Traffic Sign Recognition

Table of Contents

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 address 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 can find the project code in the ipython notebook as well as an exported html file at the same place in the folder as this report.

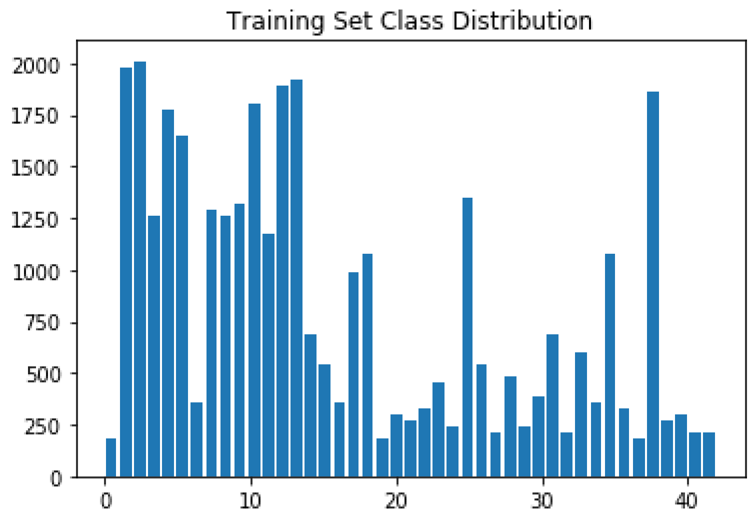
Data Set Summary & Exploration

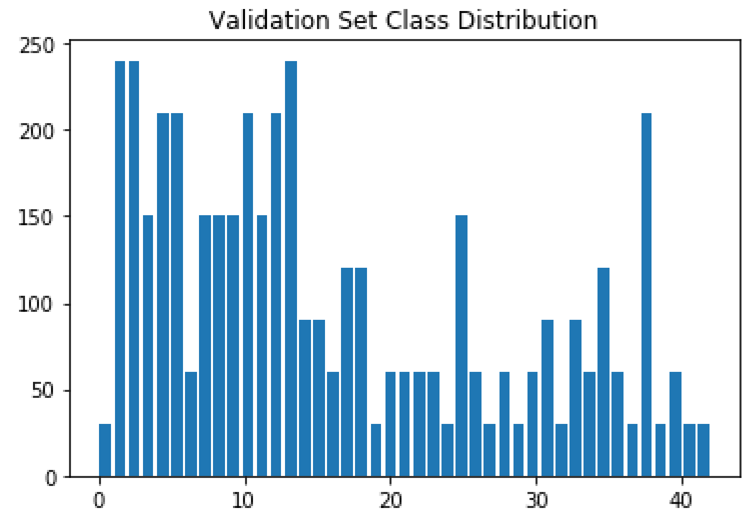
1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hard coding results manually.

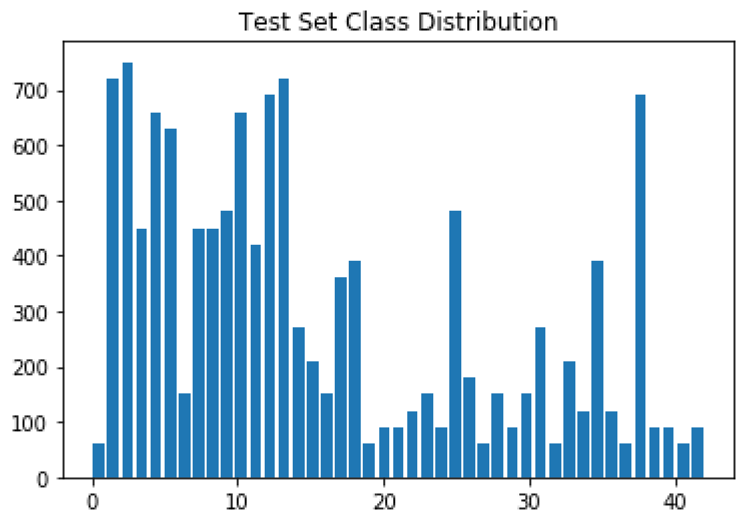
I used python and numpy methods to calculate the summary statistics of the traffic sign data set, and I get the results as following:

- Size of the training set is: 34799
- Size of the testing set is: 12630
- Size of the validation set is: 4410
- Shape of a traffic sign image is: 32 x 32 x 3
- Number of unique classes in the data set is: 43

2. Include an exploratory visualization of the dataset.







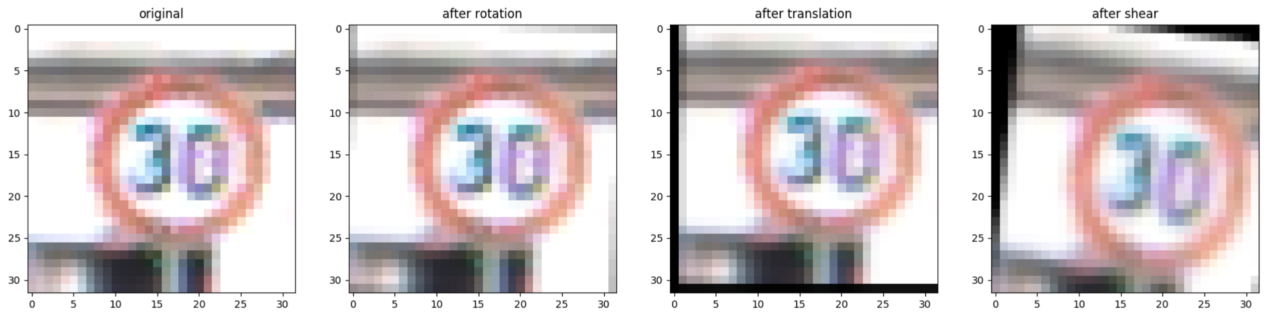
The class distribution of the data set are shown as above. The horizontal axis is the class label value and the bar represents the number of examples in the dataset. From the chart, we can see that the testing set is highly unbalanced, and it can cause a bias against those classes with small number of examples.

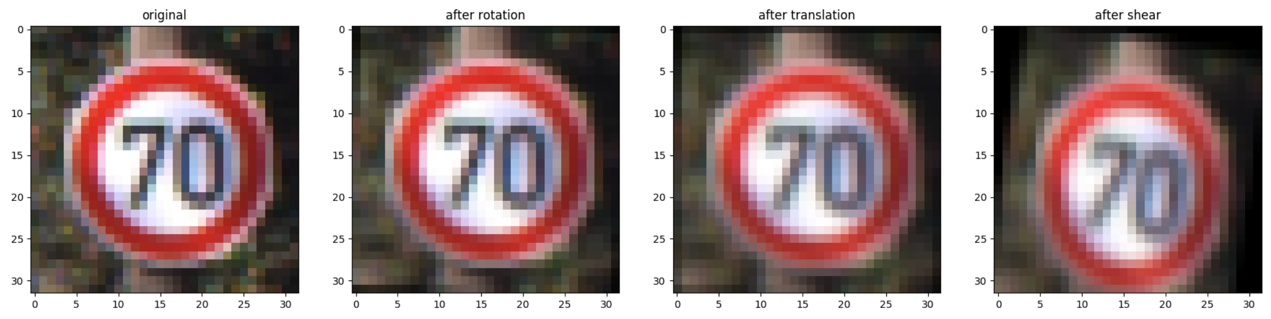
Design and Test a Model Architecture

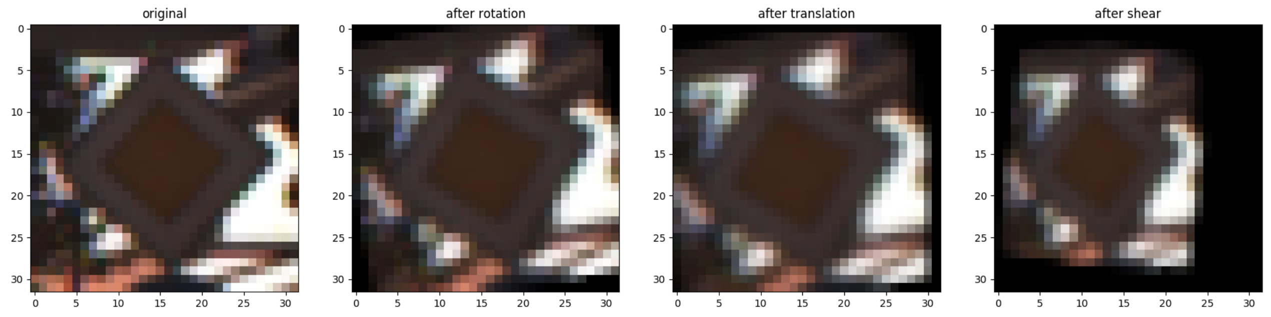
1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc. (OPTIONAL: As described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

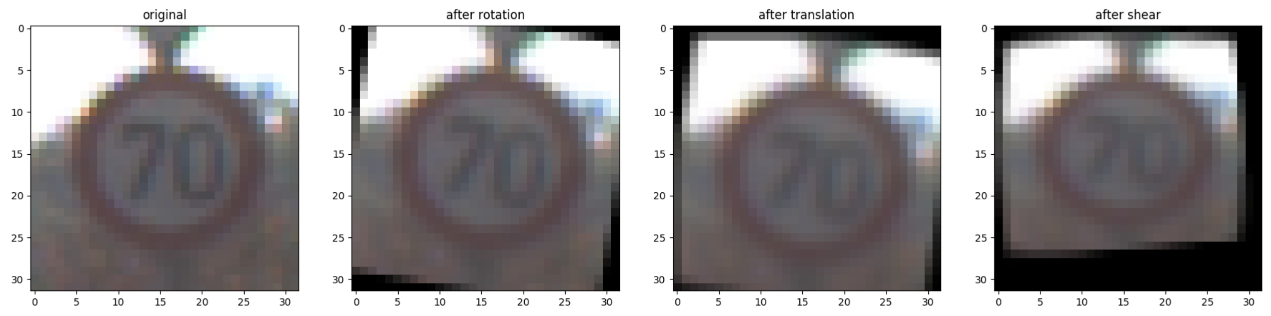
From the visualization of the training set, we could see that the training set is highly unbalanced with the number of samples for each class. Therefore, the first step I take is to augment the training set and try to make it balanced by generating fake augmented training data. After that, I convert the color image to gray scale and perform normalization on the gray scale image.

1.1 Augment training dataset





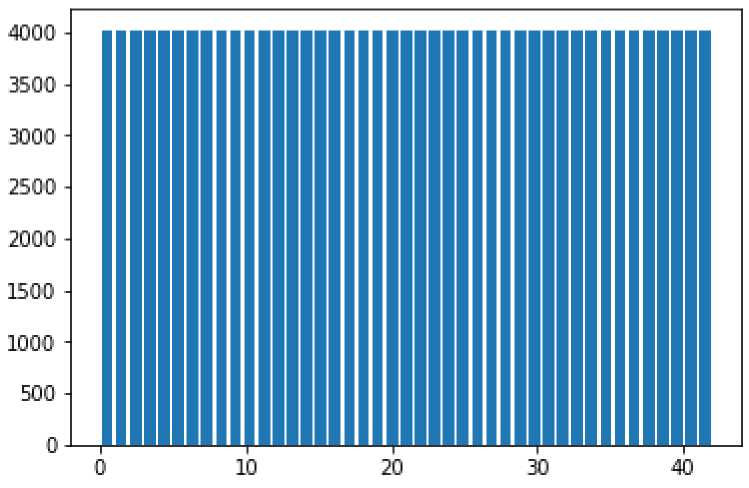




To augment the training set, I apply three operations to an original image. First, a random counter clockwise rotation between -15 degrees and +15 degrees around the center of the image is applied to the original image. Then, a random translation between -2 and +2 pixels along both horizontal and vertical axis is applied. After that, a random shear operation is applied. The details of this shear operation can be found from the source code.

By using these transformations to generate augmented data, it makes the training robust to small disturbs caused by camera angle or movement.

The images above show the result after applying each random transformation. To make the training data balanced across all classes, I generate different number of augmented images for different classes and make each class contain 4020 examples, including the original examples. The final training set contains 172,860 examples in total. And the class distribution of the testing set is balanced as shown in the following chart:

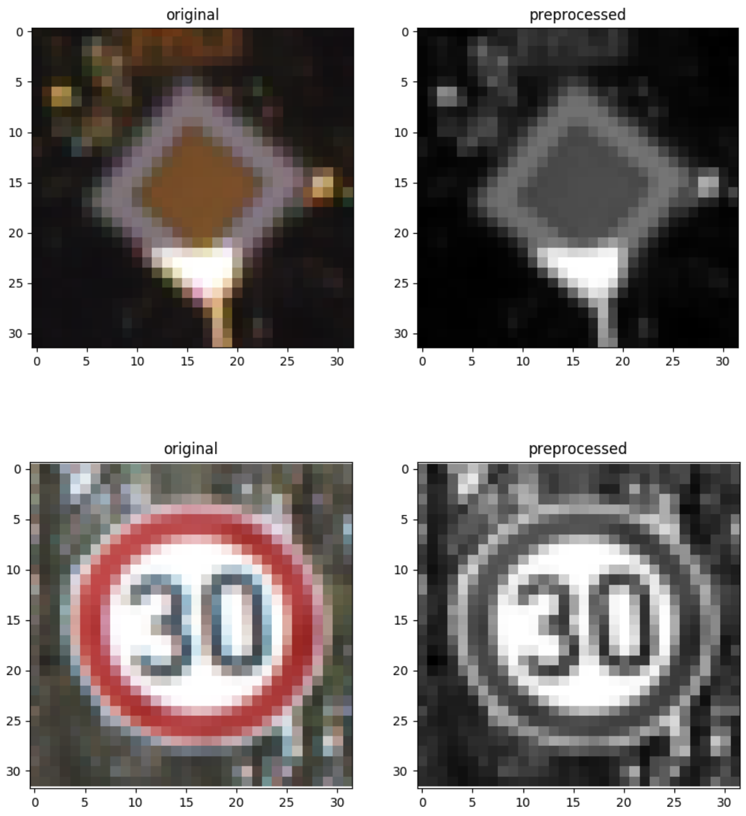


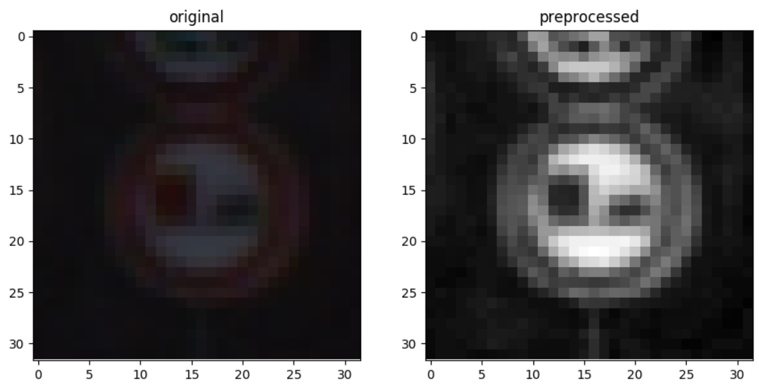
1.2 Pre-processing

After making the training dataset balanced by augmenting with randomly generated examples, I perform further process.

Firstly, I convert the color image to gray scale. According to the paper "Traffic Sign Recognition with Multi-Scale Convolutional Networks" by Pierre Sermanet and Yann LeCun, converting the image to gray scale will improve the performance. And also it seems color information is not so important for traffic signs, since those traffic signs in the data set can be fully described by the edges in the image. At the same time, using a single channel gray scale image can also reduce the size of weights as well as memory needed for storing the middle state of the image in the convolutional neural network.

Secondly, to prepare the data to be used by the neural network and to make it easier to converge when being feed for training, I normalize the gray scale images.





The above images show the result after converting the color image to gray scale image and normalize. After preprocessing, images change to 32 x 32 x 1.

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.

In this project, I have tried several different convolution neural network architectures, including the original LeNet with dropout, a deep convolutional neural network which I found from a Keras tutorial (https://chsasank.github.io/keras-tutorial.html), and a convolutional neural network using multi-scale features as mentioned in the paper "Traffic Sign Recognition with Multi-Scale Convolutional Networks" by Pierre Sermanet and Yann LeCun. Finally I choose the deep convolutional neural network from the Keras tutorial. The architecture of this model can be described as following:

Layer	Description
Input	32 x 32 x 1 gray scale image
Convolution 3 x 3 x 32	1 x 1 stride, same padding, outputs 32 x 32 x 32
RELU	
Convolution 3 x 3 x 32	1 x 1 stride, valid padding, outputs 30 x 30 x 32
RELU	
Max Pooling	2 x 2 stride, valid padding, outputs 15 x 15 x 32
Dropout	
Convolution 3 x 3 x 64	1 x 1 stride, same padding, outputs 15 x 15 x 64
RELU	
Convolution 3 x 3 x 64	1 x 1 stride, valid padding, outputs 13 x 13 x 64
RELU	
Max Pooling	2 x 2 stride, valid padding, outputs 6 x 6 x 64
Dropout	
Convolution 3 x 3 x 128	1 x 1 stride, same padding, outputs 6 x 6 x 128
RELU	
Convolution 3 x 3 x 128	1 x 1 stride, valid padding, outputs 4 x 4 x 128
RELU	

Max Pooling	2 x 2 stride, valid padding, outputs 2 x 2 x 128
Dropout	
Flatten	outputs 512
Fully Connected	outputs 512
RELU	
Dropout	
Fully Connected	outputs 43

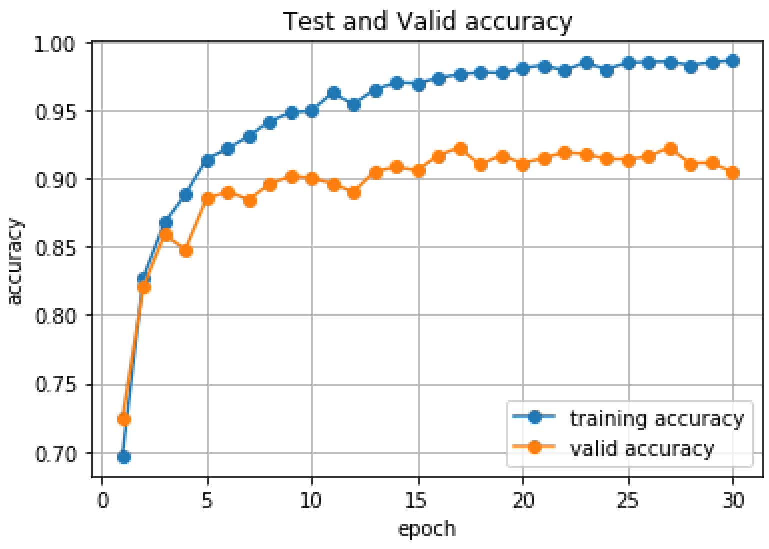
3. Describe how you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

When I train my model, I use the AdamOptimizer provided by Tensorflow as the optimizer. And after some tuning, I select the batch size to be 128, the number of epochs to be 30 and the keep probability for dropout to be 0.8. For the learning rate, I set the start learning rate to be 0.0005, and use a exponential decay with decay_steps to be 5000 and decay rate to be 0.96.

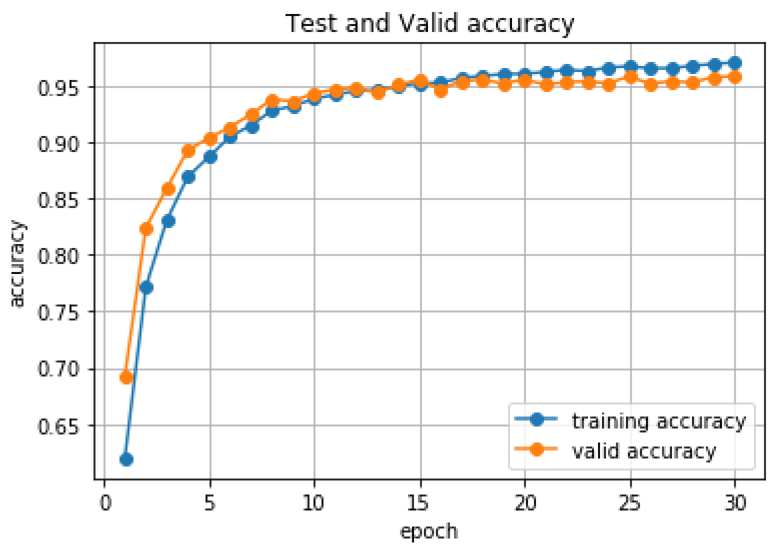
4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, 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 are: training set accuracy of 1.0, validation set accuracy of 0.996, and test set accuracy of 0.984.

At the beginning, I decide to start with the LeNet learned from the course, since it is already implemented and it is a good starting point. After some tuning of the hyper parameters and training it for 30 epochs, I got 0.986 for training accuracy, 0.905 for validation accuracy. The learning curve is shown as following:

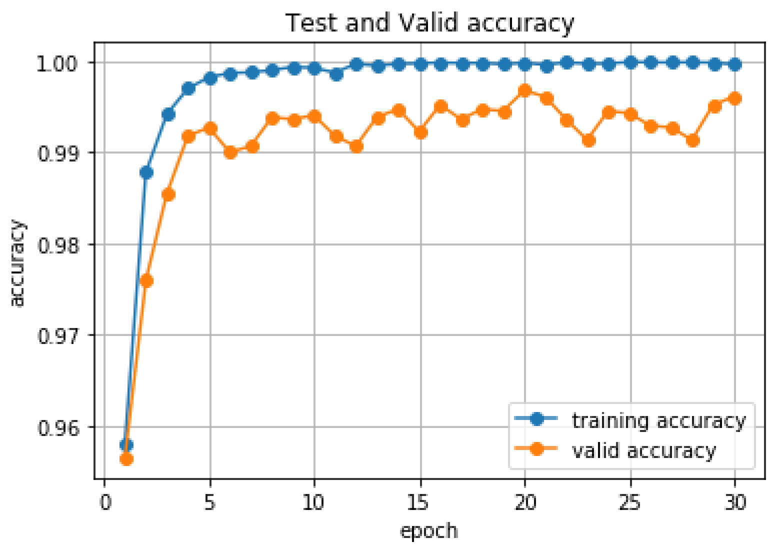


From this we can see that the model is overfitting since the validation accuracy is much lower than the training accuracy. Therefore, I decide to add dropout to the original LeNet to improve the regularization of the model. After some tuning with the keep probability, I chose it to be 0.8, and I got 0.971 for training accuracy, 0.958 for validation accuracy and 0.938 for testing accuracy. The learning curve of this model is as following:



By now the result achieved seems already quite good, and the model fits the problem quite well. However, if we compare the result with human result, there is still some distance. Considering the model already fits the problem, it is somehow underfitting. Therefore we need a more capable model. I have tried different model architectures and finally I select the model mentioned above.

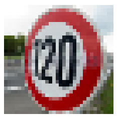
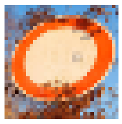
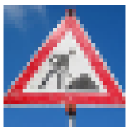
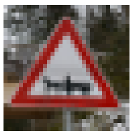
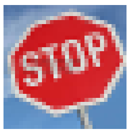
At the end, I got 1.0 for training accuracy, 0.996 for validation accuracy and 0.984 for testing accuracy. The learning curve of this model is shown as following:



Test a Model on New Images

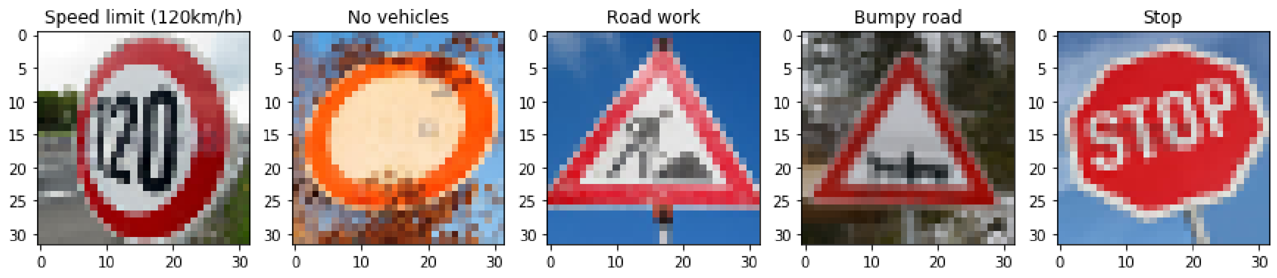
1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.

The five German traffic signs I have found from the web are as following:

Traffic Sign	Meaning	Difficulties
	Speed limit 120 km/h	The sign is not facing the camera in a proper way.
	No vehicles	The color of the sign is fade out and also covered partially by tree leaves, which may make it difficult to be classified.
	Road work	The sign has some glare effect due to the sunshine, which may also make it difficult to be classified.
	Horse Vehicle	This sign is not in the data set, the model can never classify it correctly. But it should predict some similar ones.
	Stop Sign	The sign is not straightly aligned in image, but the model should able to handle it.

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).

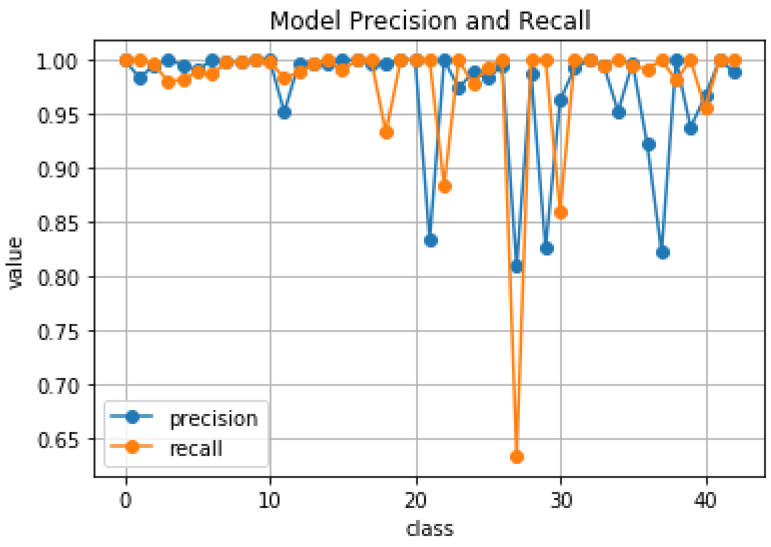
By feeding those new traffic signs to the trained model, I get the predictions as following, the text on top of the image represents the prediction and the image is right the image feed in.



The model is able to correctly predict 4 out of 5 traffic signs. The accuracy for the new images is 0.8. Compare to the accuracy on the old test set, which is 0.984, it seems the model is overfitting.

The only wrong case is the horse vehicle sign, which is not in the dataset at all. The trained model predicts it as bumpy road. Actually, this prediction is not bad, since it looks quite similar to a bumpy road sign.

I have analyzed the model in more detail by calculating the precision and recall for each traffic sign type from the test set. The result is as shown in the following figure:



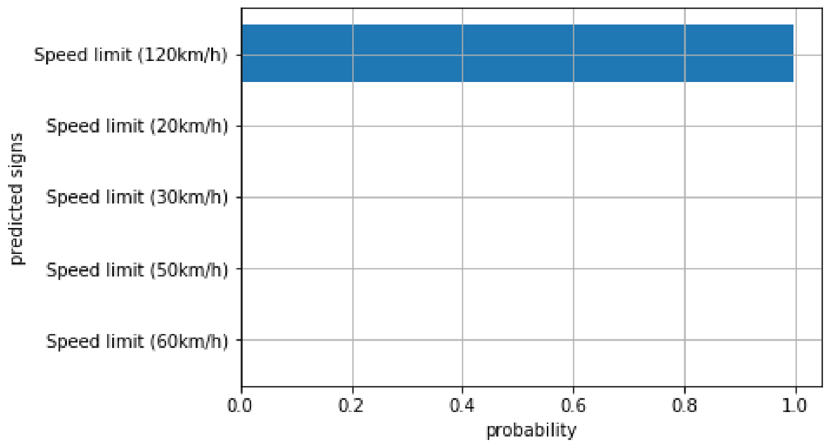
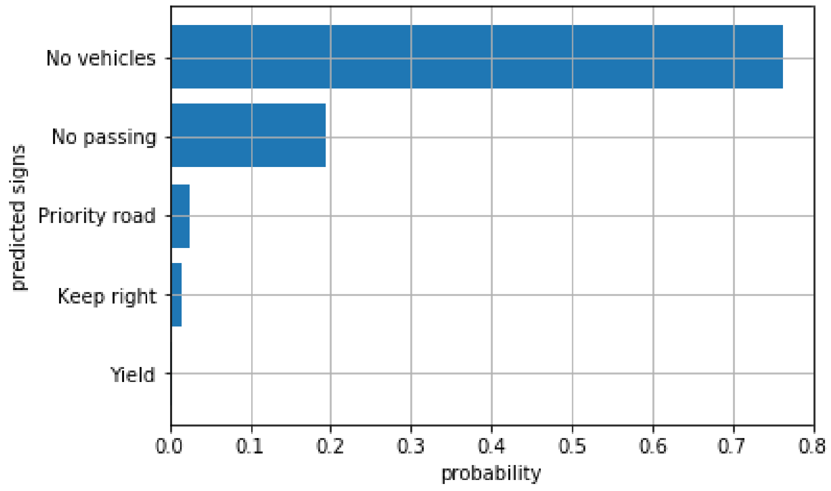
For the traffic sign images I got the from web, the precision and recall of the model is list as in the following table:

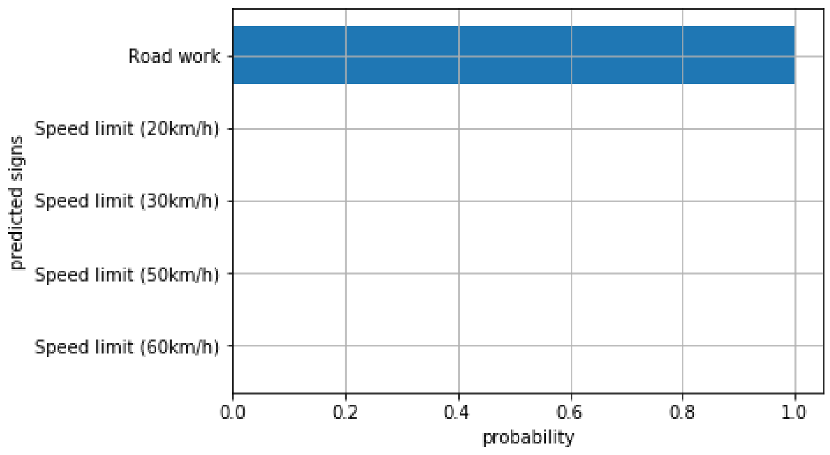
Traffic Sign	Precision	Recall
Speed limit (120km / h)	0.998	0.998
No vehicles	1.0	0.99
Road work	0.984	0.994
Bumpy road	1.0	0.883
Stop	0.996	1.0

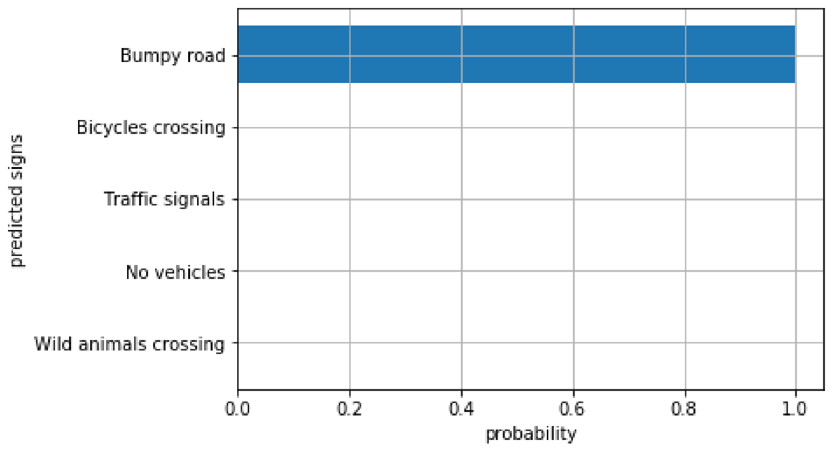
From the precision and recall chart, we can see that the model has already performed very well for most traffic signs, but there are still some signs which are not so well predicted. This could be used as an reference to augment the data on those less well performed signs and improve the performance of the model further.

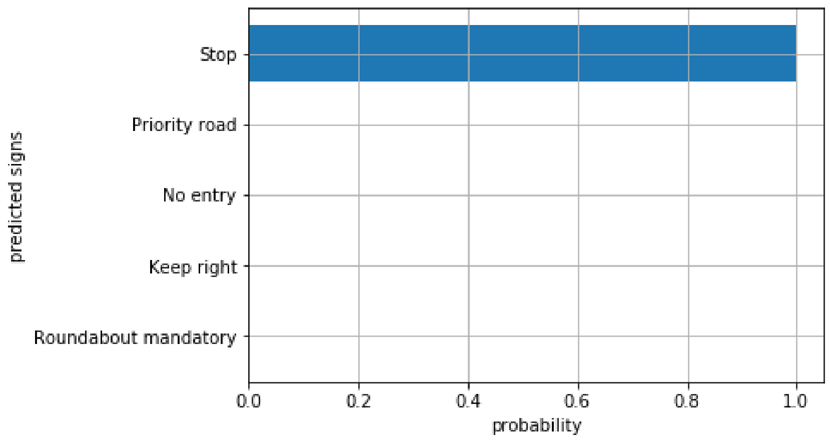
3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions"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

For each of the image above, the top 5 softmax probabilities are shown in the following table. From the top 5 softmax probabilities, we can see that for the new image “120 km/h”, “road work”, “bumpy road” and “stop” sign, the model is quite sure about the prediction. Though for the bumpy road, it actually is not a bumpy road but a horse vehicle sign. For the “no vehicles” sign, the model is about 76% sure for the “no vehicles” sign, and 19% for the “no passing” sign, it also predicts it to be a “priority road” sign with 2.5% and a “keep right” sign with 1.4%.

Image	Top 5 Softmax probabilities
	
	





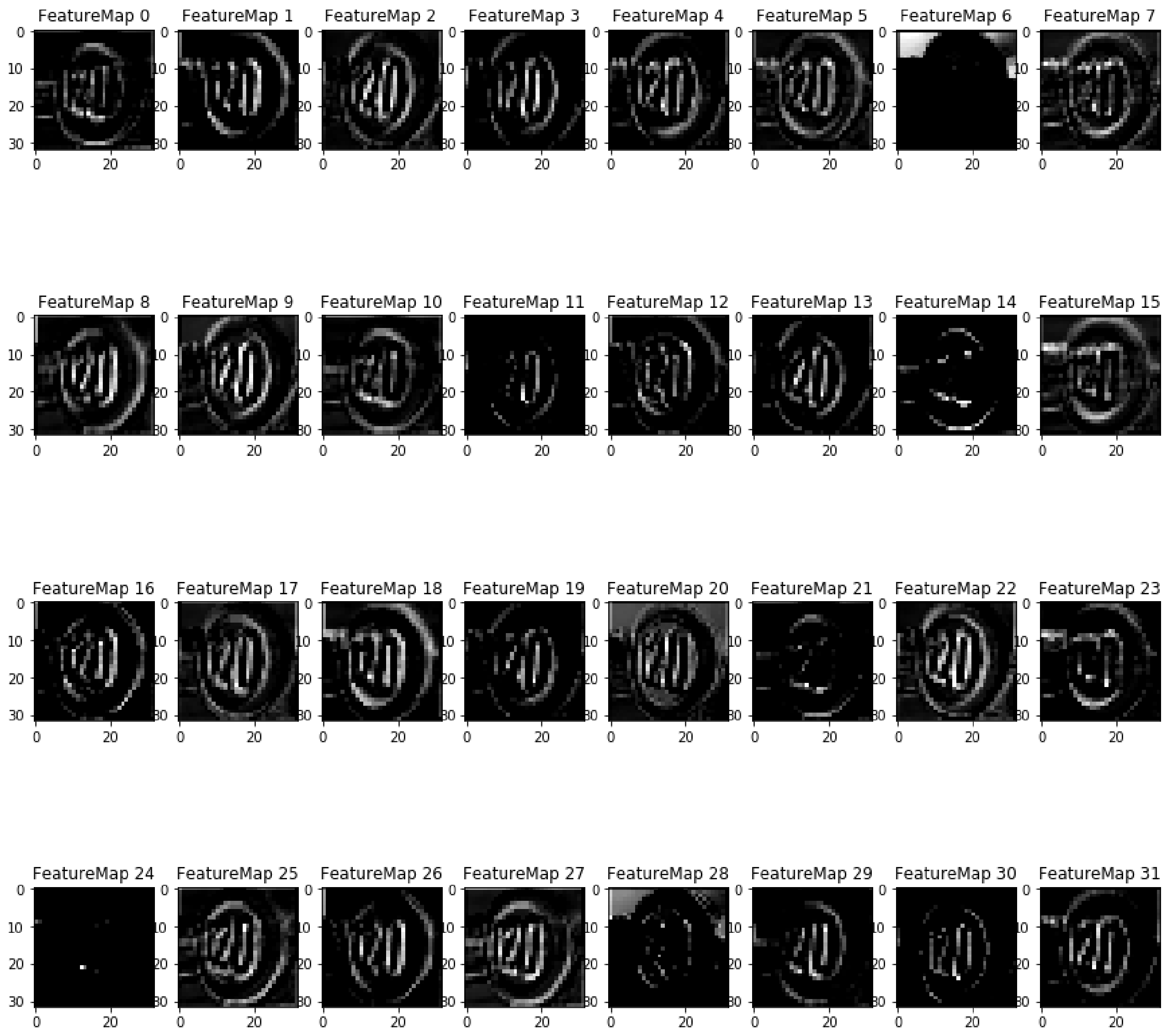


(Optional) Visualizing the Neural Network

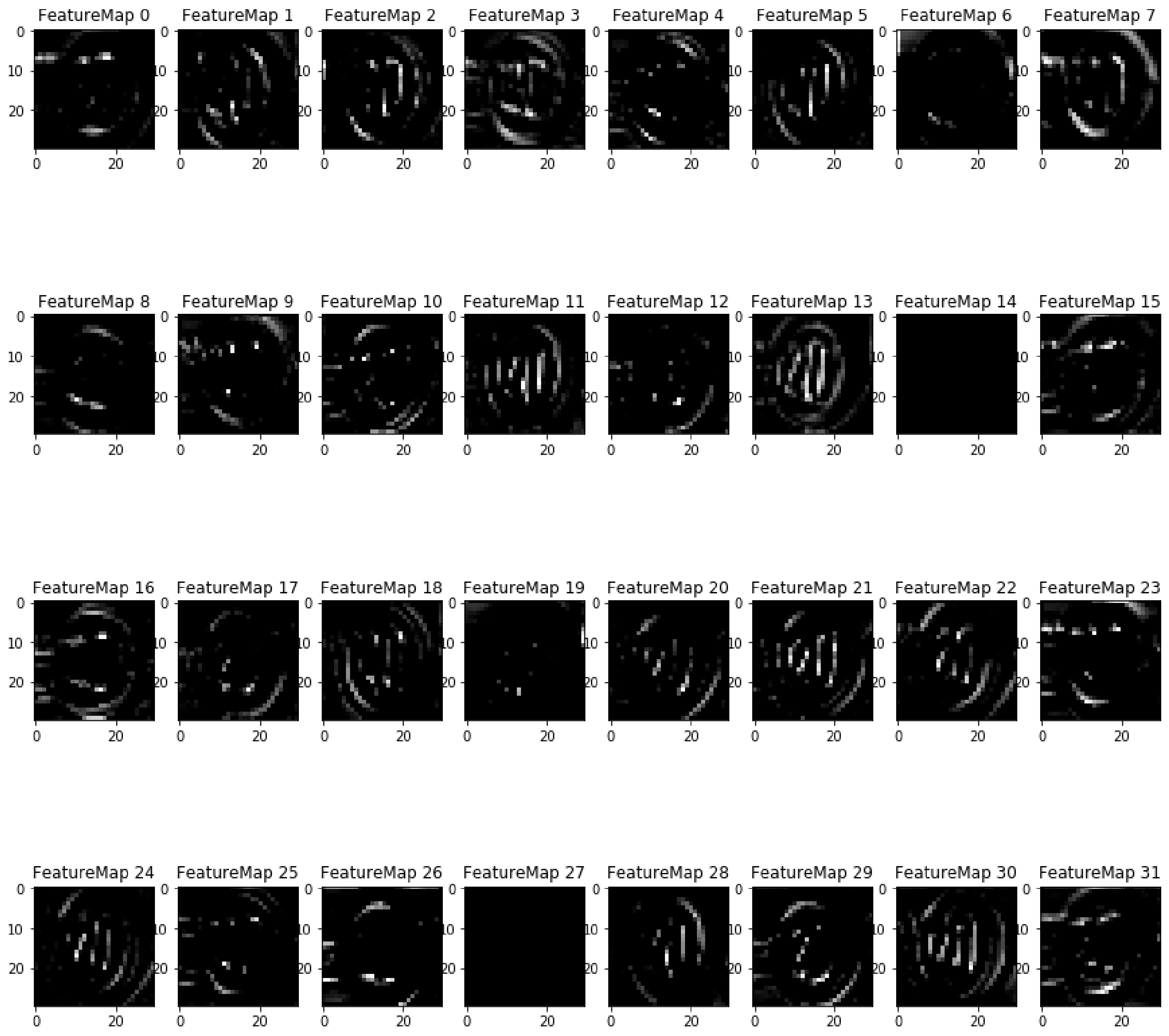
1. Discuss the visual output of your trained network's feature maps. What characteristics did the neural network use to make classifications?

The visualization of my trained network’s feature maps with the 120 km/h speed limit traffic sign image can be shown as following. From the visualization, we could see the neural network uses the edges detected in the image for classification, and it also seems to use the shape of the numbers and signs to help to classify the signs.

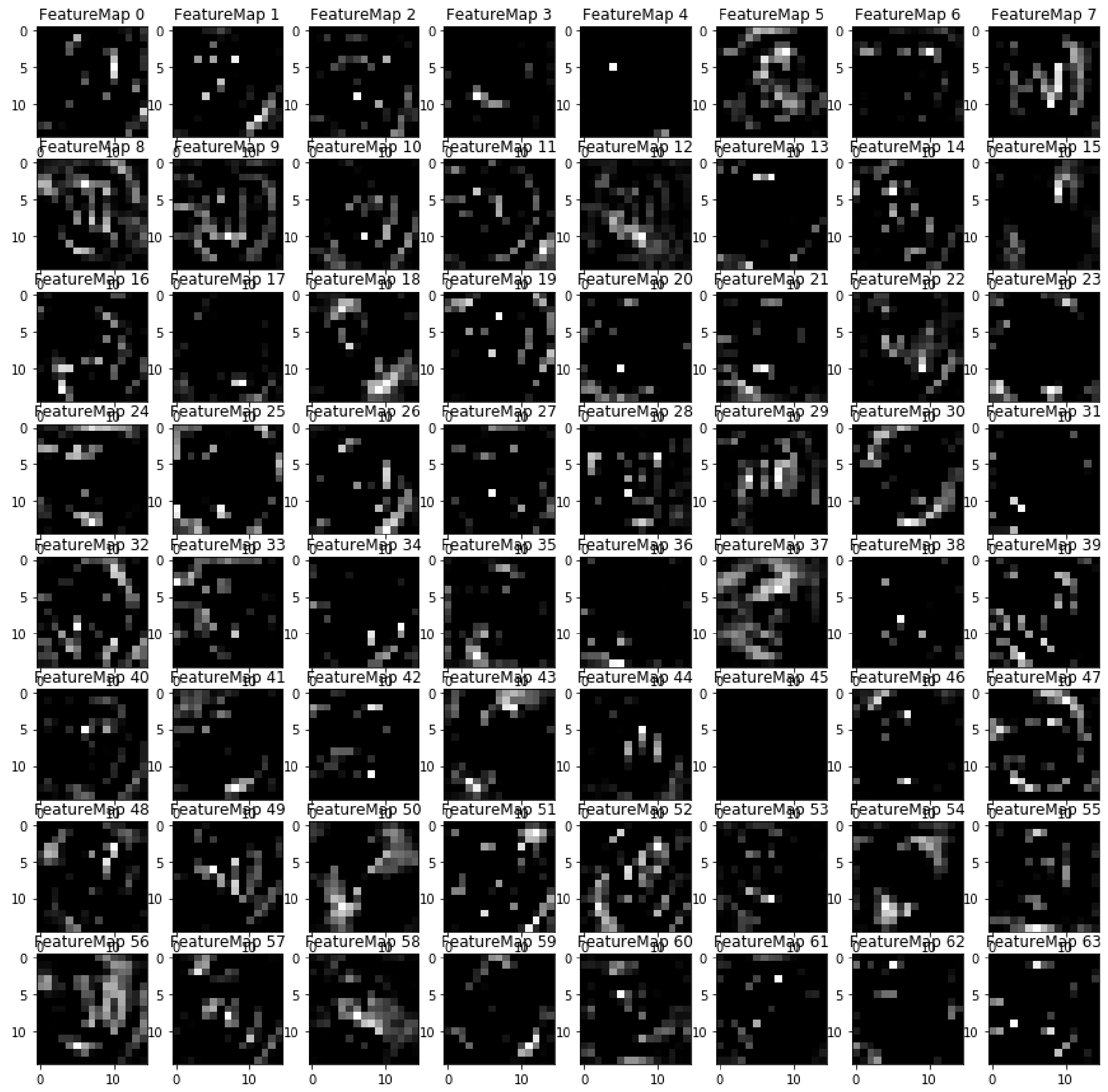
Convolution layer 1:



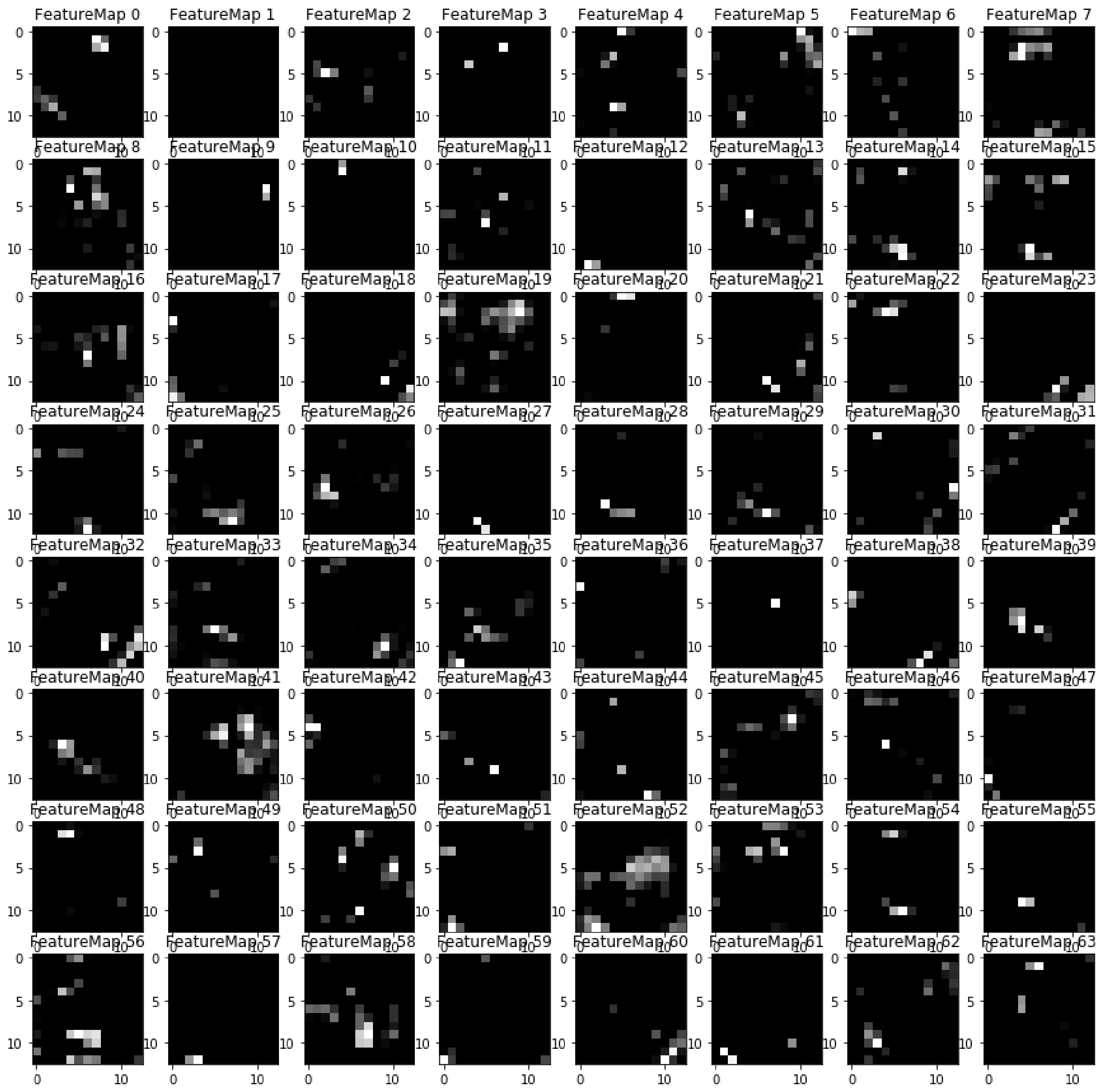
Convolution layer 2:



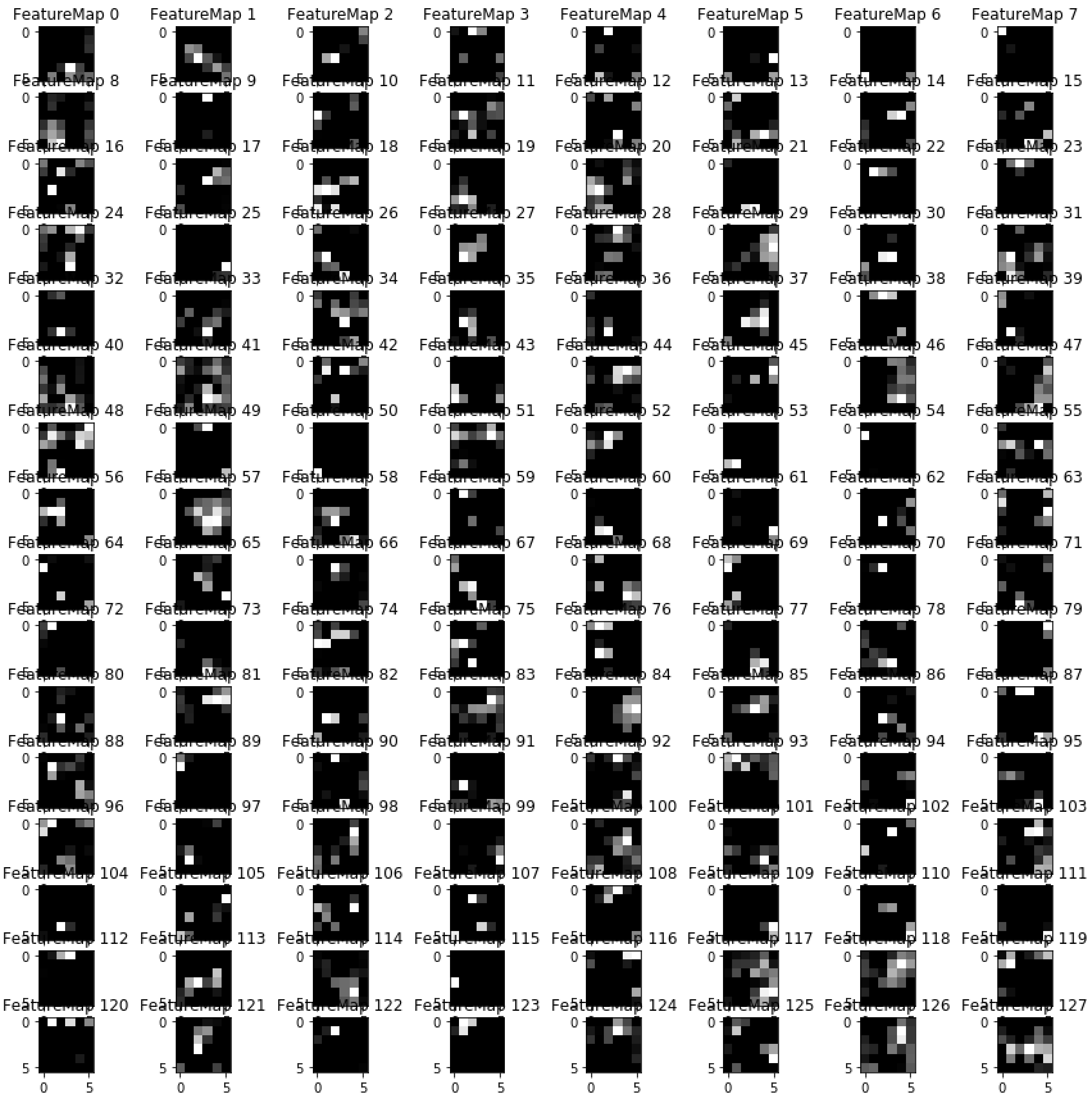
Convolution layer 3:



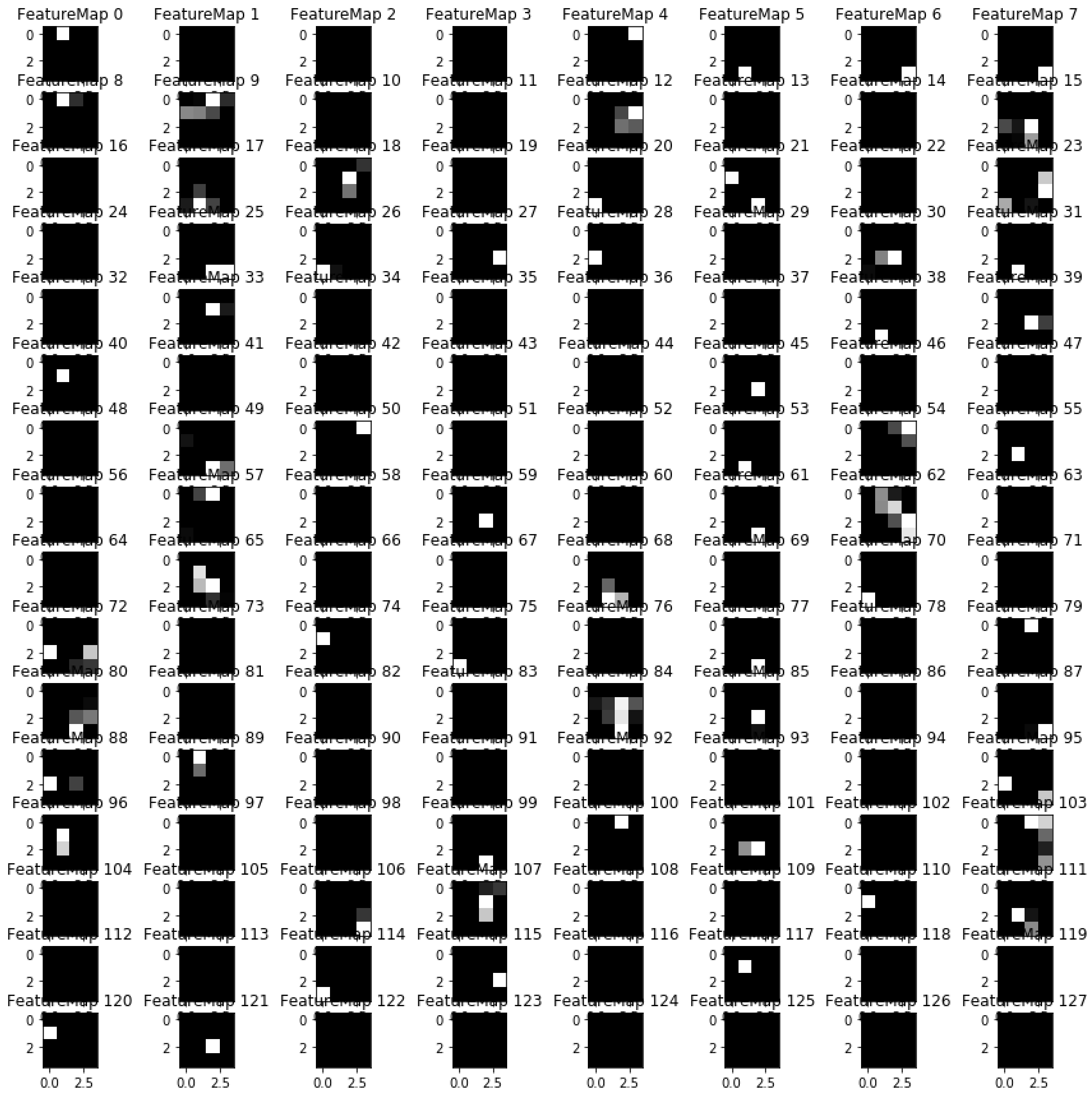
Convolution layer 4:



Convolution layer 5:



Convolution layer 6:


FeatureMap 0
FeatureMap 1
FeatureMap 2
FeatureMap 3
FeatureMap 4
FeatureMap 5
FeatureMap 6
FeatureMap 7
FeatureMap 8
FeatureMap 9
FeatureMap 10
FeatureMap 11
FeatureMap 12
FeatureMap 13
FeatureMap 14
FeatureMap 15
FeatureMap 120
FeatureMap 121
FeatureMap 122
FeatureMap 123
FeatureMap 124
FeatureMap 125
FeatureMap 126
FeatureMap 127