()
    image1[:,:,2] = image1[:,:,2]*random_bright
    image1 = cv2.cvtColor(image1,cv2.COLOR_HSV2RGB)
    return image1

for i in range(len(X_train_gen)):
    X_train_gen[i] = augment_brightness_camera_images(X_train_gen[i])
X_train_gen_norm = normalize(X_train_gen[:,])

X_train_norm = np.concatenate((X_train_norm, X_train_gen_norm), axis=0)
y_train = np.concatenate((y_train, y_train_gen), axis=0)
```

After normalized, the data sets are: Train set shape is (55678, 32, 32, 3), valid set shape is (6960, 32, 32, 3), and test set shape is (12630, 32, 32, 3).

2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.

My final model consisted of the following layers:

(Note: img_channel in method 3 and 7 are 1, and others are 3.)

```
| Layer                 |     Description                                    |
|:---------------------:|:--------------------------------------------:      |
| Input                 | 32x32ximg_channel Grayscale image                  |
| Convolution 5x5       | 1x1 stride, VALID padding, outputs 28x28x6         |
| RELU                  |       Activation function                          |
| Max pooling           | 2x2 stride (ksize of 2), VALID padding,  outputs 14x14x6  |
| Convolution 5x5       | 1x1 stride, VALID padding, outputs 10x10x16        |
| RELU                  |       Activation function                          |
| Max pooling           | 2x2 stride (ksize of 2), VALID padding,  outputs 5x5x16   |
| Fully connected       | Input 400, Output 120                              |
| RELU                  |       Activation function                          |
| Fully connected       | Input 120, Output 84                               |
| RELU                  |       Activation function                          |
| Fully connected       | Input 84, Output 43                                |
```

3. Describe how you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

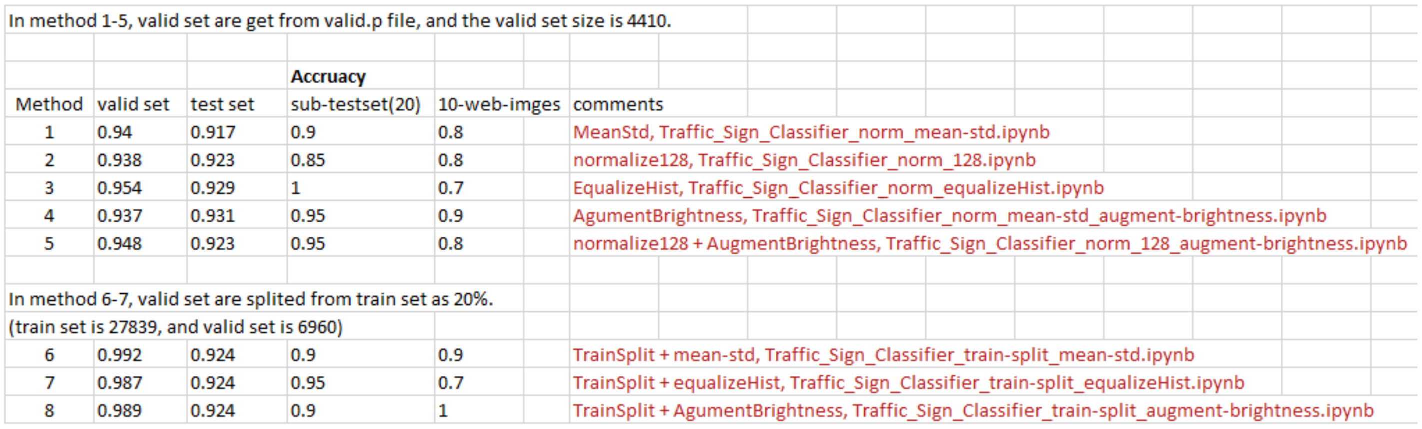
The following are the hyperparameters used: - Batch size: 128 - Epoch : 50 - Learning rate: 0.001 - Standard deviation (sigma): 0.1 - Mean: 0.0

The training procedure is a kind of standard. First, the dataset is divided into batches, then the batches are fed to the algorithm that tries to reduce the mean entropy and optimize it using the Adam optimizer. In the training process, various batch sizes, learning rate and epochs are tested.

I have also try batch size of 64, found a little performance improvement, but the test accuracy on 10-web-images are dropped from 90% to 60%. (see test result in Traffic*Sign*Classifier*norm*128_batch-64.ipynb)

4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think the architecture is suitable for the current problem.

My final model results , see below tables comparison:



In method 1-5, valid set are get from valid.p file, and the valid set size is 4410.

			Accruacy		
Method	valid set	test set	sub-testset(20)	10-web-imges	comments
1	0.94	0.917	0.9	0.8	MeanStd, Traffic_Sign_Classifier_norm_mean-std.ipynb
2	0.938	0.923	0.85	0.8	normalize128, Traffic_Sign_Classifier_norm_128.ipynb
3	0.954	0.929	1	0.7	EqualizeHist, Traffic_Sign_Classifier_norm_equalizeHist.ipynb
4	0.937	0.931	0.95	0.9	AgumentBrightness, Traffic_Sign_Classifier_norm_mean-std_augment-brightness.ipynb
5	0.948	0.923	0.95	0.8	normalize128 + AugmentBrightness, Traffic_Sign_Classifier_norm_128_augment-brightness.ipynb

In method 6-7, valid set are splited from train set as 20%.
(train set is 27839, and valid set is 6960)

Method	valid set	test set	sub-testset(20)	10-web-imges	comments
6	0.992	0.924	0.9	0.9	TrainSplit + mean-std, Traffic_Sign_Classifier_train-split_mean-std.ipynb
7	0.987	0.924	0.95	0.7	TrainSplit + equalizeHist, Traffic_Sign_Classifier_train-split_equalizeHist.ipynb
8	0.989	0.924	0.9	1	TrainSplit + AgumentBrightness, Traffic_Sign_Classifier_train-split_augment-brightness.ipynb

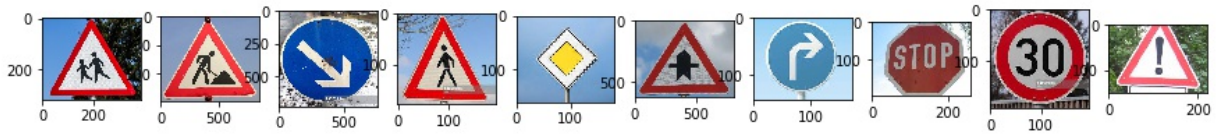
It seems method 6-8 has much higher accuracy on validation set(Method 8 seems has the best performance for 10-web- images). I have no idea about why this happen, it seems small train set will get more higher accuracy?

Test a Model on New Images

1. Choose 10 German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.

Here are five 10 traffic signs that I found on the web:

```
images_filenames.len= 11 ['.ipynb_checkpoints', 'children.jpg', 'construction.jpg', 'keep right.jpg', 'pedestrian.jpg', 'prioritty.jpg', 'ri
ght of way.jpg', 'right.jpg', 'stop.jpg', 'thirty_miles.jpg', 'warning.jpg']
Number of new images:  10
```



The first image might be difficult to classify because ...

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).

Here are one of the results of the prediction:

```
| Image                  |     Prediction                            |
|:---------------------:|:---------------------------------:         |
| Children crossing     |  Children crossing                        |
| Road work             | Road work                                 |
| Keep right            | Keep right                                |
| Pedestrians           | **General caution**                       |
| Priority Road         | Priority road                             |
| Right-of-way at next intersection | Right-of-way at next intersection |
| Turn right ahead      | Turn right ahead                          |
| Stop                  | Stop                                      |
| Speed limit (30km/h)  | Speed limit (30km/h)                      |
| General caution       | **Traffic signals**                       |
```

The model was able to correctly guess 8 of the 10 traffic signs, which gives an accuracy of 80%. This compares favorably to the accuracy on the test set of ~92%.

3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)

The code for making predictions on my final model is located in the 11th cell of the Ipython notebook.

E.g, for true image of speed sign(1- Speed limit (30km/h)) , the top five soft max probabilities were

```
| Probability           |     Prediction                            |
|:---------------------:|:-----------------------------------------:|
| 60.4%                 | 1- Speed limit (30km/h)                   |
| 37.34%                | 3- Speed limit (60km/h)                   |
| 0.17%                 | 29- Bicycles crossing                     |
| 0.08%                 | 0- Speed limit (20km/h)                   |
| 0.01%                 | 13- Yield                                 |
```

Below are the output from python:

```
1 True sign: label 28 - Children crossing
  Predict. sign 1: 100.00 %  28- Children crossing
  Predict. sign 2: 0.00 %  29- Bicycles crossing
  Predict. sign 3: 0.00 %  24- Road narrows on the right
  Predict. sign 4: 0.00 %  23- Slippery road
  Predict. sign 5: 0.00 %  27- Pedestrians
-------------------------------------
2 True sign: label 25 - Road work
  Predict. sign 1: 100.00 %  25- Road work
  Predict. sign 2: 0.00 %  20- Dangerous curve to the right
  Predict. sign 3: 0.00 %  26- Traffic signals
  Predict. sign 4: 0.00 %  18- General caution
  Predict. sign 5: 0.00 %  19- Dangerous curve to the left
-------------------------------------
3 True sign: label 38 - Keep right
  Predict. sign 1: 100.00 %  38- Keep right
  Predict. sign 2: 0.00 %  32- End of all speed and passing limits
  Predict. sign 3: 0.00 %  40- Roundabout mandatory
  Predict. sign 4: 0.00 %  36- Go straight or right
  Predict. sign 5: 0.00 %  6- End of speed limit (80km/h)
-------------------------------------
4 True sign: label 27 - Pedestrians
  Predict. sign 1: 99.93 %  18- General caution
  Predict. sign 2: 0.07 %  27- Pedestrians
  Predict. sign 3: 0.00 %  11- Right-of-way at the next intersection
  Predict. sign 4: 0.00 %  26- Traffic signals
  Predict. sign 5: 0.00 %  21- Double curve
   -------------------------------
  INCORRECT classification
-------------------------------------
5 True sign: label 12 - Priority road
  Predict. sign 1: 100.00 %  12- Priority road
  Predict. sign 2: 0.00 %  26- Traffic signals
  Predict. sign 3: 0.00 %  32- End of all speed and passing limits
  Predict. sign 4: 0.00 %  38- Keep right
  Predict. sign 5: 0.00 %  11- Right-of-way at the next intersection
-------------------------------------
6 True sign: label 11 - Right-of-way at the next intersection
  Predict. sign 1: 100.00 %  11- Right-of-way at the next intersection
  Predict. sign 2: 0.00 %  27- Pedestrians
  Predict. sign 3: 0.00 %  30- Beware of ice/snow
  Predict. sign 4: 0.00 %  21- Double curve
  Predict. sign 5: 0.00 %  23- Slippery road
-------------------------------------
7 True sign: label 33 - Turn right ahead
  Predict. sign 1: 100.00 %  33- Turn right ahead
  Predict. sign 2: 0.00 %  35- Ahead only
  Predict. sign 3: 0.00 %  39- Keep left
  Predict. sign 4: 0.00 %  40- Roundabout mandatory
  Predict. sign 5: 0.00 %  13- Yield
-------------------------------------
8 True sign: label 14 - Stop
  Predict. sign 1: 99.99 %  14- Stop
  Predict. sign 2: 0.01 %  25- Road work
  Predict. sign 3: 0.00 %  13- Yield
  Predict. sign 4: 0.00 %  2- Speed limit (50km/h)
  Predict. sign 5: 0.00 %  15- No vehicles
-------------------------------------
9 True sign: label 1 - Speed limit (30km/h)
  Predict. sign 1: 62.40 %  1- Speed limit (30km/h)
  Predict. sign 2: 37.34 %  3- Speed limit (60km/h)
  Predict. sign 3: 0.17 %  29- Bicycles crossing
  Predict. sign 4: 0.08 %  0- Speed limit (20km/h)
  Predict. sign 5: 0.01 %  13- Yield
-------------------------------------
10 True sign: label 18 - General caution
  Predict. sign 1: 96.32 %  26- Traffic signals
  Predict. sign 2: 2.21 %  18- General caution
  Predict. sign 3: 0.96 %  22- Bumpy road
  Predict. sign 4: 0.31 %  29- Bicycles crossing
  Predict. sign 5: 0.16 %  24- Road narrows on the right
   --------------------------------
  INCORRECT classification
-------------------------------------
```

(Optional) Visualizing the Neural Network (See Step 4 of the Ipython notebook for more details)

1. Discuss the visual output of your trained network's feature maps. What characteristics did the neural network use to make classifications?