writeup.md

Traffic Sign Recognition

Writeup

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

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.

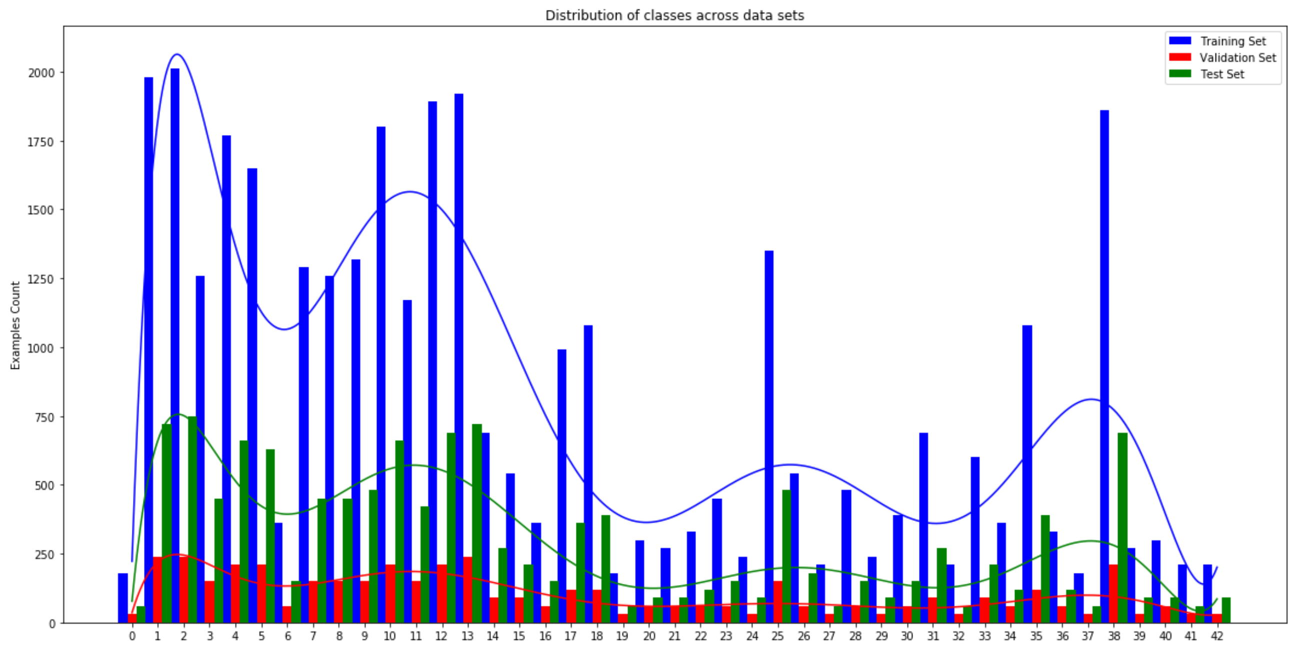
I only used basic numpy functionality to inspect the training, validation and test data sets and their shapes. Therefore, I got to the following numbers.

- The size of training set is ? Number of training examples = 34799
- The size of the validation set is ? Number of validation examples = 4410
- The size of test set is ? Number of testing examples = 12630
- The shape of a traffic sign image is ? Image shape = (32, 32) (for RGB, it also has 3 channels)
- The number of unique classes/labels in the data set is ? Number of classes = 43

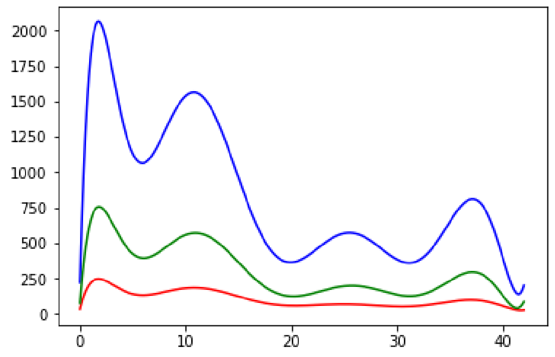
2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. I wanted to see the data distribution for all of the sets (training, validation and test).

In the first plot I wanted to see the actual numbers as well, but also plotted a fitted curve to clearly examine the distribution.



The second plot just helps distinguish the distribution curves easier.



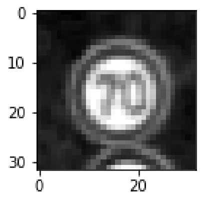
I observed that the distributions are roughly similar, with big changes in amplitude due to the sizes of each of the data sets.

A clear disadvantage of the data set was observed through these plots though: the fact that some of the classes had way more data points than others, the ration, in some cases, even getting closer to a factor of 10. I could have addressed this by data augmentation (for example creating more variations of data points for the classes with a few count, starting from the data points we already had - either rotation, or adding noise to the images, trying to increase the overall count), but I decided to try and build a model good enough for this sort of not so uniform data set, and see how good can the model get even in situations like these.

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.)

I pre-processed the images by converting them to grayscale and normalizing the pixel values. Here is an example of such an image.



Initially I trained the model on RGB images. The results were still decent, getting closer to an accuracy rate of ~89%. Since in most cases, the color of the traffic sign might not bring much more relevant information about the type of traffic sign, I decided to go on and try gray-scaling. This reduces the number of features that the model needs to train on, focusing on the essential features of the image.

Normalization was done in order to adjust the data (pixel values) to zero mean and variance equals to one. This is necessary in order to optimize the learning process through gradient descent.

As mentioned above, I did not do any data augmentation, although I considered it for the reasons stated before.

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 was a slight modification of LeNet, by adding some dropout steps as well. Here is a detailed description of each layer.

Layer	Description
Input	32x32x1 Grayscale image
Convolution 5x5	1x1 stride, valid padding, outputs 28x28x6
RELU	
Max Pooling	2x2 stride, outputs 14x14x6
Convolution 5x5	1x1 stride, valid padding, outputs 10x10x16
RELU	
Max Pooling	2x2 stride, outputs 5x5x16
Dropout	Keep probability 50%
Flatten	Output 400
Fully Connected	Output 120
RELU	
Fully Connected	Output 84
RELU	
Dropout	Keep probability 50%
Fully Connected	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.

I trained the model with an Adam Optimizer that minimizes a loss function. This loss function is defined as following:

```
reduce_mean(softmax cross entropy) + L2_regularization(model weights)
```

I will explain in the next section why I decided to use this loss function, over the original one from LeNet.

The model was trained over `10` epochs, with a batch size of `128`, on my local CPU (laptop). I used a learning rate of `0.001` and a regularization lambda of `0.0001`. Moreover, for the dropout regularization step, I used a "keep probability" of `50%`.

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.

I trained the model initially with the same optimizer that was described in LeNet: the Adam Optimizer for minimizing a the mean softmax cross entropy. This worked well enough, getting me around 89%-91% on grayscale images, but I wanted something more. After reading some more through Michael Nielsen's online book, I found out about the L2 regularization and how it improves the problem of overfitting on the training set (his descriptions are very clear and awesome, highly recommend!).

My final model results were:

- training set accuracy of ? 97.4%
- validation set accuracy of ? 93.7%
- test set accuracy of ? 91.5%

If an iterative approach was chosen:

- What was the first architecture that was tried and why was it chosen?

I started off with LeNet, because that was the architecture that I got introduced to in the class / lab.

- What were some problems with the initial architecture?

These were not *problems*, but more improvements that we could do to it in order to get a higher accuracy rate - adding various regularization techniques, to avoid training overfitting (dropout, L2 regularization on the loss function).

- How was the architecture adjusted and why was it adjusted? Typical adjustments could include choosing a different model architecture, adding or taking away layers (pooling, dropout, convolution, etc), using an activation function or changing the activation function. One common justification for adjusting an architecture would be due to overfitting or underfitting. A high accuracy on the training set but low accuracy on the validation set indicates over fitting; a low accuracy on both sets indicates under fitting.

I described this above - I adjusted it mostly due to overfitting on the training set, and the adjustments involved adding more regularization.

- Which parameters were tuned? How were they adjusted and why?

I tried changing the `epochs` size, but given the fact that I was training on my CPU, I eventually decided to leave it at `10`. With more epochs (I tried `15`), the model did not get much better in terms of accuracy though. I did not adjust other parameters as I was happy with the performance so far. I only added one extra parameter, the `regularization lambda`, set to `0.0001` as also described in Michael Nielsen's book, mentioned above.

- What are some of the important design choices and why were they chosen? For example, why might a convolution layer work well with this problem? How might a dropout layer help with creating a successful model?

Given the fact that we are analyzing core features within images, a convolution layer helps extracting these features and abstracting them from the other details present in the image. Therefore, one of the key decisions was the grayscale preprocessing of the images, which lowered the details from each input image. In terms of the dropout, this was a regularization technique used in order to avoid overfitting on the training set. Overfitting on the training set was resulting in lower accuracy for the validation and test sets.

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.

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

Label 11

Label 18

Label 2

Label 32

Label 9

I chose image #3 because of its angled position, and image #2 due to the weather noise that was naturally added in, trying to see how robust my model was. The other images are other normal examples of traffic signs, similar to what was present in the original data set. The images have been downloaded at higher scales, but resized accordingly, to 32x32.

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 the results of the prediction:

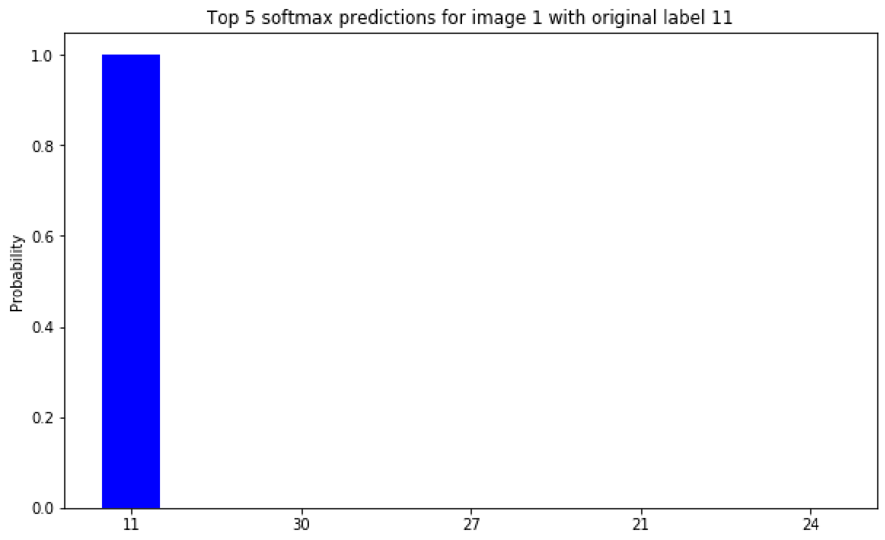
Image	Prediction
50km/h	No entry
No passing	No passing
Right-of-way	Right-of-way
General caution	General caution
End of all speed and passing limits	End of all speed and passing limits

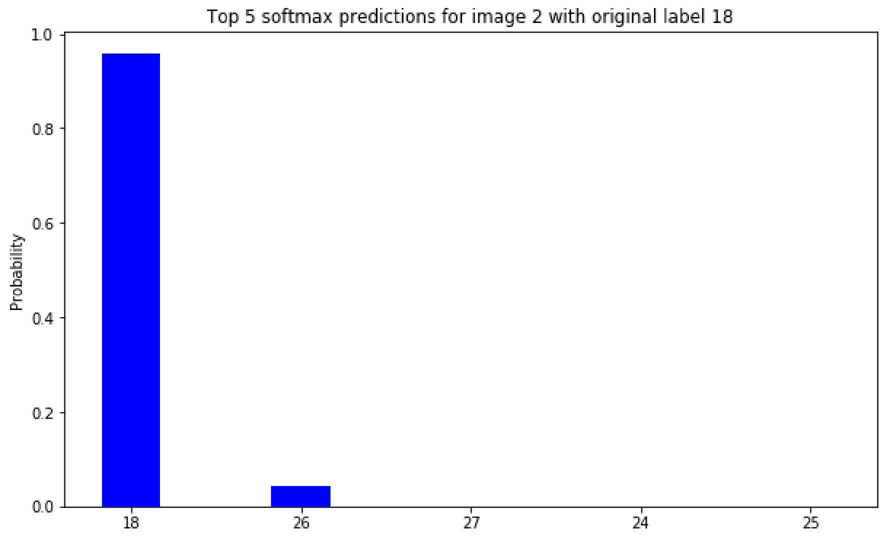
The model correctly predicted 4 out of the 5 input traffic signs, which gives an overall accuracy of `80%` on these newly seen images. This is way below the accuracy on the test set (`91.5%`).

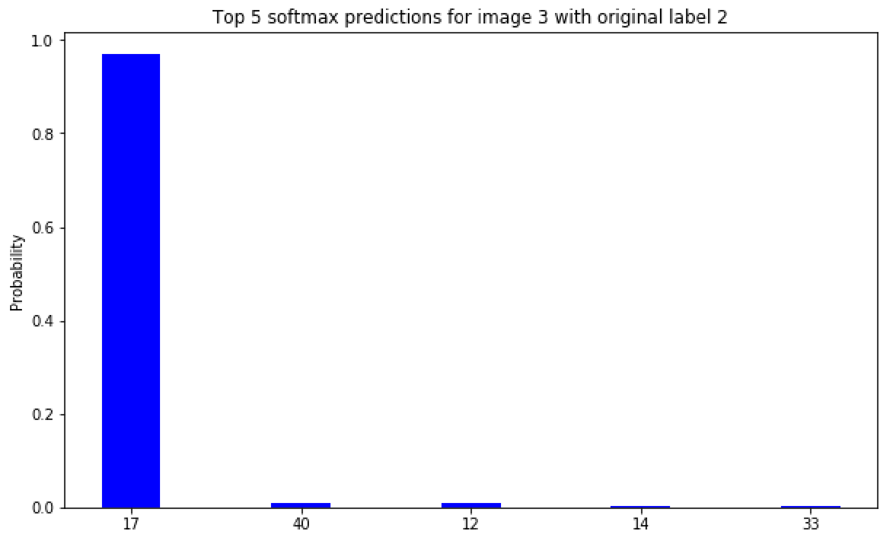
3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions"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

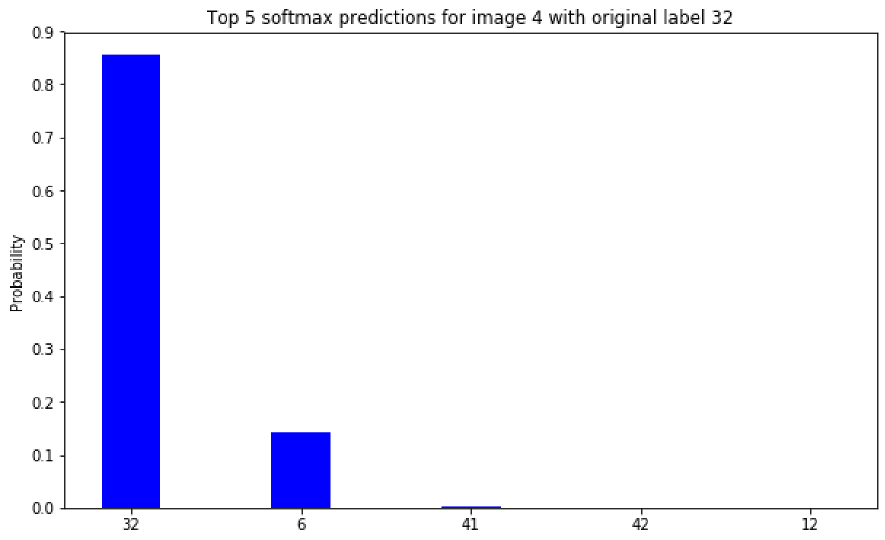
The code for making predictions on my final model is located in the 11th cell of the Ipython notebook.

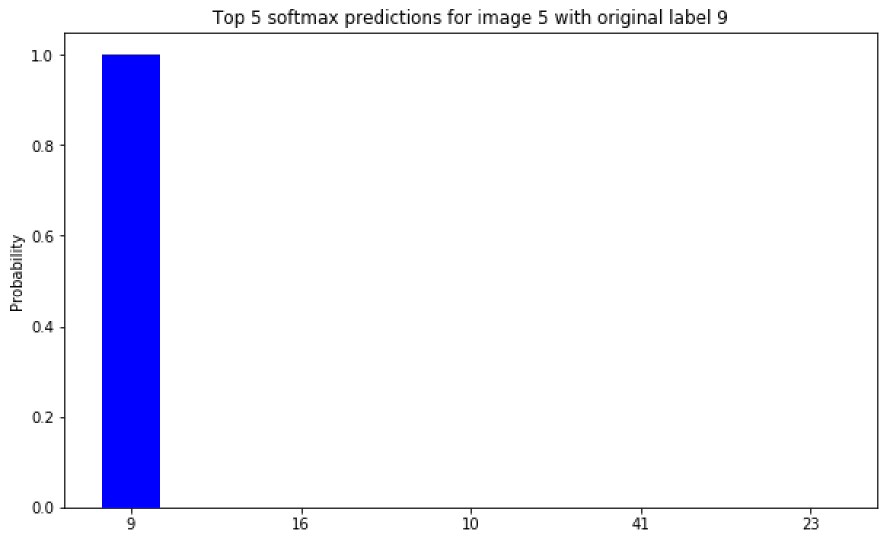
Below are the graphs that show the top 5 softmax probability distributions for each of the 5 images I used.


Top 5 softmax predictions for image 1 with original label 11
Probability
1.0
0.8
0.6
0.4
0.2
0.0
11
30
27
21
24


Top 5 softmax predictions for image 2 with original label 18
Probability
1.0
0.8
0.6
0.4
0.2
0.0
18
26
27
24
25


Top 5 softmax predictions for image 3 with original label 2
Probability
1.0
0.8
0.6
0.4
0.2
0.0
17
40
12
14
33



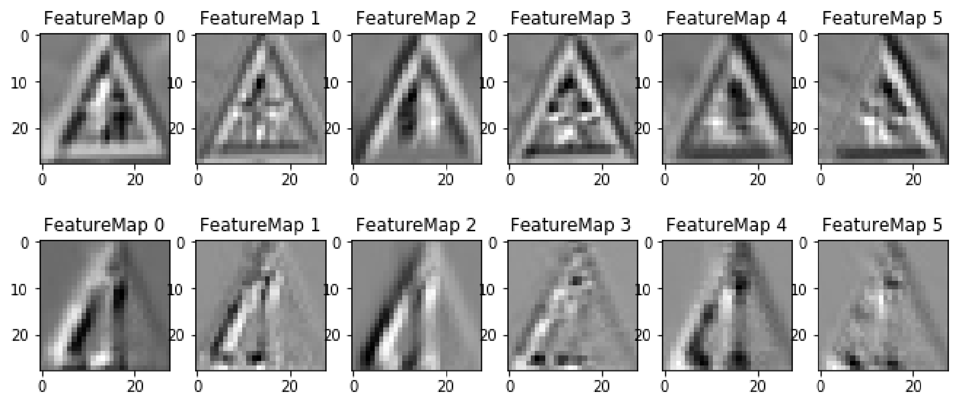


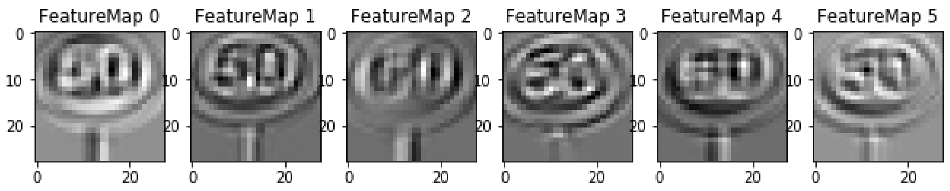
Something strange I observed was related to image #3, with original label `2` - the model was not able to predict this image at all, as label 2 is not even in its top 5 softmax predictions. I assume this might be due to the angled position, although this seems quiet strange, in comparison to image #2 (covered by snow) which was correctly predicted.

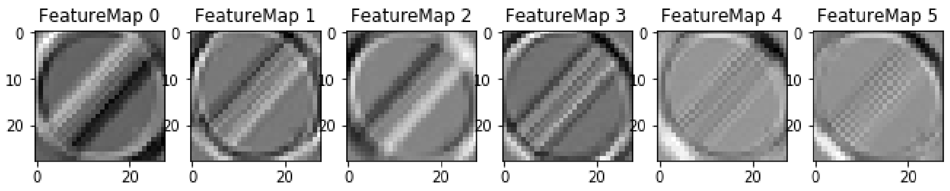
(Optional) Visualizing the Neural Network (See Step 4 of the Ipython notebook for more details)

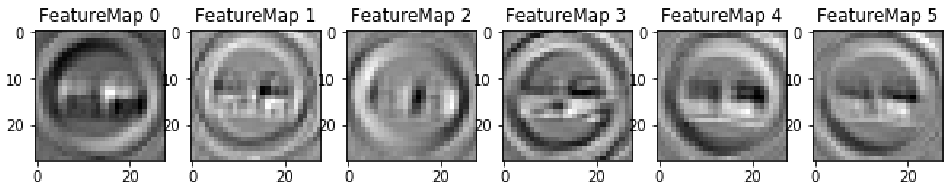
####1. Discuss the visual output of your trained network's feature maps. What characteristics did the neural network use to make classifications?

Below are the activation feature maps for the first convolutional layer. It seems like the neural network uses differences in contrast (shades of gray) in order to make the classifications.



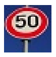






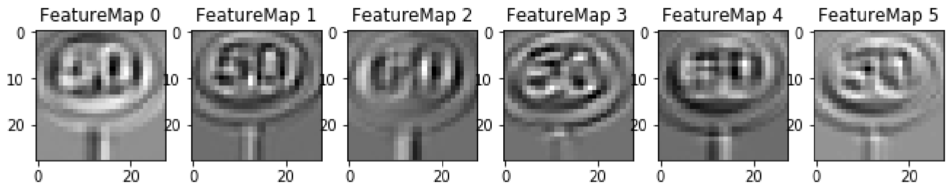
Looking at these images (especially the 3rd row, associated with the only downloaded image that my model was not able to properly classify), I am wondering why did it fail, since the feature map seems clear enough.

Let's look again at the image that caused problems to my model:

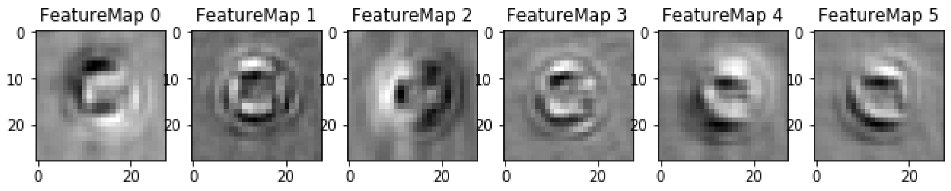


The label of this image should be Label 2, but my model predicted Label 17 instead. Let's compare the following 3 activation feature maps and see why this actually happened:

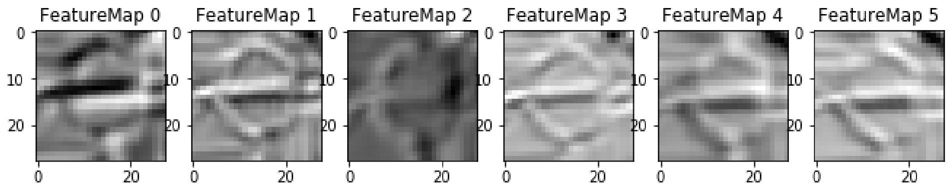
- activation feature map of my test image



- activation feature map of a training set image with Label 2



- activation feature map of a training set image with Label 17



We can see that there is a bigger similarity between the the first and the third feature maps, which makes me think that this is why my model incorrectly predicted the image.