

Project Code: <https://github.com/anilrajgr/CarND-Traffic-Sign-Classifier-Project>

Look for [Traffic\\_Sign\\_Classifier.ipynb](#) in that repository.

## Data Set Summary & Exploration

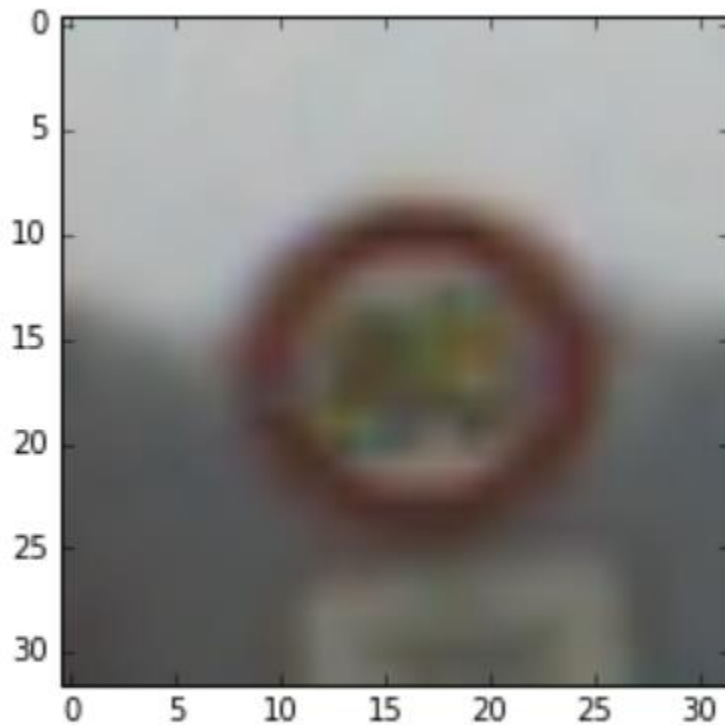
Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

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 32x32
- The number of unique classes/labels in the data set is 43

Include an exploratory visualization of the dataset.

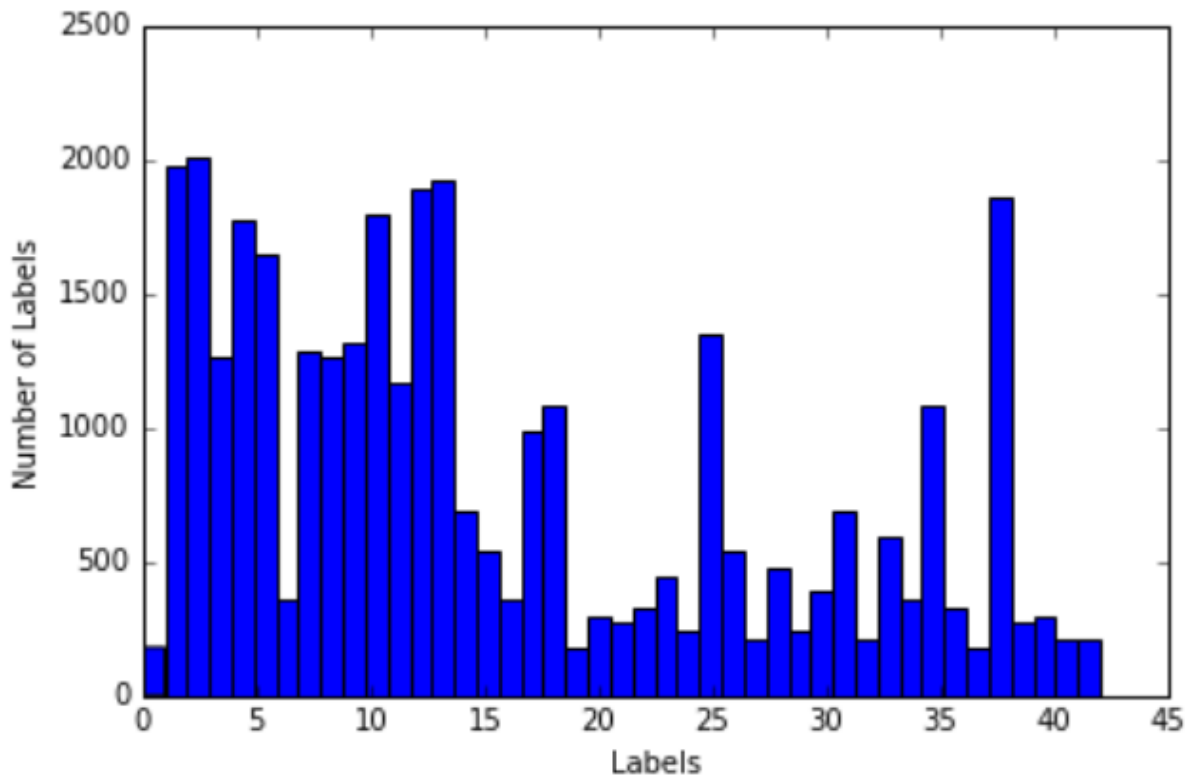
The following figure shows an example image.



5

The number "5" in the bottom is the classification. 5 stands for speed limit 80km/h.

The histogram of all the 43 labels is like the following:



The labels are not very equally distributed. Some are more common than others.

## Design and Test a Model Architecture

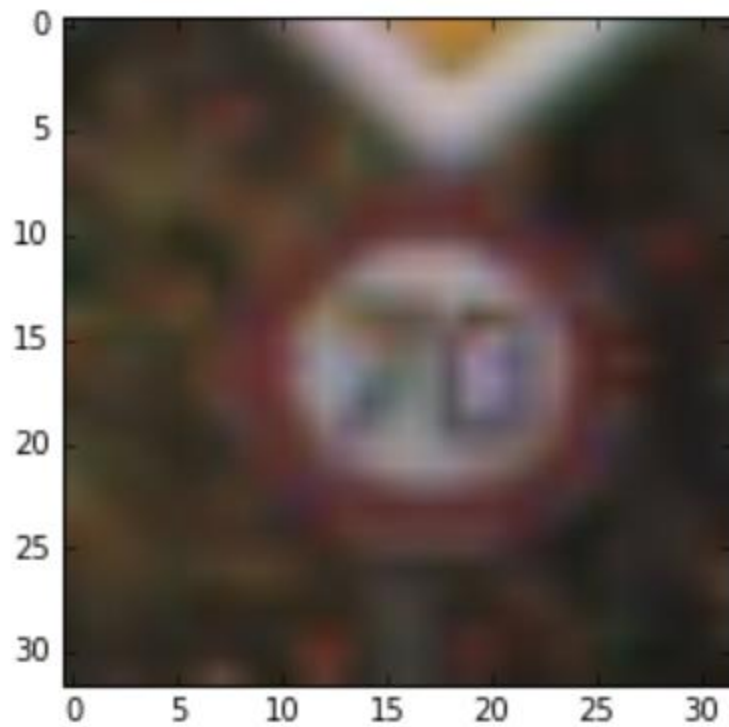
Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc.

The information in the traffic signs are not color based. Meaning, there are no two signs which differ only by the color. In traffic light, there could be a circular red and green light which are same in shape but differ in color. But here, we don't have that situation. Hence, the color information can be safely removed.

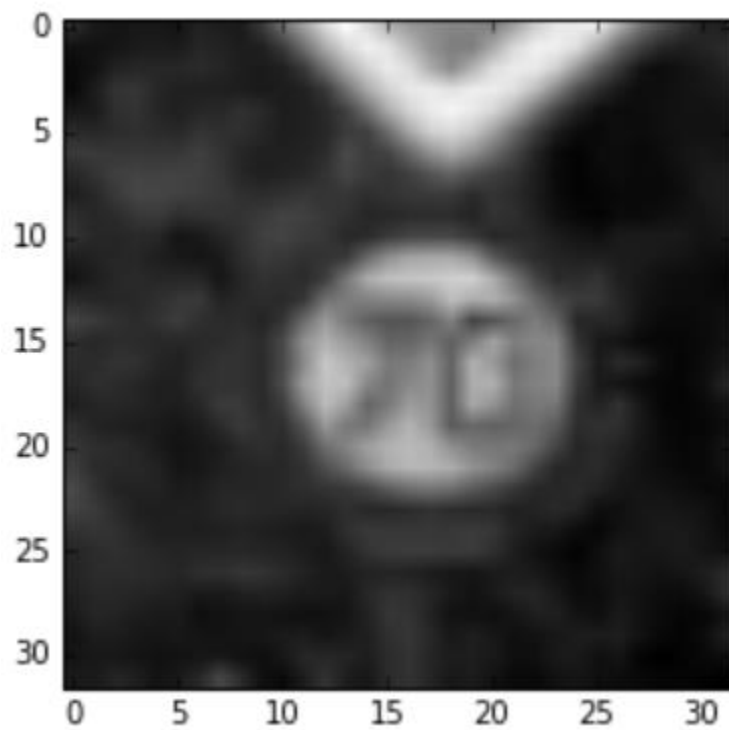
Removing color serves two purpose.

1. Obviously, we remove unnecessary noise in our data (color information)
2. We have only one third of the original data size.

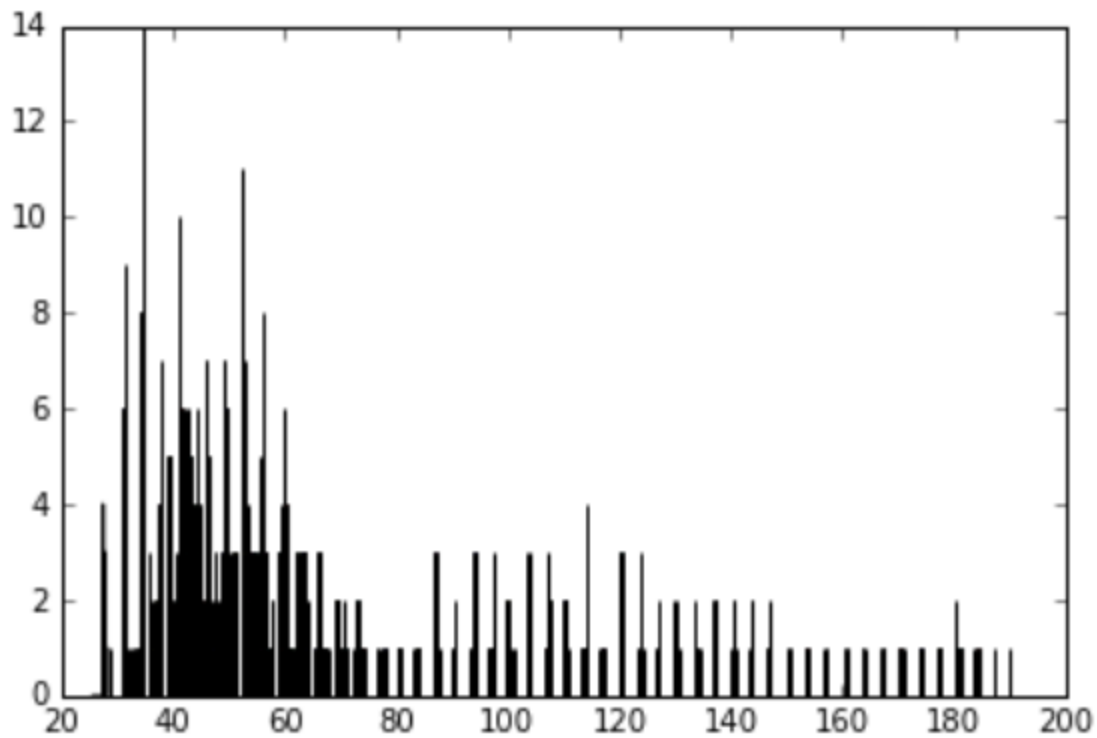
An example of an original image is this:



After grayscale, it looks like the following:

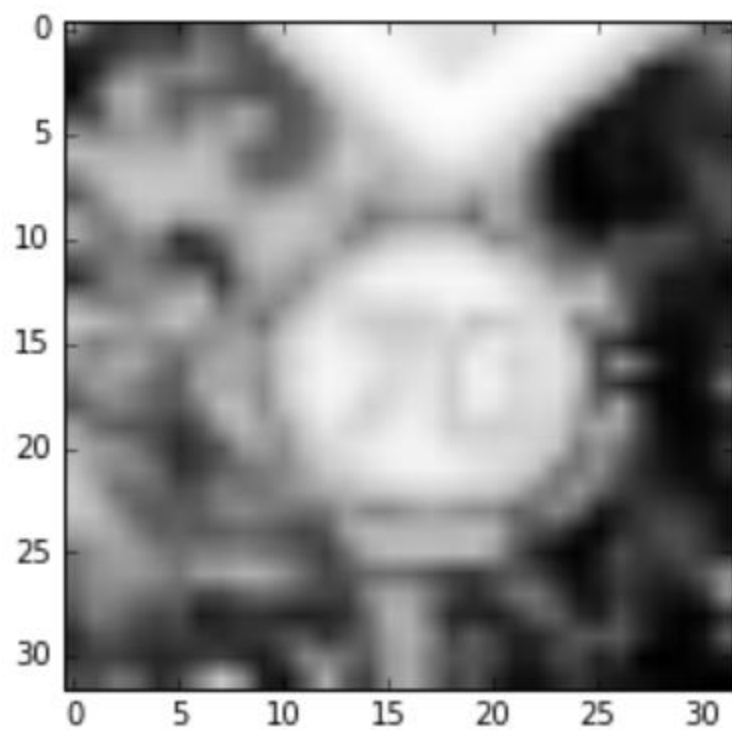


The Histogram for the grayscale image is this:

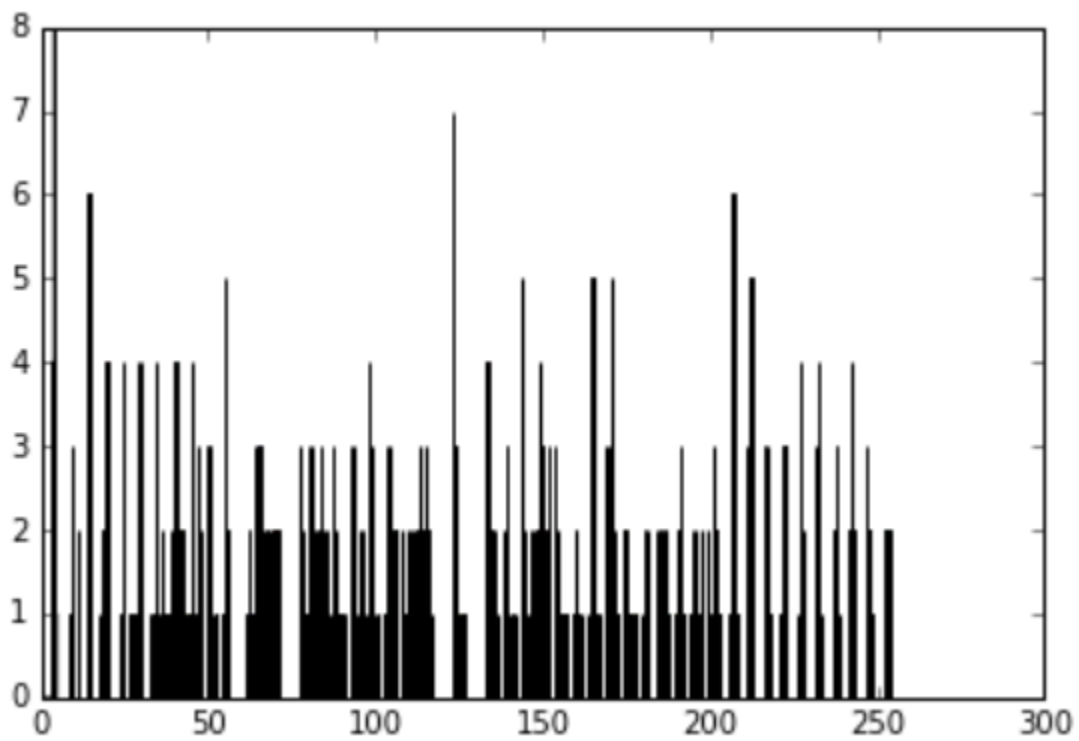


If you notice, the pixel values are all under 190. And there is a heavy concentration of value towards the lower end. Hence, after making the image grayscale, I decided to apply Histogram Equalization. It improves the contrast of the image by stretching the intensity of the image. In the above picture the third one is after applying histogram equalization. It is visually a clearer image.

The image after applying histogram equalization is this:



And the histogram is this:



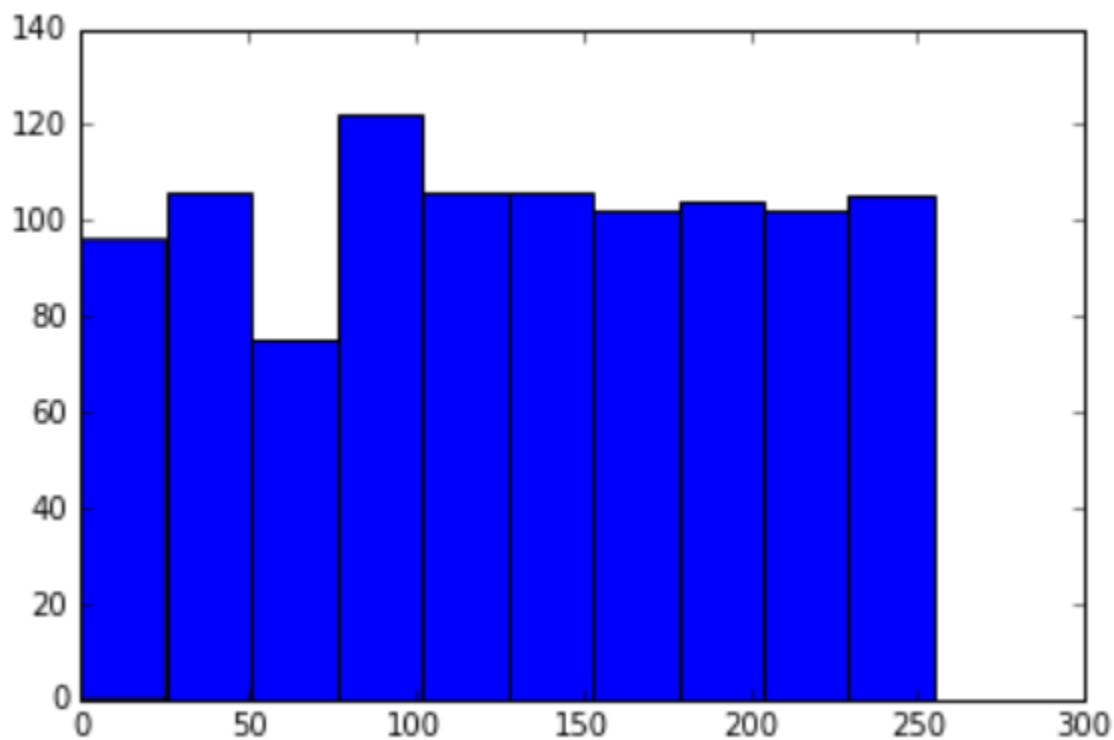
The pixel values are well spread from 0 to 255.

After that, I applied scaling. The webpage <http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html> describes is best. I am quoting it below.

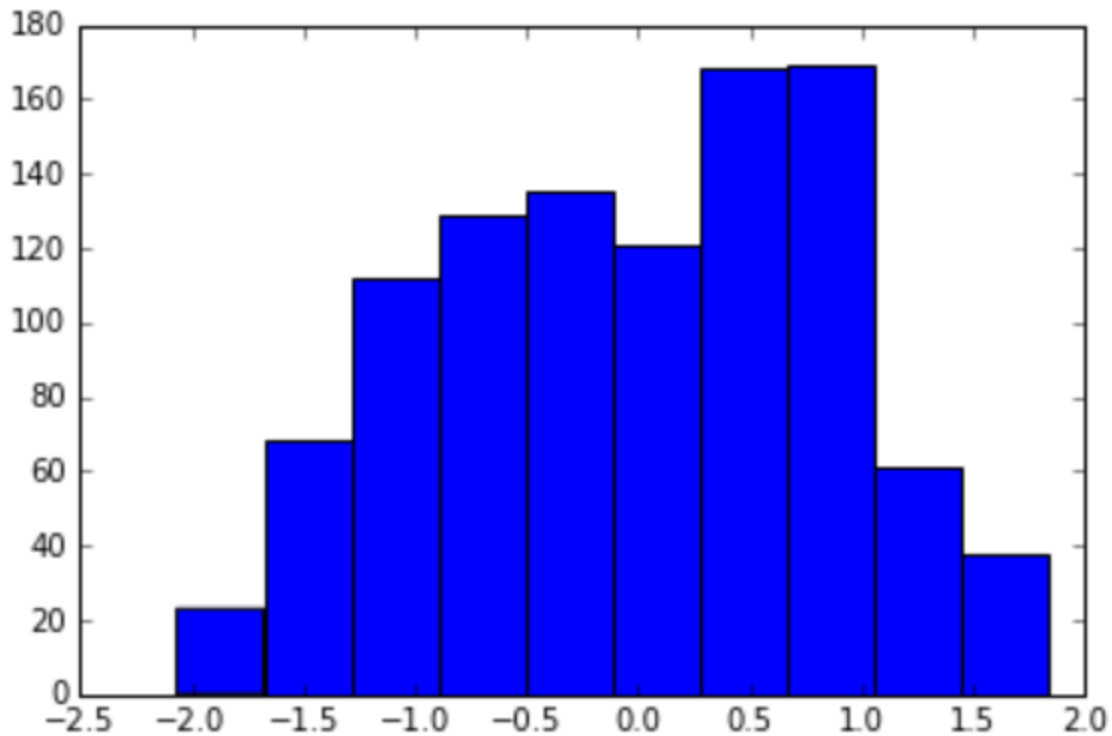
“Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual feature do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

For instance many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the L1 and L2 regularizers of linear models) assume that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger that others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.”

For example, if the image before scaling is like this:



After scaling it looks like this:



It is nicely centered around 0 and have a bell shaped curve also.

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.

The final model consisted of the following layers

Layer	Description
Input	32x32x1 Grayscale image
Convolution	20@5x5, 1x1 stride, padding=VALID, Output: 28x28x20
RELU	
Dropout	0%
Pooling	2x2 stride, padding=VALID, Output 14x14x20
Convolution	500@5x5, 1x1 stride, padding=VALID, Output: 10x10x500
RELU	
Dropout	0%
Convolution	500@5x5, 1x1 stride, padding=VALID, Output: 6x6x500
RELU	
Dropout	0.05%
Pooling	2x2 stride, padding=VALID, Output: 3x3x500
Flatten	Output: 4500
Fully Connected	Output: 100
RELU	
Fully Connected	Output: 200
RELU	

Output: Fully Connected	Logits: 43
Softmax	

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 training and optimizing of the model involved running an iterative script on top of the existing model and playing around with different parameter values.

The following parameters were chosen:

```
all_epochs = {10, 15, 20, 50, 100, 500}
all_batchsize = {64, 128, 256, 512} # , 1024, 2048}
all_mu = {0}
all_sigma = {0.1, 0.01, 0.001}
all_conv1depth = {6, 10, 20, 60, 200}
all_conv2depth = {16, 40, 100, 500}
all_fc1size = {100, 200, 400, 1000}
all_fc2size = {84, 116, 200, 500}
all_conv1keepprob = {0.5, 0.8, 0.9, 0.95, 1.0}
all_conv2keepprob = {0.5, 0.8, 0.9, 0.95, 1.0}
all_conv1adepth = {3, 5, 10, 30, 100}
all_conv2adepth = {8, 20, 50, 250}
all_fc1asize = {50, 100, 200, 500}
all_fc2asize = {42, 58, 100, 250}
all_conv1akeepprob = {0.5, 0.8, 0.9, 0.95, 1.0}
all_conv2akeepprob = {0.5, 0.8, 0.9, 0.95, 1.0}
all_learningrate = {0.001} # {0.01, 0.001, 0.0001}
```

I randomly selected a combination of parameters and passed to the Python file, which was running the Traffic classifier.

Later, I removed the "epochs" parameter from the iteration and used the validation accuracy to decide to exit the training. I took an average of last N validation accuracy and when it started drooping down, I exited the training. The following code shows that.



```

validation_accuracy = evaluate(X_valid, y_valid)
lastNacc[1:] = lastNacc[:-1]
lastNacc[0] = validation_accuracy
lastNvalidation_accuracy = np.mean(lastNacc)
wfile.write("{}: Validation Accuracy = {:.3f} ({:.3f})\n".format(i+1, validation_accuracy, lastNvalidation_accuracy))
if validation_accuracy <= lastNvalidation_accuracy:
    print("Breaking")
    break

```

Another approach, I tried was to use random values from conv-depth, fc-size and keep-probs. Earlier, I showed that I gave specific values to try out. But, in this case, I gave random ranges.

```

e = randint(6, 20)
f = randint(16, 100)
g = randint(100, 200)
h = randint(84, 150)
i = uniform(0.8, 1.0)
j = uniform(0.8, 1.0)
k = randint(3, 10)
l = randint(8, 50)
m = randint(50, 100)
n = randint(42, 75)
o = uniform(0.8, 1.0)
p = uniform(0.8, 1.0)

```

In the above code, the variables, "e" to "p" correspond to the various parameters, I can randomize.

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 are:

- Training set accuracy: 99.7%
- Validation set accuracy: 97.2%
- Test set accuracy: 94.0%

Approaches tried before the final one

1. Raw LeNet: LeNet architecture without changing the convolution depth or any other parameter was not giving enough accuracy. I was barely touching 90% on validation set.
2. LeNet style with different parameters: I kept the number of layers the same as LeNet, but tried different values for the depth each layer. These also, didn't give much accuracy beyond 90%.
3. Changing the color space to YUV: Didn't help much.
4. Using only Y component of the color: I thought, this would be equivalent to grayscale. But, I didn't have great luck in terms of high accuracy.
5. Inception Module: I used two LeNet style nets and combined the result at the output stage. One side had bigger convolution depth and pooling depths and the other one had smaller ones. It took longer to train. And the accuracy was hitting at and around 94%.
6. Manual tuning of all the above parameters: I manually tuned all the parameters before using a iteration script. I found that batch-size of 64 is good because above that would sometimes run out of memory. Also, I learning rate was fixed to 0.001 as I didn't see any other learning rate as good.
7. Another important approach was taking was to slowly grow the size of the network. Meaning first convolution layer depth would be smaller than second and second would be smaller than third. And then slowly converge to a smaller size at the end. So, second full connected layer would be smaller than the first fully connected layer. Overall, this strategy was working in both my manual attempts and script based attempts. In the, I was surprised that the second full connected layer is larger than the first full connected layer. All the other layer sizes conform my initial thought process.

## Test a Model on New Images

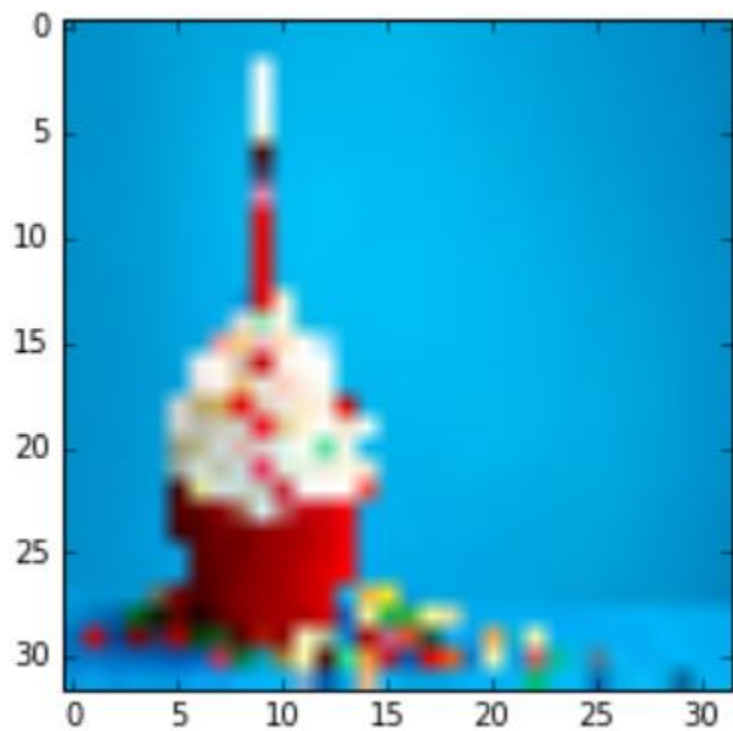
1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.



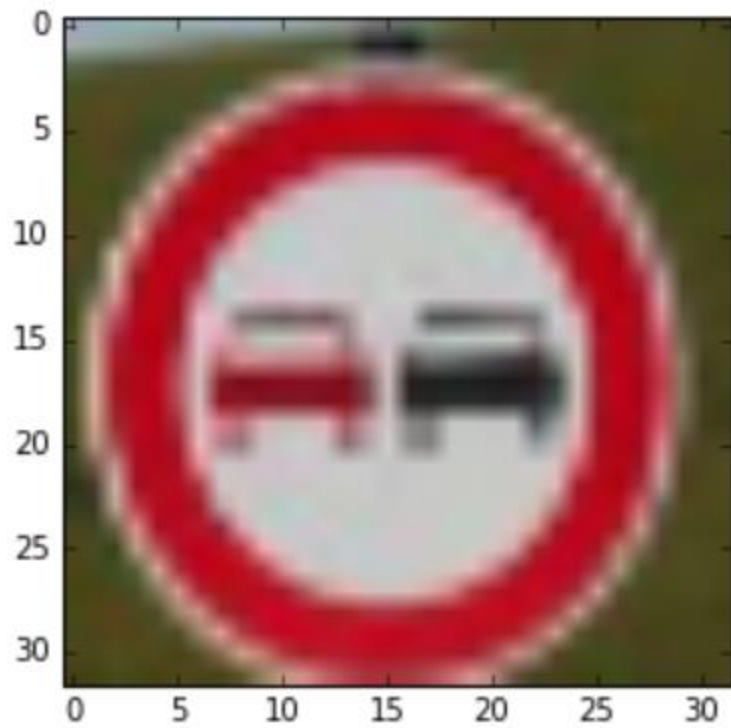
This image is very close to the “general caution” sign. I want to see if it gets classified correctly. (Indeed, in my model, if I reduce the epochs, this image gets classified as “general caution” sign)



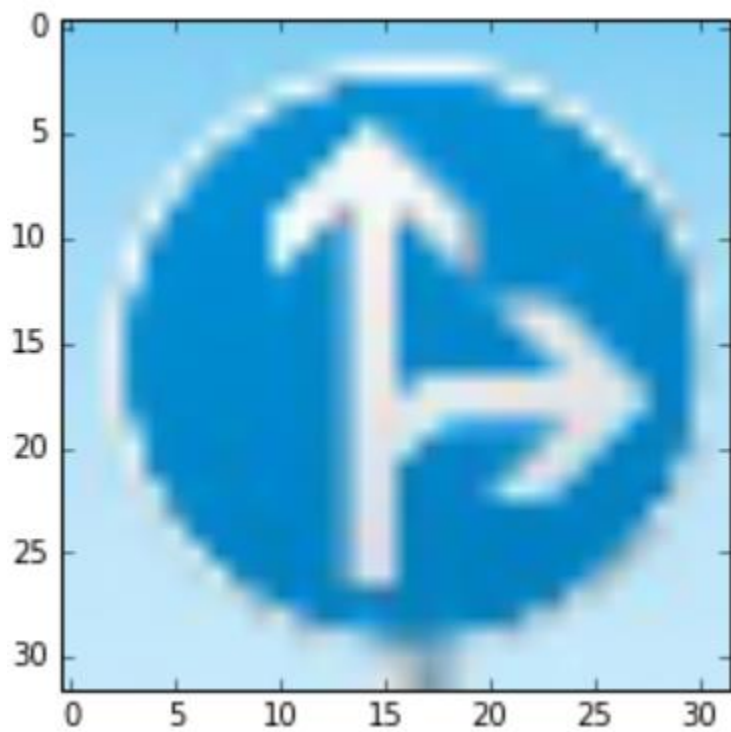
The above is not an image. It is more of a clipart with perfect white background.



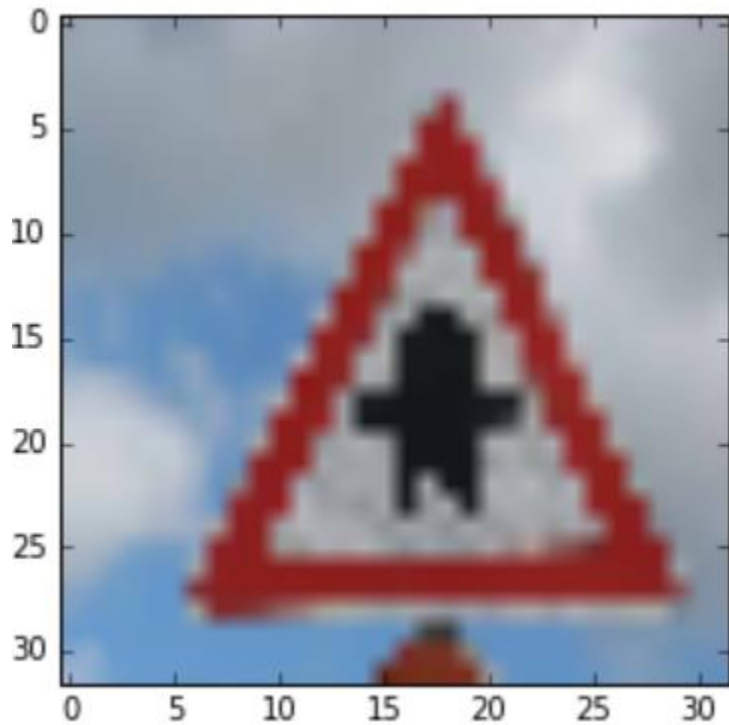
This is not a traffic sign. It is a cupcake with a candle on it.



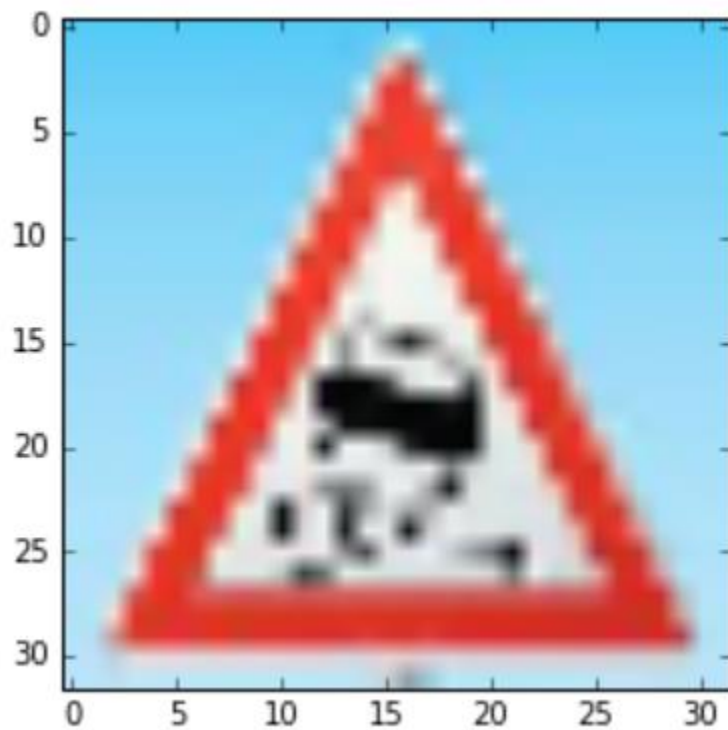
Want to see if “No passing” and “No passing for vehicles of 3.5 tons” can be caught properly. They both have similar signs.



Blue sign with blue background. I want to see, if the gray-scaling is working fine.



The above pic can be confused with other signs like “general caution” or “pedestrian”. Also, the background is not uniform in this pic.





Frankly speaking, this pic is not clear even with naked eye. Since, I clipped this image from a different image, I know what it is.

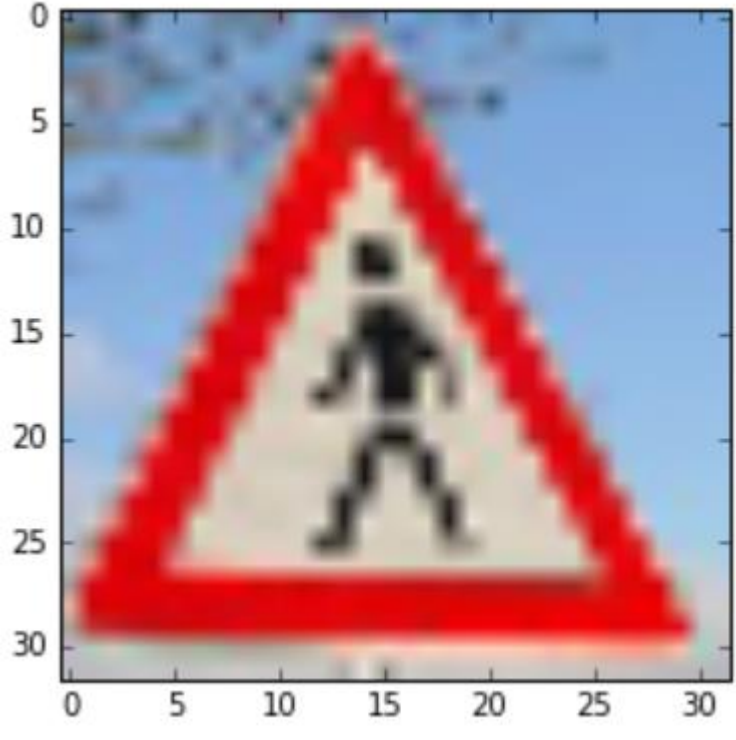


Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set

Image	Prediction	Comment
Pedestrians	Pedestrians	
No Entry	Stop	This was a not a real image. It was a clipart.
Cupcake	Keep Left	This was not even a traffic sign.
No passing	No passing	
Go straight or right	Go straight or right	
Right of way at the next intersection	Right of way at the next intersection	
Slippery road	Slippery road	



If I ignore the trick images, then I got 100% accuracy on the test image. It is well in terms with the test accuracy of 94%.


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.

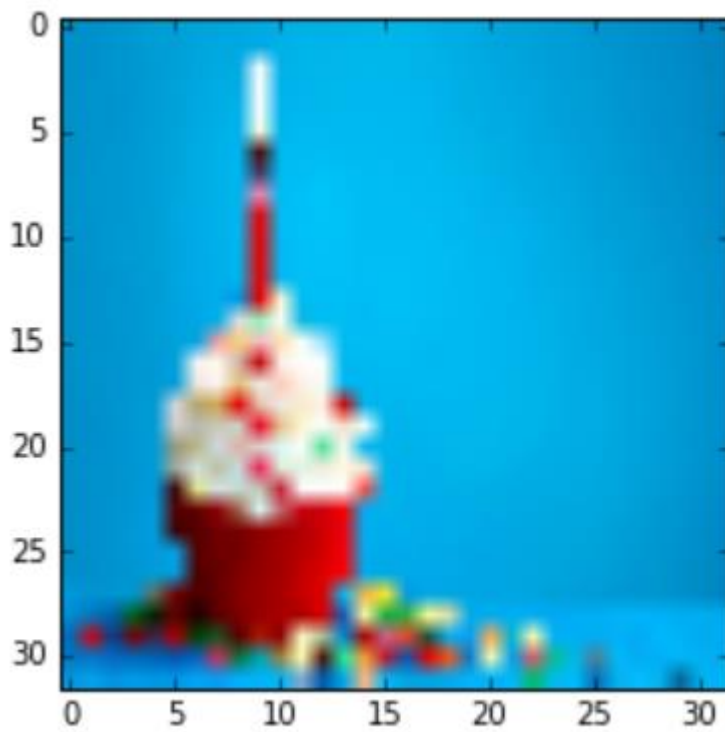
Original Image	Prediction Image	Pro b
	 Pedestrians	0.59
		0.22

	General Caution	
	 Right of way at the next intersection	0.19
	 Go straight or left	0.02



Original Image	Prediction Image	Pro b
	 Stop	1.00

Original Image	Prediction Image	Pro b
	 Keep left	0.18



No vehicles

0.11



Traffic signals

0.10



Speed limit 30km/h

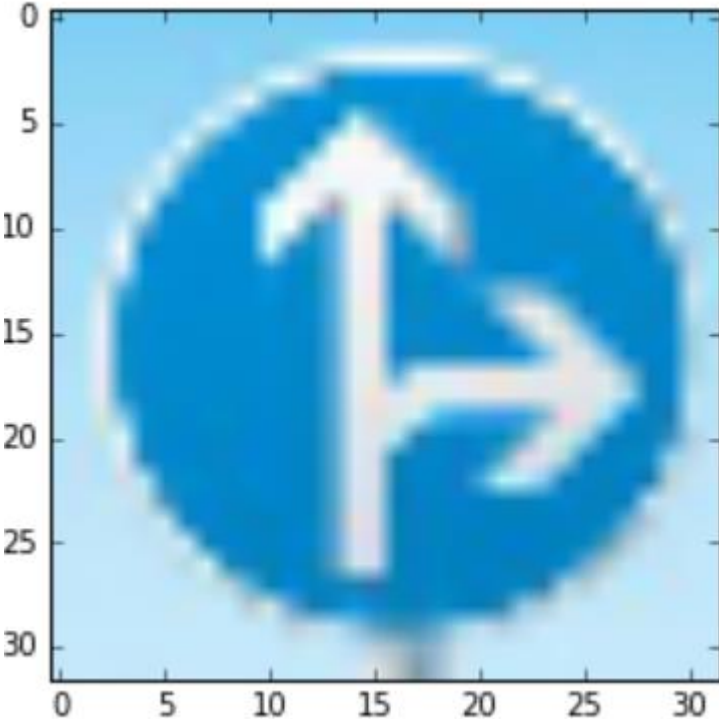

0.08



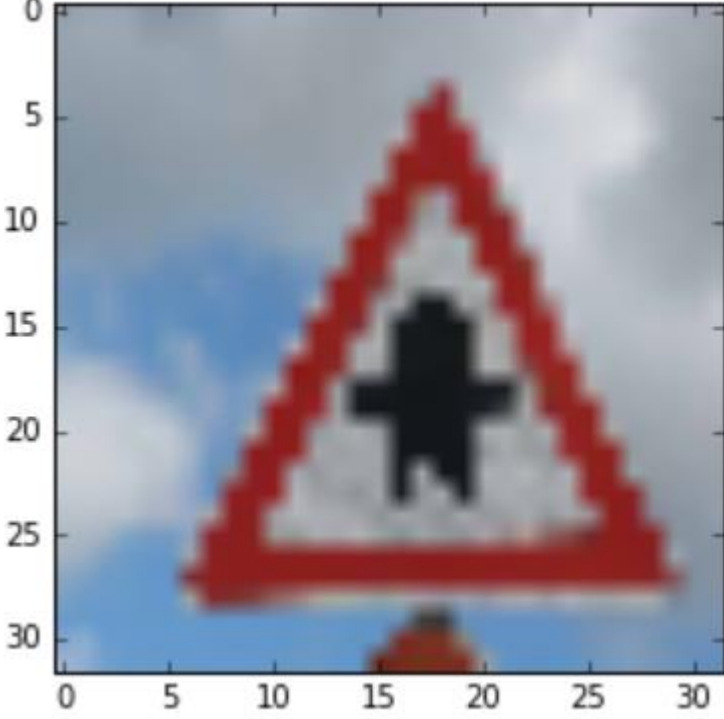



Speed limit 120km/h

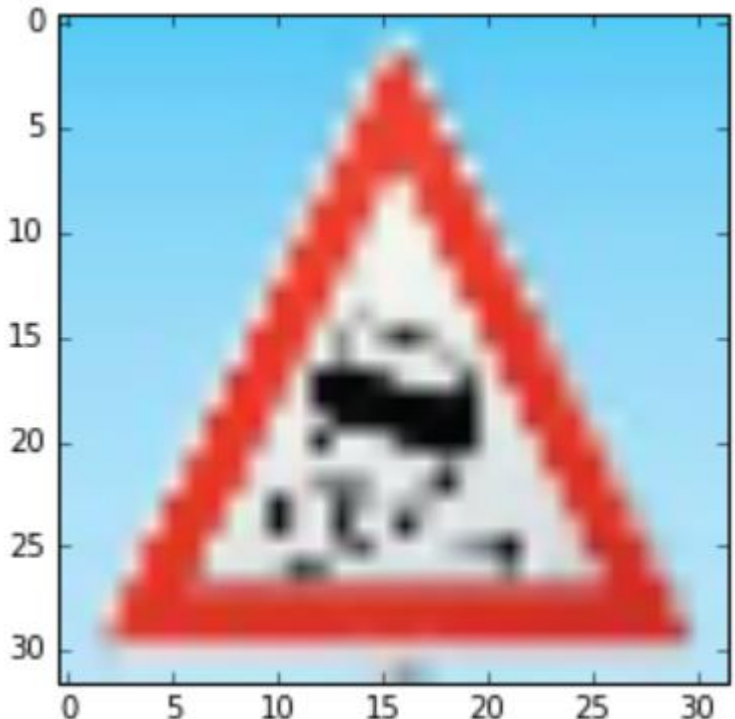

0.08

Original Image	Prediction Image	Pro b
	 <p data-bbox="948 814 1088 848">No passing</p>	1.00

Original Image	Prediction Image	Pro b
	 <p>Go straight or right</p>	1.00

Original Image	Prediction Image	Pro b
	 <p>Right of way at the next intersection</p>	0.61

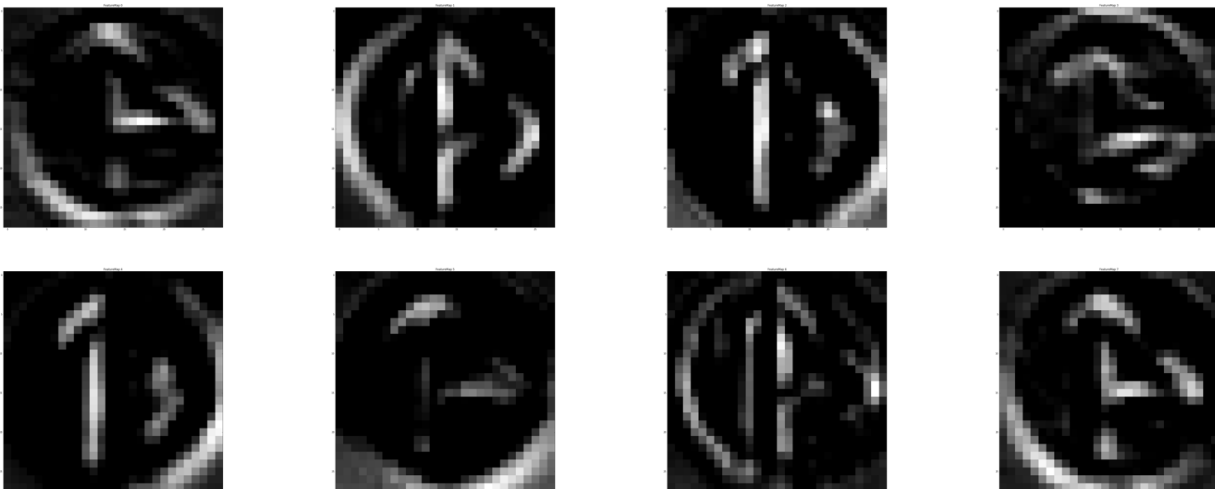
	 <p>Priority road</p>	0.20
	 <p>Double curve</p>	0.14
	 <p>Roundabout mandatory</p>	0.05

Original Image	Prediction Image	Pro b
	 Slippery road	1.00

## Visualizing the Neural Network

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

The output of the convolution network 1, has some of these pics.



If you see, the first image is learning a different curve than the second image. Overall all the images are learning the curves. But, first one alone can't recognize the sign. It understands part of the outer circle. The third image, is learning the upward pointing arrow. The last image in this list is learning both upward and right pointing arrows.