Traffic Sign Recognition

Writeup

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:

- (Complete) Load the data set (see below for links to the project data set)
- (Complete) Explore, summarize and visualize the data set
- (Complete) Design, train and test a model architecture
- (Complete) Use the model to make predictions on new images
- (Complete) Analyze the softmax probabilities of the new images
- (Complete) 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.

Rubric Points Summary

Category	Criteria	Status
Files Submitted	IPython Notebook with Code	Complete
Files Submitted	HTML Output of the Code	Complete
Files Submitted	A writeup report (this file)	In Progress
Dataset Exploration	Dataset Summary	Complete
Dataset Exploration	Exploratory Visualization	Complete
Design and Test a Model Architecture	Preprocessing	Complete
Design and Test a Model Architecture	Model Architecture	Complete
Design and Test a Model Architecture	Model Training	Complete
Design and Test a Model Architecture	Solution Approach	Complete
Test a Model on New Images	Acquiring New Images	Complete
Test a Model on New Images	Performance on New Images	Complete
Test a Model on New Images	Model Certainty - Softmax Probabilities	Complete

A detailed description of each criteria mentioned in the table above follows.

Files Submitted - IPython Notebook with Code

The Jupyter notebook containing my code is Traffic_Sign_Classifier.ipynb.

Files Submitted - HTML Output of the Code

The HTML file containing the output of the Jupyter Notebook is milestones/2018-08-12-11-47_complete.html.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one.

This file is the writeup. Here is a link to my project code. The HTML output of the code is here.

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.

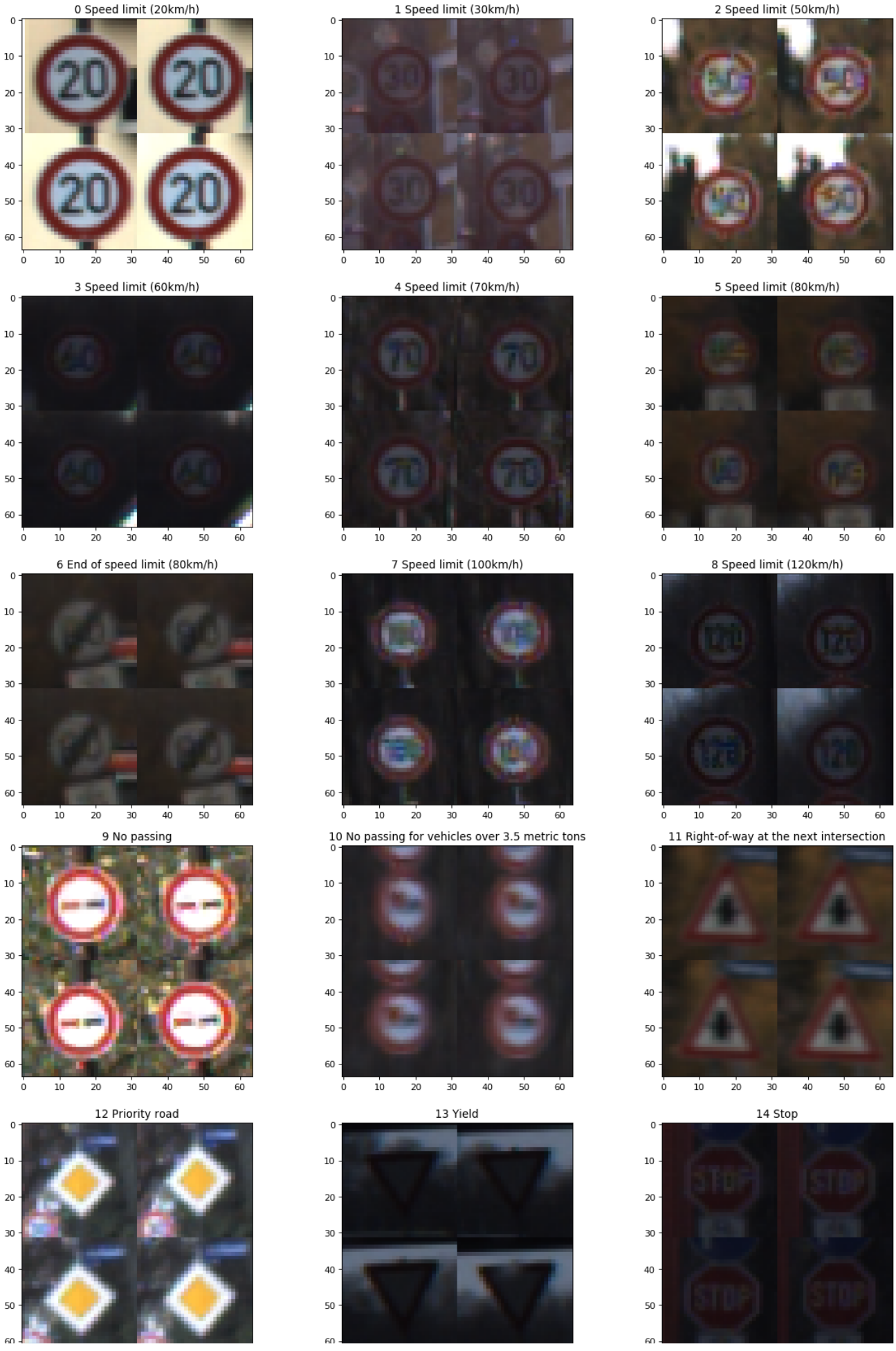
I used the numpy library to calculate summary statistics of the German traffic signs data set:

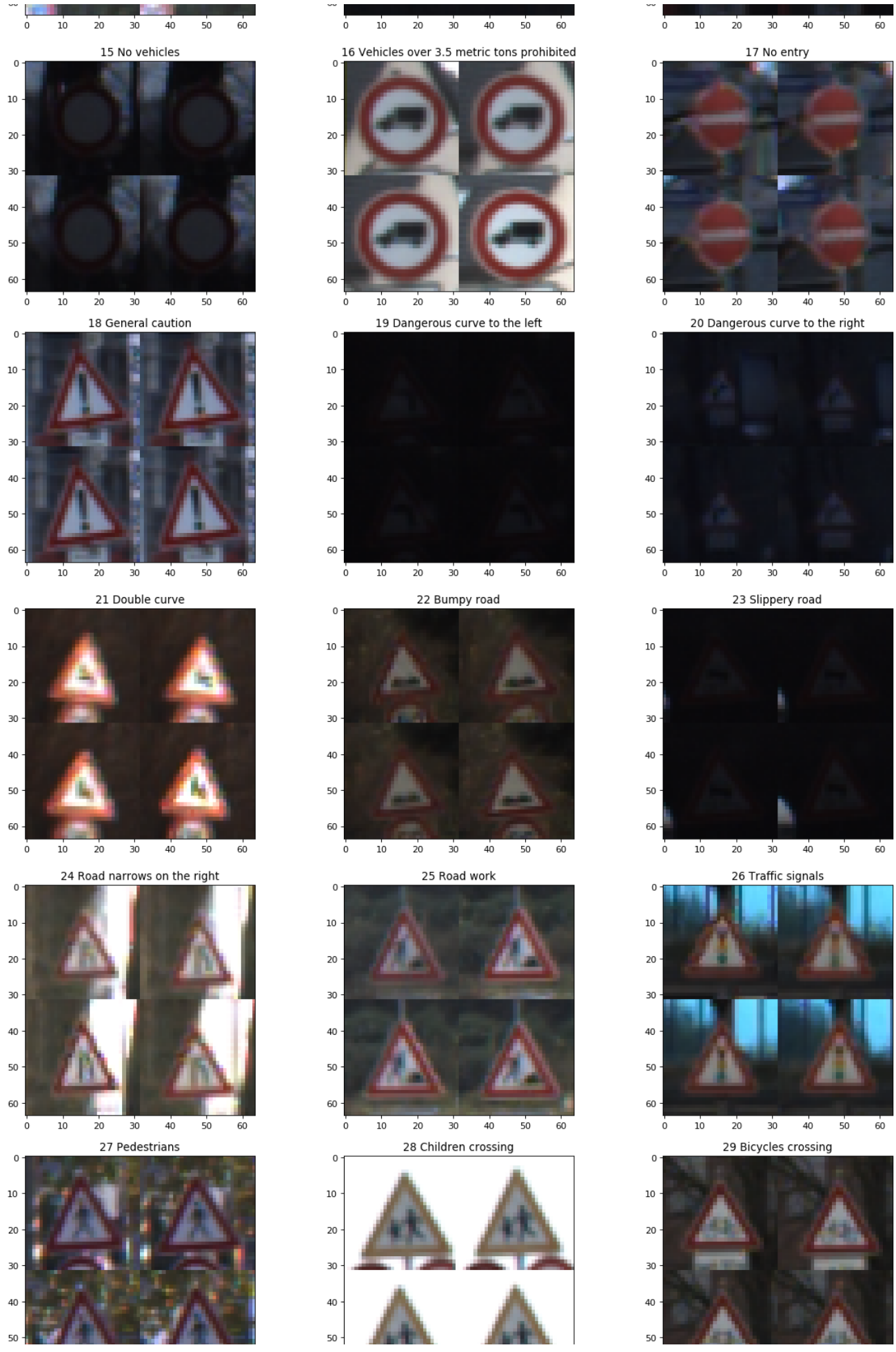
- The size of training set is 34799

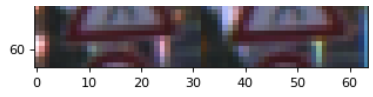
- The size of the validation set is 4410
- The size of test set is 12630
- The shape of a traffic sign image is 32x32x3
- The number of unique classes/labels in the data set is 43 (0 to 42)

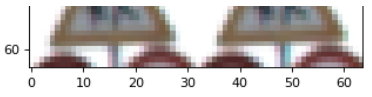
2. Include an exploratory visualization of the dataset.

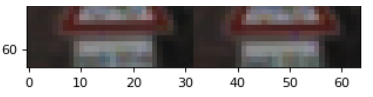
Below are a few images that represent my exporatory visualization of the dataset. I started with the first 4 occurrences of each class in the training data set. This gave me a feel for what each sign generally looked like. I also created graphs that indicate the counts of images in the train, validation, and test sets. I generated graphs of both unnormalized (the plain counts) and normalized (the counts divided by the greatest count). On the normalized graph you can see that the proporitions of the different classes in the 3 sets are quite similar.

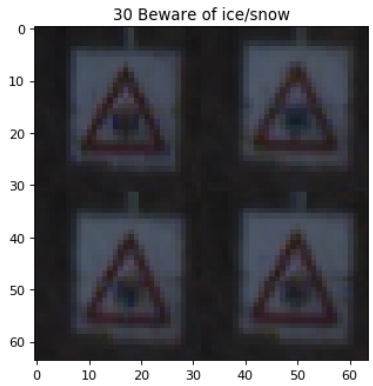

0 Speed limit (20km/h)
1 Speed limit (30km/h)
2 Speed limit (50km/h)
3 Speed limit (60km/h)
4 Speed limit (70km/h)
5 Speed limit (80km/h)
6 End of speed limit (80km/h)
7 Speed limit (100km/h)
8 Speed limit (120km/h)
9 No passing
10 No passing for vehicles over 3.5 metric tons
11 Right-of-way at the next intersection
12 Priority road
13 Yield
14 Stop

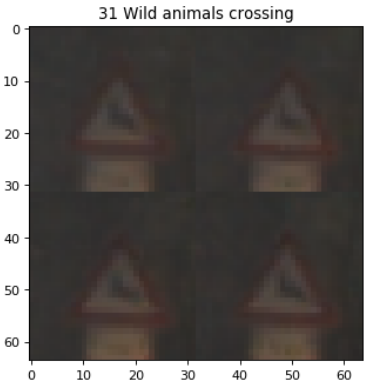

15 No vehicles
16 Vehicles over 3.5 metric tons prohibited
17 No entry
18 General caution
19 Dangerous curve to the left
20 Dangerous curve to the right
21 Double curve
22 Bumpy road
23 Slippery road
24 Road narrows on the right
25 Road work
26 Traffic signals
27 Pedestrians
28 Children crossing
29 Bicycles crossing

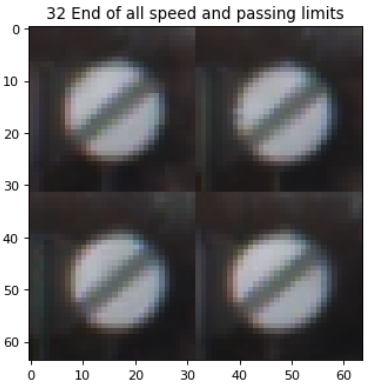


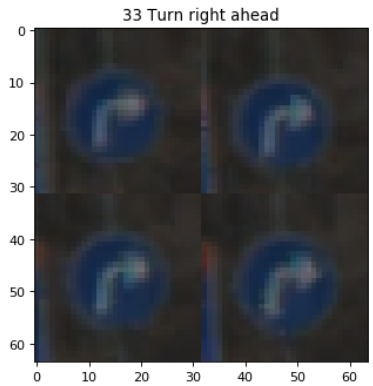


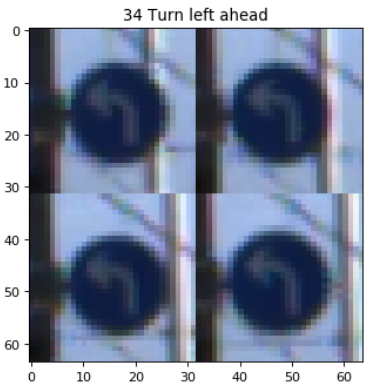


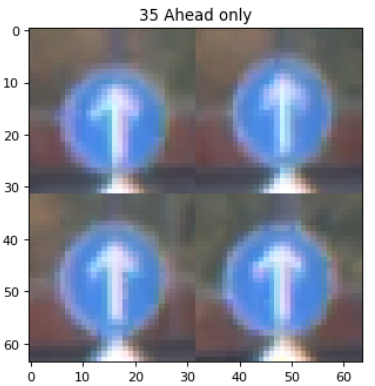

30 Beware of ice/snow


31 Wild animals crossing

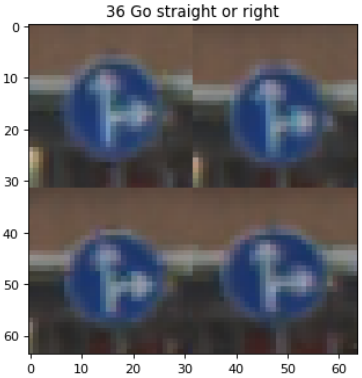

32 End of all speed and passing limits


33 Turn right ahead

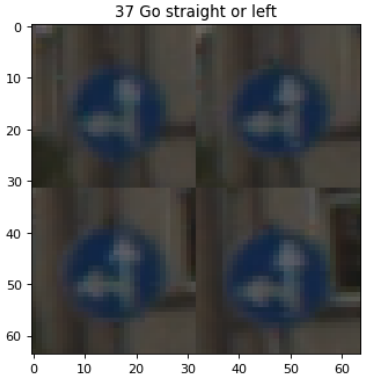

34 Turn left ahead


35 Ahead only

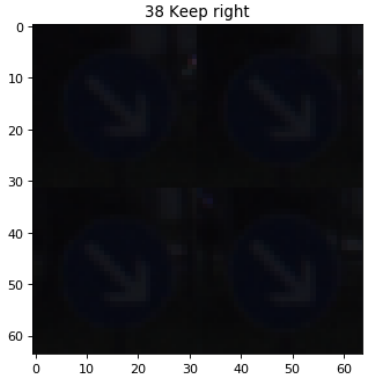
36 Go straight or right



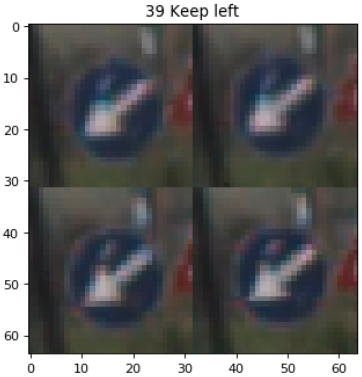
37 Go straight or left



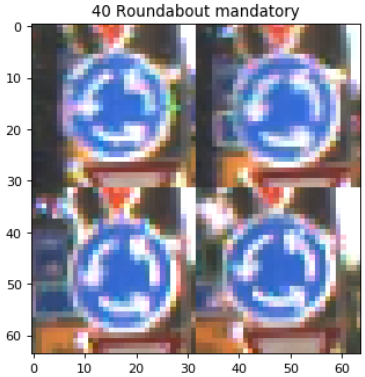
38 Keep right



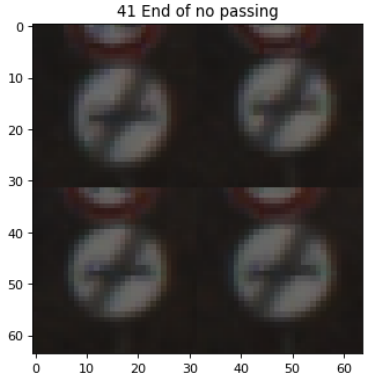
39 Keep left



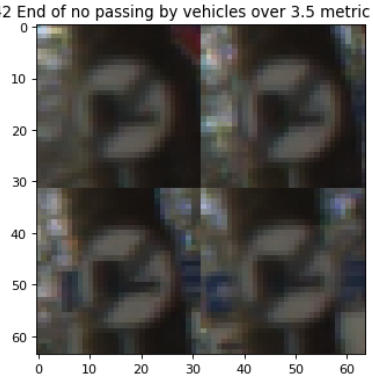
40 Roundabout mandatory

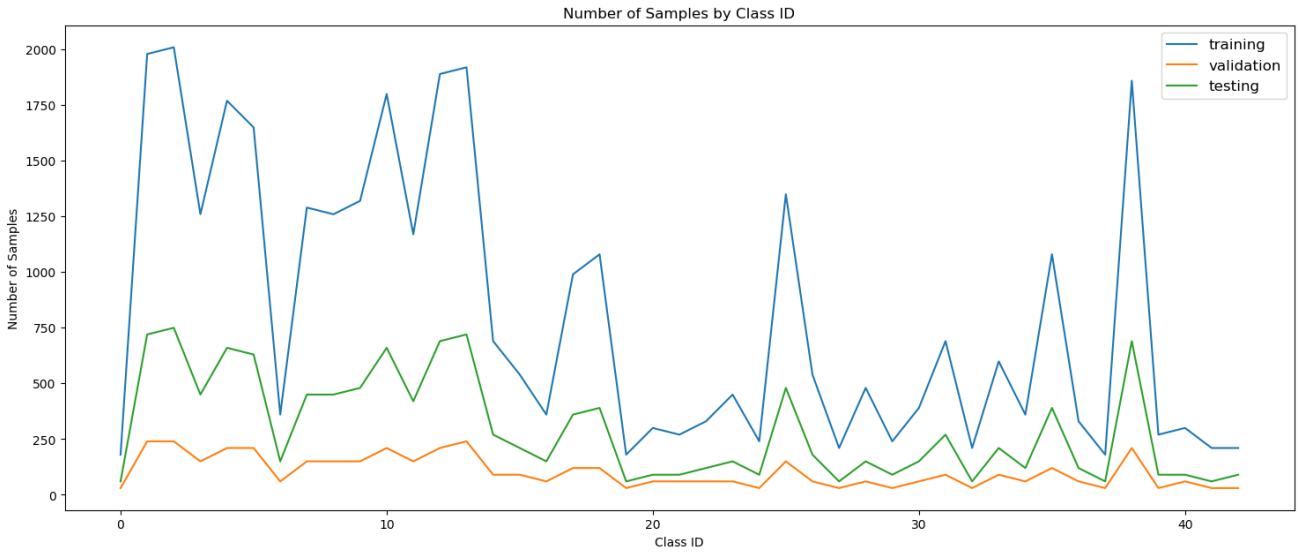


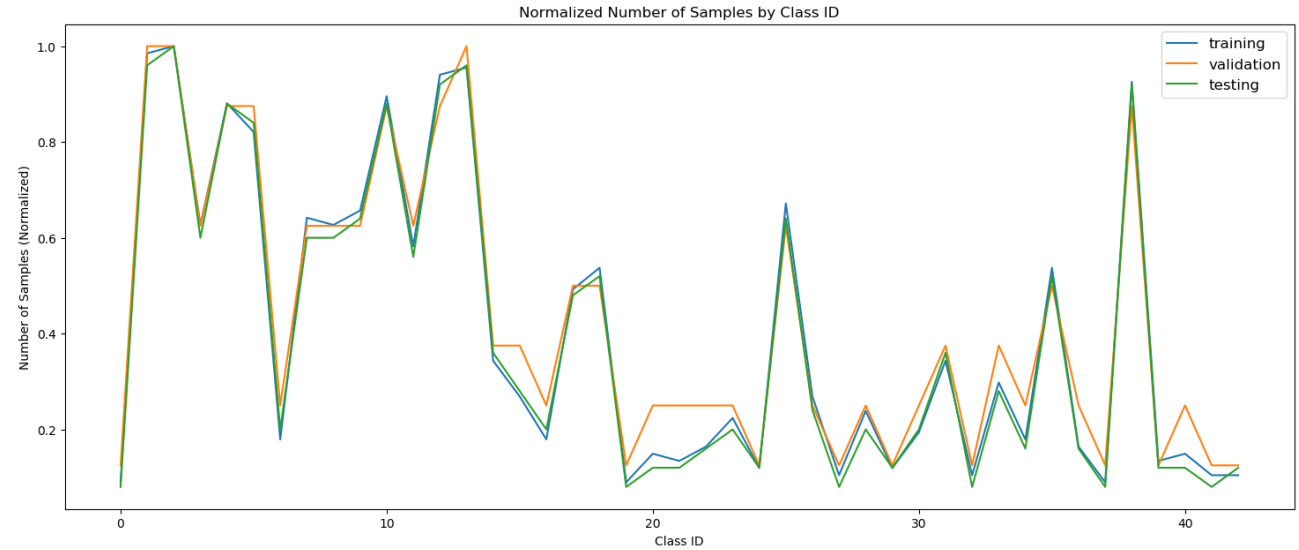
41 End of no passing



42 End of no passing by vehicles over 3.5 metric tons







Design and Test a Model Architecture

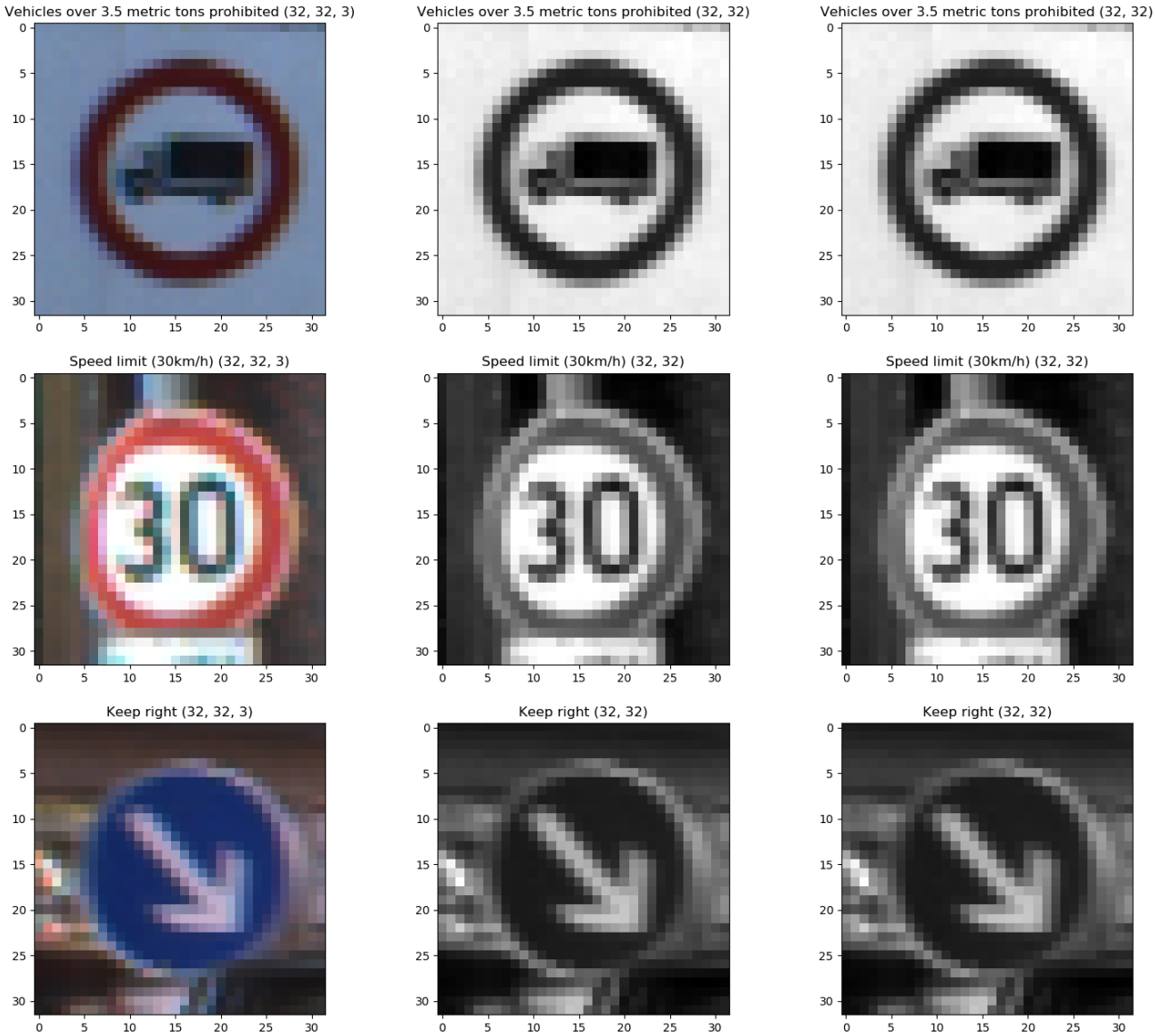
1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc. (OPTIONAL: As described 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 inspection of the various classes in the dataset indicated that the structure of the image should be sufficient to differentiate between classes. In addition, I assumed that using less data as input would result in faster training and inference times, since fewer parameters need to be adjusted by the optimizer. The time required to finalize the architecture and meet the requirements is inversely proportional to the training and inference times of the network assuming that the information lost did not significantly detract from network performance.

To convert the images to greyscale, I ended up using 29% of the red component, 58.7% of the green component, and 11.4% of the blue component. Initially I simply tried averaging all of them but got poor results - the images didn't look clear. The matlab documentation of the matlab function rgb2gray at https://www.mathworks.com/help/matlab/ref/rgb2gray.html suggests the above formula (which is apparently related to 'luminance' in some way).

As a last step, I normalized the image data because I expected that this would make the process of optimization easier.

Here is an example of a traffic sign image before pre-processing, after grayscaling, and after normalization:



Due to how matplotlib plots images by default, the normalized image appears the same as the greyscale image.

I did not generate any additional data using brightness, contrast, rotation or shearing effects.

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:

Layer	Description
Input	32x32x1 Greyscale image
Convolution 5x5	5x5 kernel, 1x1 stride, valid padding, outputs 28x28x12
RELU	
Max pooling 2x2	2x2 kernel, 2x2 stride, valid padding, outputs 14x14x12
Convolution 5x5	5x5 kernel, 1x1 stride, valid padding, outputs 10x10x32
RELU	
Max pooling 2x2	2x2 kernel, 2x2 stride, valid padding, outputs 5x5x32
Flattened	output size is 5 * 5 * 32=800
Fully Connected	input is 800, output is 200
RELU	
Fully Connected	input is 200, output is 400
RELU	**
Fully Connected	input is 400, output is 600
Fully Connected	input is 400, output is 43 (logits output)**
Softmax	combined with cross entropy for training, but used directly for inference

**Dropout is applied after layers marked with a double asterisk in their description.

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.

To train the model, I used the adam optimizer which acted on a cost function defined as the mean of the cross entropy (with the one hot encoding of the training labels) of the softmax of the logits mentioned in the above layer. I used a learning rate of 0.001 after experimenting with values between 0.1 and 1e-8. I actually found that a smaller batch size produced equal or better results (despite what I learned in the lessons), and settled on a value fo 128. I am still unsure as to why a smaller batch size seemed to be producing better results. I trained for 200 epochs, although I experimented epoch counts from 10 to 2000. 200 epochs were completed in minutes on an NVIDIA GTX 1060.

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 were:

- training set accuracy of 0.997442 (99.74%)
- validation set accuracy of 0.957596 (95.76%)
- test set accuracy of 0.945447 (94.54%)

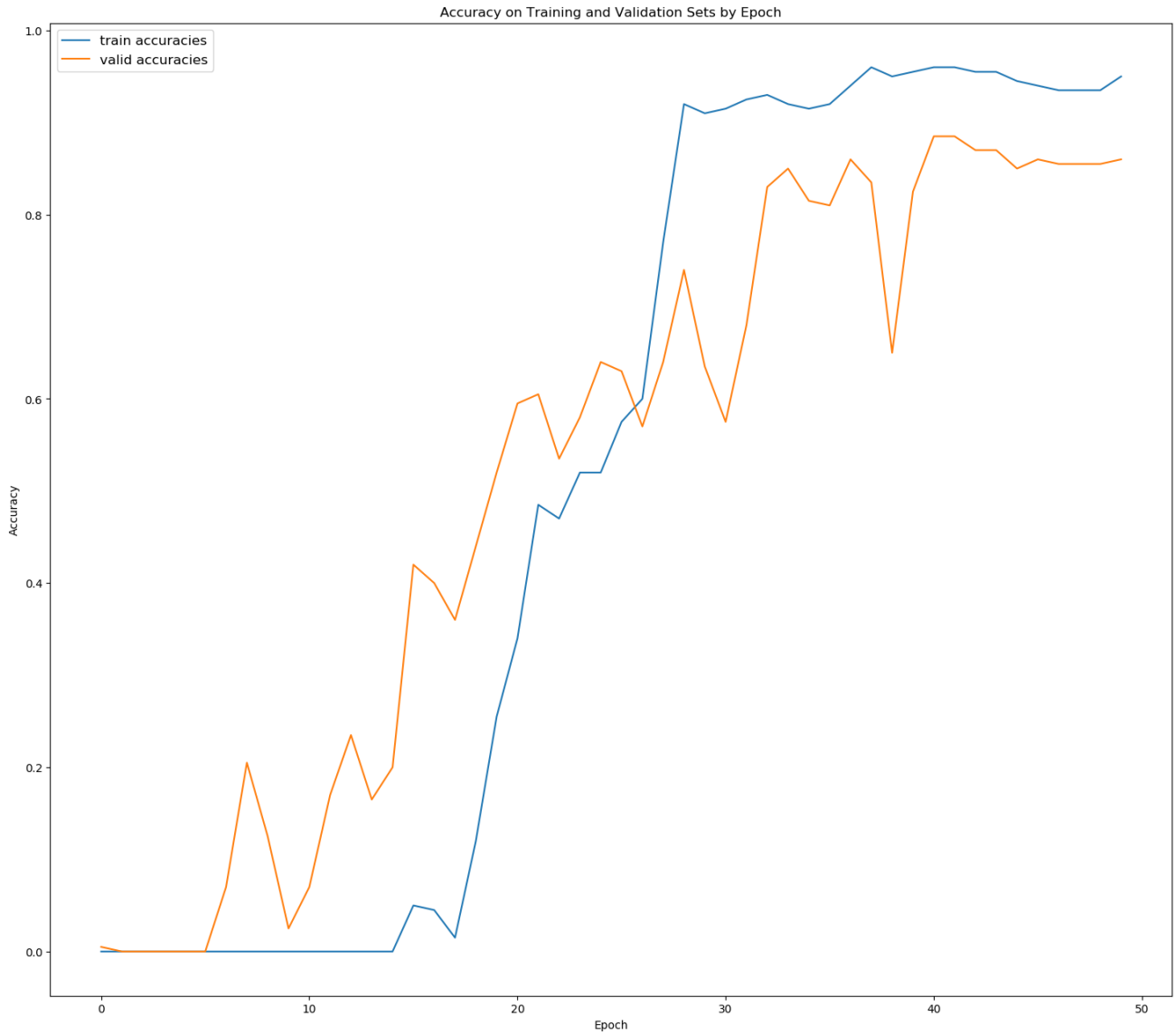
I chose an iterative approach. I answer the following supplied questions below about this approach. I did not choose a well known architecture to my knowledge (I have not read all literature on the subject), although at times I tried one idea from a well known paper (see details below).

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

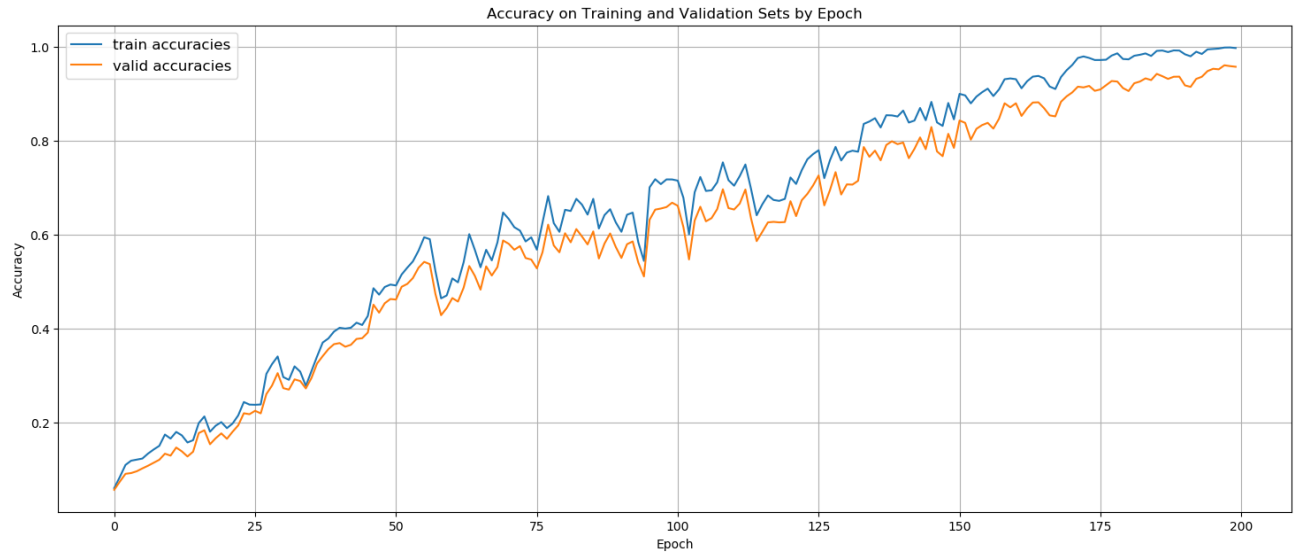
I started with the architecture used to classify the MNIST data from the lab.

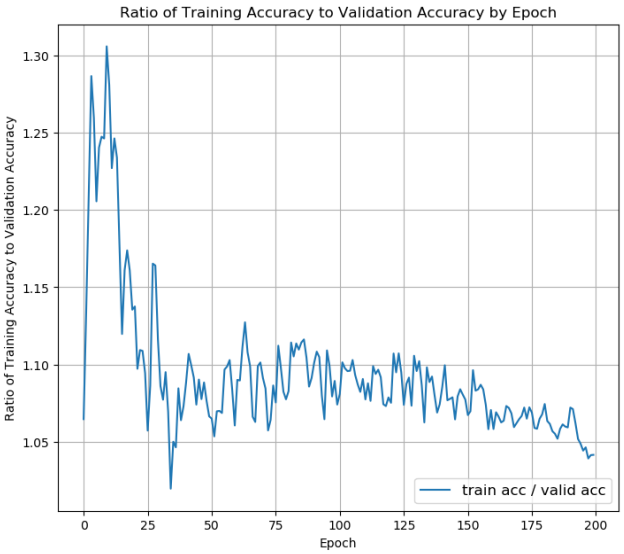
What were some problems with the initial architecture?

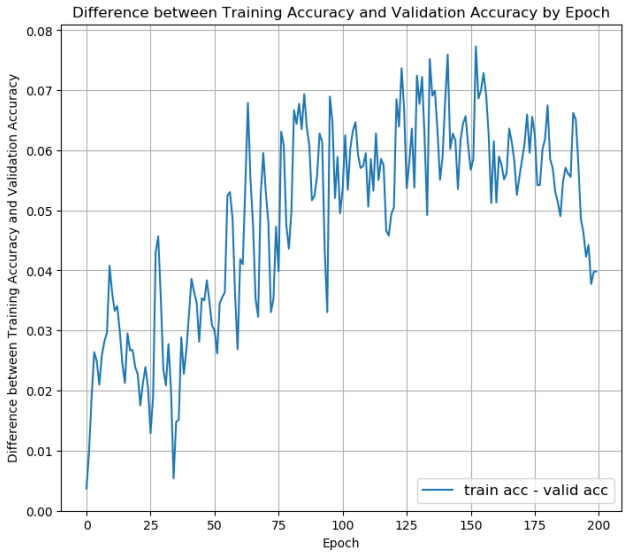
The main problem with the initial architecture was that it did not generalize well to the validation data - the training accuracy was close to or exactly 100% but the validation accuracy was stuck at about 85%.



As I modified the architecture and spent more time, I changed my graphs to also show the ratio and difference between the training and validation accuracy. Through these graphs, I could observe whether or not the network was underfitting or overfitting the training data. If the training data accuracy failed to reach 100% (or close to it), the network was underfitting. I never saw a meaningful case where the validation accuracy exceeded the training accuracy in the long term. Assuming the network does not underfit the training data, then I was evaluating whether or not the network was overfitting by looking at the gap between the training and validation accuracies towards the end of the training period. A larger gap indicates overfitting, while no gap (the ideal case) indicates no overfitting. It seems almost impossible to not have at least some 'overfitting' (otherwise the network would be completely generic). Note that the final architecture does a better job of generalizing/not overfitting the training data.







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 overfitting; a low accuracy on both sets indicates underfitting.

As I explained in my answer to the previous question, the main problem was overfitting at first (the network was not generalizing well). In the end dropout proved to be very effective at eliminating this error.

As I went, I made some observations about my progress, which are in the file milestones/notes.txt. I have also copied and pasted the notes below:

2018-07-27-23-20

- by increasing the number of epochs for training, the neural network can be made to closely fit the training data. however, there is a consistent gap between the validation data and the training data.
- this gap seems to represent information that is not 'learned' by fitting to the training data. in other words, this gap provides some information about the generality of the network. an ideally general network would have validation accuracy identical to the training data.
- the gap stays relatively constant after epoch 75 at around 10%.

- even if I was to continue training until both randomly became high, this could simply mean that I have fit the network to both the training and validation data. Essentially by avoiding simply setting the number of epochs to 1000 or some huge number, I am using the 'Early Termination' principle. It seems that now something like dropout, L2 Regularization, or something similar is necessary to implement in order to create a network that generalizes.

2018-07-28-00-35

- I increased the depth of the network by using 3 more hidden layers, and the training and validation accuracies increase much faster (requires fewer epochs) and approach 100% much closer (400 epochs reaches 3 9s, 341 epochs matches 400 epochs to 3 decimal places). The next step is to try dropout.

2018-07-28-14-37

- When dropout is used, the generality of the network seems much improved. Also, even when the training accuracy is 100%, the validation accuracy continues to improve. There are spikes every 50 or so epochs in which the training accuracy drops by less than 15%, then recovers almost immediately.
- now I am going to try training for 1000 epochs, and see how good the network will become with different batch sizes, learning rates, and possibly different classifier sizes.

2018-08-11-21-40

- Analysis of specific images from the web indicates that when the probability of the top output class is very close to 1 (maybe greater than 0.95 or so) the classifications are very accurate, and the next possibilities also seem to make sense. Otherwise, the prediction is wrong.
- Looking at the ones that were classified incorrectly and visualizing the corresponding feature layer conv1, it actually looks like a "3" 0 to the human eye, indicating a network architecture problem. Also signs that have distinct shapes, such as 'priority road' or 'no entry' with large features are easily recognizable. It seems unlikely that this is a resolution problem because I can identify the signs - it is therefore an architecture problem, under the assumption that CNNs can identify features of arbitrary size and shape.

Other critical notes include various attempts I made along the way that are not explicitly documented in my above notes. They were:

1. I considered using an idea from the published baseline model in which the fully connected layers use the concatenation of the flattened first and second convolutional layers. Testing indicated that this actually worsened performance, although this might have had to do with some other aspect of my network modifications at the time. For example, perhaps I should have spent more time adjusting the learning rate or convolution sizes or classifier fully connected layer sizes in order to enable use of the additional information.
2. I made use of the principle of early termination after training up to 2000 epochs or more - I found that once the training data reached 100% and remained there for some time, the validation accuracy was only fluctuating a small amount and did not seem to be generally improving.
3. I noticed that every so often the validation and training accuracies would spike downwards after they reached 'steady state'. I believe this might be a characteristic of the Adam optimizer, although I have not yet confirmed this.
4. My strategy was to produce a validation accuracy approximately 2-3% higher than the rubric requirement, as I expected the test accuracy to be even lower, hopefully by less than 2%. Indeed this was the case in the end; the validation set proved to be a good way to improve the generality of the network without using the test set.

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

The learning rate, number of epochs, batch size, kernel sizes for the 2d convolutions, depth of the 2d convolution layers (i.e. number of convolution filters) and fully connected layer sizes were adjusted.

I adjusted the kernel sizes for the 2d convolution completely based on trial and error, as was the learning rate and batch size. The conv2d kernel sizes and batch size seemed to improve in the opposite directions to my intuition - I thought that smaller convolutional kernels would allow finer detail and therefore better performance on certain classes that rely on smaller structural features, but this did not seem to be the case. It might be that other aspects of the network were not properly using the addition information generated. In addition, I thought that a larger batch size would provide better performance, but a smaller batch size (ex 128) did better than a larger batch size(ex 1024). I only tested the effects of changing parameters such as batch size intermittently - I think it might be worthwhile to write code that tests the effects of batch size, learning rate, and other things and report the results each time - I might have missed out on higher accuracies since I was varying these things manually.

The sizes of the fully connected layers were chosen in a mannel that I hoped would improve the non-linearity of the classifier and enable faster fitting of the data (hopefully not at the cost of overfitting). The number of epochs was chosen according to the principle of early termination after experiments with a very large number of epochs.

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?

One choice I made that I think worked well was using quite a deep (4 layers) classifier after the 2 convolution/relu/max pooling stages. I believe that this contributed quite a bit to the speed with which the network was able to fit the training data as it should have enabled more non-linear data processing. I only used activation functions on the first two layers - I am intersted in experimenting more with appending the relu functions to different combinations of the layers. My current architecture uses activation functions on the outputs of the first two only. While the training speed certainly improved, I was also wary of the larger network being more prone to overfitting the data, although this did not seem to be an issue.

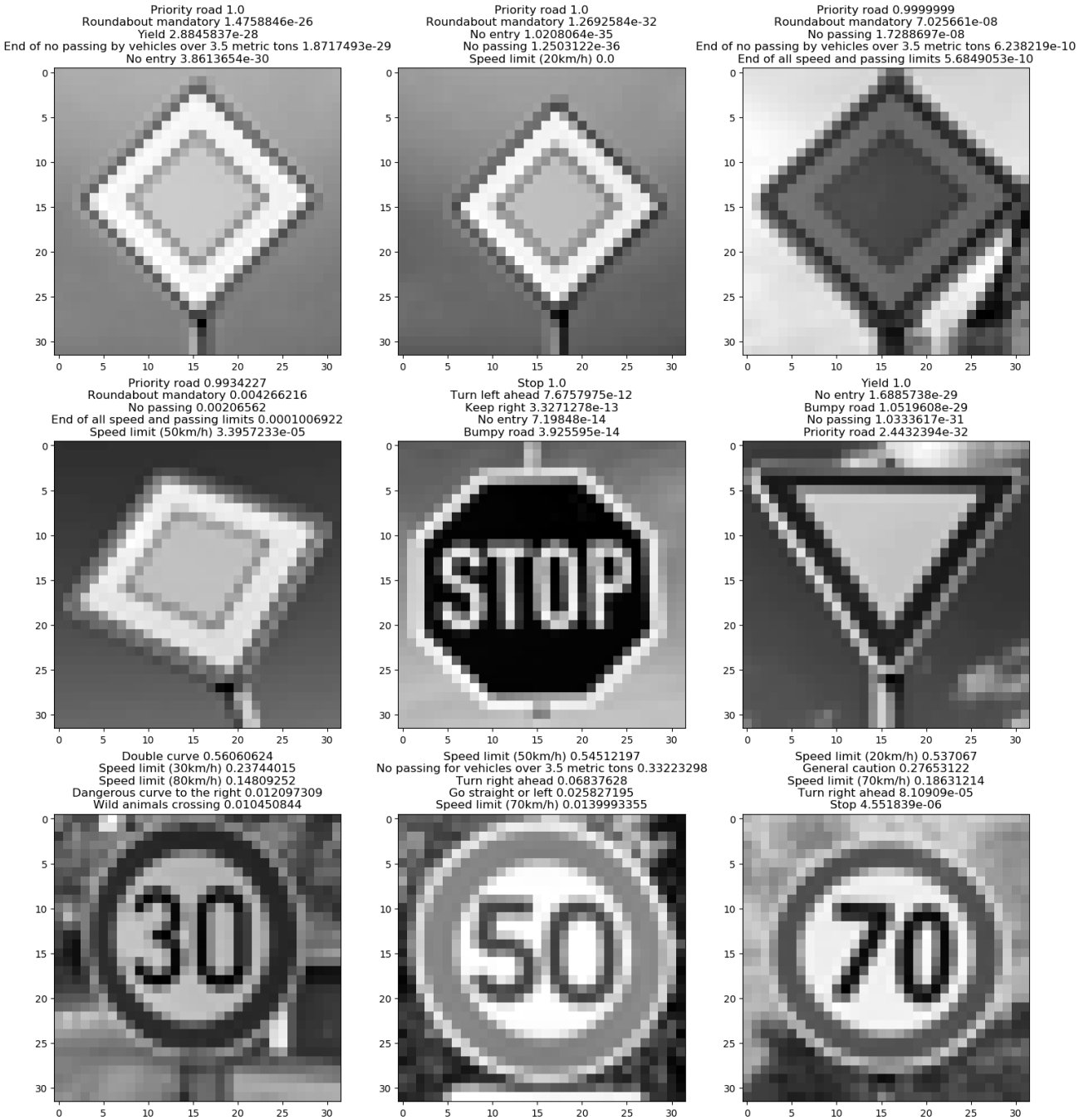
Dropout was an important part of improving the performance of the network on the validation set - after applying dropout I noticed the gap between the training accuracy (at this point at 100% after a short training time) decreased significantly.

On final modification I made was to the number of convolutional filters in each convolutional layer - I used 12 in the first conv2d layer and 32 in the second conv2d layer (increased from 6 and 16 respectively). This really seemed to improve the validation performance and put the network accuracy on the validation set reliably above 95%. I believe that the max pooling layers following the conv2d operations were important for generating translation invariance, and the combination of the two produced scale invariance. I tested this manually with some of the images I pulled from the web (see the next section).

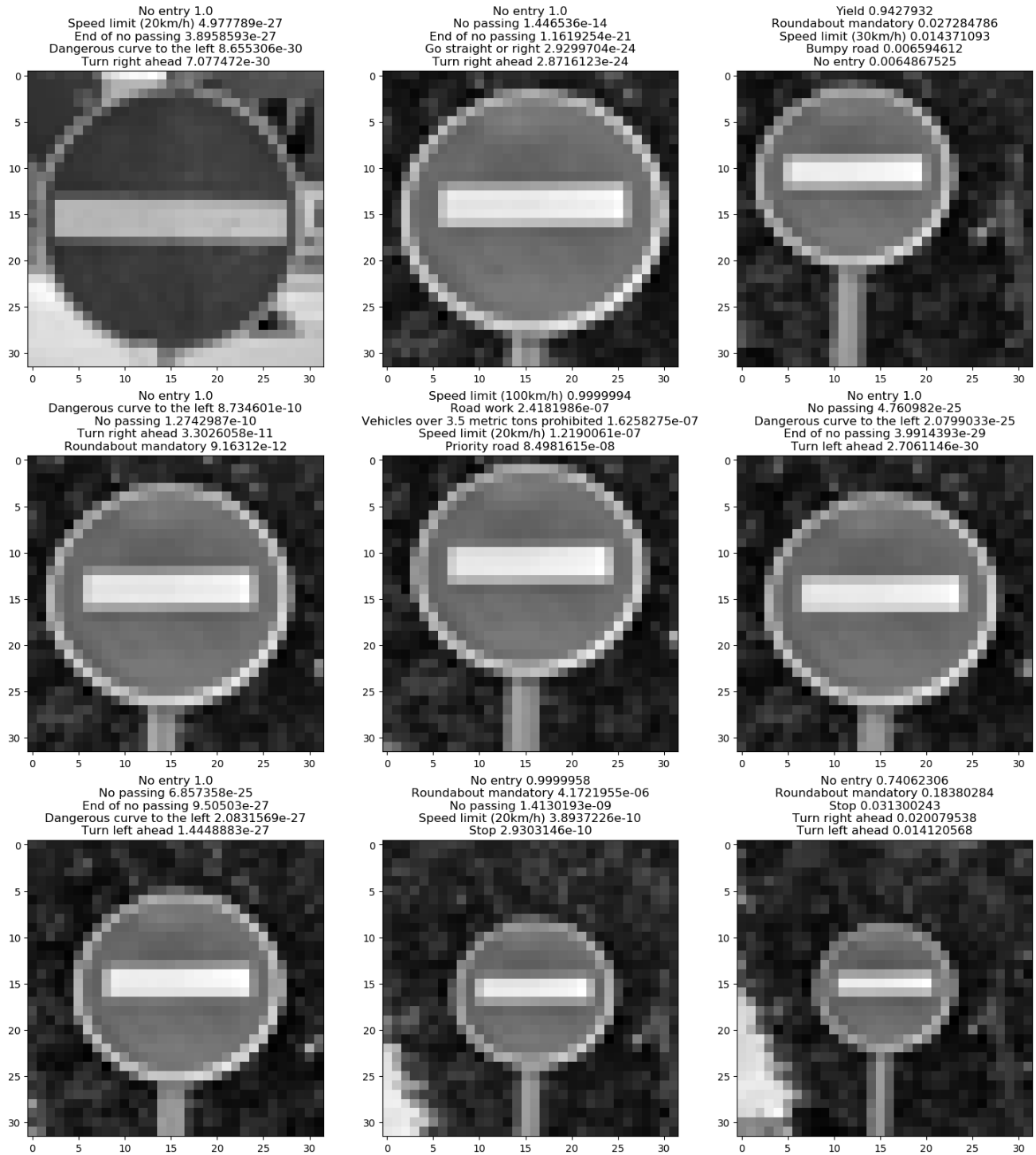
Test a Model on New Images

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.

I ended up using more than 5 signs so I could investigate certain properties of the network. Here are nine German traffic signs that I found on the web:



In addition, I looked at a single type of sign, no entry, at various scales and translations (although always in a 32x32 pixel image). The original no entry sign image and my manually translated and scaled versions are shown below. The top left image is one no-entry sign image, the rest are translations and scalings of the same image.



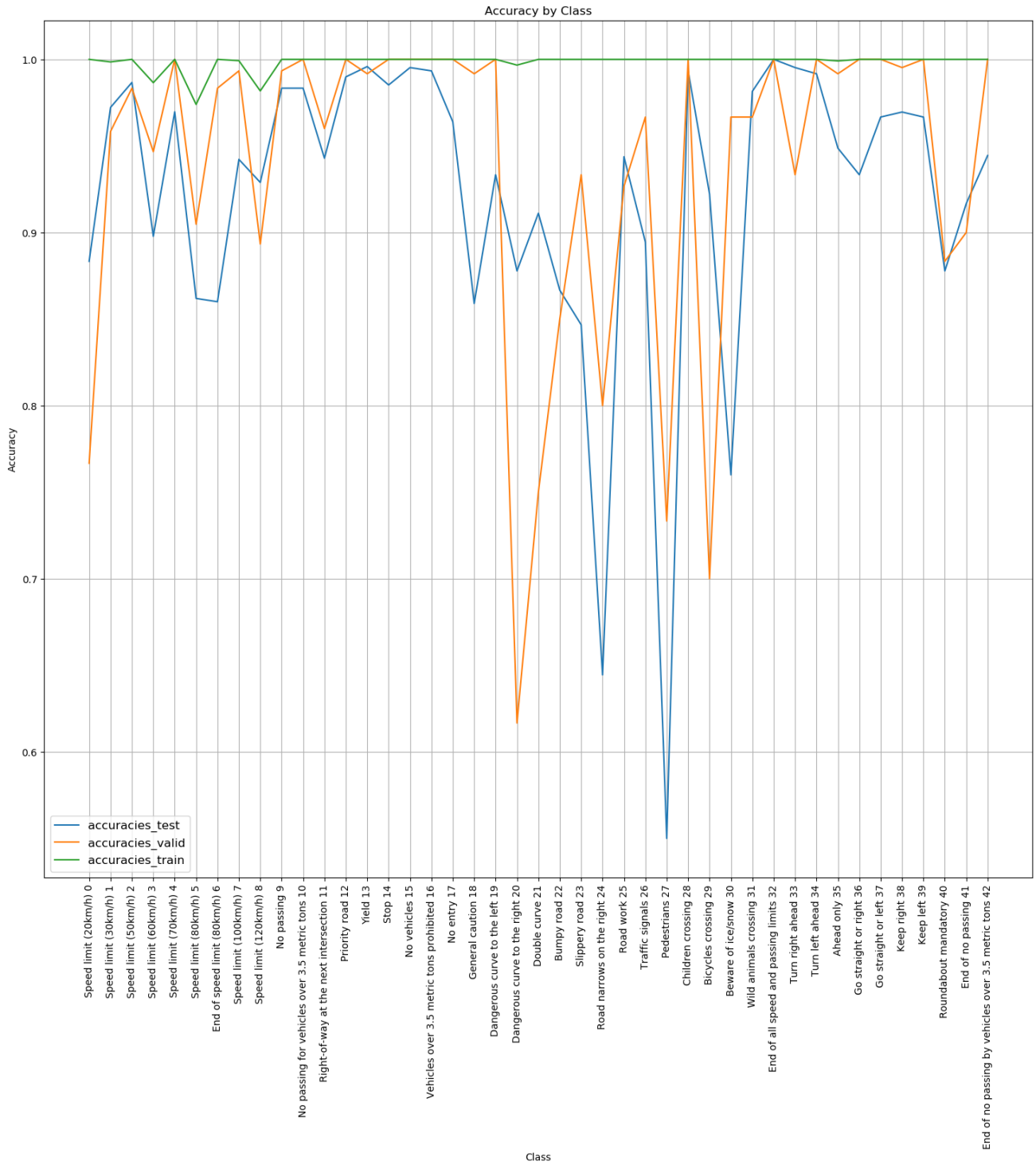
It seems that the network is easily able to classify images of signs with distinct shapes. For example, the concentric diamond shape of the priority road, or the octagon of the stop sign, or the triangle of the yield sign. On the other hand, there is a set of signs that are made up of a detailed structure within two concentric circles. For example, the speed limit signs all have the same general outer shape, but the smaller details in the center determine their class. The network must be able to extract features that are only 5-10 pixels wide. I

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
Priority Road	Priority Road
Stop	Stop
Yield	Yield
Speed Limit (30km/h)	Double curve
Speed Limit (50km/h)	Speed Limit (50km/h)
Speed Limit (70km/h)	Speed Limit (20km/h)
No Entry	No Entry

The model correctly classified 5/7 of the unique signs tested (accuracy 71.4%). The accuracy on the test set was 94.5%. However, it seems that the accuracy varies signficantly by sign type. Below is a graph of the accuracy of the network by class for the training, validation, and test sets.



Notice that the yield, stop, priority road, and a few others have accuracy on all 3 sets of >95% (approximate), while for other classes it is clear that the network has overfit the training data (ex 27/pedestrians class). The speed limit signs are slightly worse but not terrible - still greater than 90%. It is also possible that the images I chose from the web for the 30 kph and 70 kph signs have certain differences compared to the training/validation/test sets, or they are part of the 5-10% that is misclassified even in the dataset. Overall it is clear that an overall high score on the entire test set (or validation set) does not necessarily indicate a strong network, especially when the network has a relatively small number images of a few classes.

One advantage to looking at the data class by class is that it allows the observer to make conclusions about which features the network might be struggling to 'pick up'. For example, I can clearly see that while my network meets the accuracy requirements for this project, it clearly would not work in practice - I would need to modify the architecture or training process so that the details of the pedestrian sign are also available.

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 under the heading 'Predict the Sign Type for Each Image and Analyze Performance' in my Jupyter notebook.

For the first image, the model is relatively sure that this is a stop sign (probability of 0.6), and the image does contain a stop sign. The top five soft max probabilities were

The top 5 softmax probabilities for each of the signs I pulled from the internet are shown in each figure title in the pictures in the previous section of this report. To match the requested format, I have also summarized those for the unique signs in the 7 tables below. I have rounded all probabilities to 3 digits (the ones that are exactly 1 or 0 are so because of rounding).

First Figure, Top Left

Correct Class: Priority Road

Probability	Prediction
1.000	Priority Road
0.000	Roundabout mandatory
0.000	Yield
0.000	End of no passign by vehicles over 3.5 metric tons
0.000	No Entry

First Figure, Middle Center

Correct Class: Stop

Probability	Prediction
1.000	Stop
0.000	Turn left ahead
0.000	Keep right
0.000	No entry
0.000	Bumpy Road

First Figure, Middle Left

Correct Class: Yield

Probability	Prediction
1.000	Yield
0.000	No Entry
0.000	Bumpy Road
0.000	No Passing
0.000	Priority road

First Figure, Bottom Left

Correct Class: Speed Limit (30km/h)

Probability	Prediction
0.561	Double curve
0.237	Speed Limit (30km/h)
0.148	Speed Limit (80km/h)
0.012	Dangerous curve to the right
0.010	Wild animals crossing

An interesting note in this case is that the network seems to think that "30" and "80" are quite similar. A 3 might be seen as the 'right half' of an 8.

First Figure, Bottom Center

Correct Class: Speed Limit (50km/h)

Probability	Prediction
0.545	Speed Limit (50km/h)
0.332	No passing for vehicles over 3.5 metric tons
0.068	Turn right ahead
0.026	Go straight or left
0.014	Speed Limit (70km/h)

First Figure, Bottom Right

Correct Class: Speed Limit (70km/h)

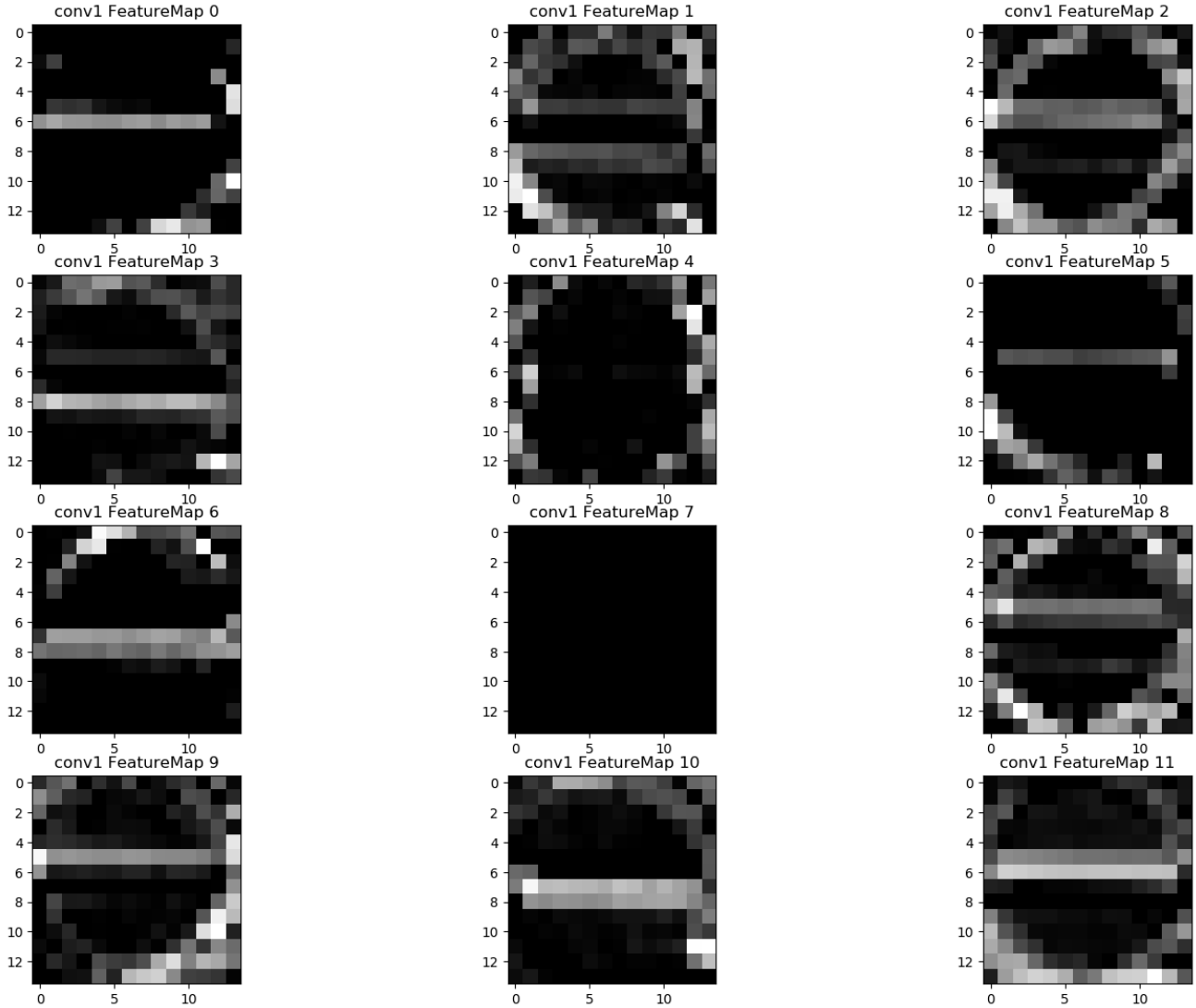
Probability	Prediction
0.537	Speed Limit (20km/h)
0.277	General caution
0.186	Speed Limit (70km/h)
0.000	Turn right ahead

| 0.000 | Stop

A 7 is like a 2 without the bottom horizontal stroke - it makes sense that 20 and 70 are both on the list.

(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?



It appears that the network uses the overall circular shape and the horizontal bar in the middle of the image to classify the no entry sign. It seems that the individual edges of the horizontal bar (ex. in maps 0, 3, 5, 8, and 9) are used in addition to the bar's 'area' itself (ex. in maps 6, 10, and 11).

An interesting conclusion of this visualization is that if there were two signs with identical shape but different rotations that had different meaning, one would have to ensure that in the data used that each labelled image had the correct orientation - in other words rotating the training data to increased the number of training images would actually lead to disasterous results. This doesn't seem to be the case for any traffic signs, however.