Traffic Sign Recognition

Writeup Template

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:

- 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.

To make it easier to evaluate my submission I have listed below the rubrics requirements and its index.

https://review.udacity.com/#!/rubrics/481/view

Traffic Sign Classification

Files Submitted

CRITERIA	MEETS SPECIFICATIONS
Submission Files	The project submission includes all required files. Files include the notebook, html version and this readme file

Dataset Exploration

CRITERIA	MEETS SPECIFICATIONS
Dataset Summary	The submission includes a basic summary of the data set. Section #1
Exploratory Visualization	The submission includes an exploratory visualization on the dataset. Section #2

Design and Test a Model Architecture

CRITERIA	MEETS SPECIFICATIONS
Preprocessing	The submission describes the preprocessing techniques used and why these techniques were chosen. Section #3. Most of preprocessing was turned off or not implemented because it was not improving the accuracy
Model Architecture	The submission provides details of the characteristics and qualities of the architecture, such as the type of model used, the number of layers, the size of each layer. Visualizations emphasizing particular qualities of the architecture are encouraged. Section #5
Model Training	The submission describes how the model was trained by discussing what optimizer was used, batch size, number of epochs and values for hyperparameters.

CRITERIA	MEETS SPECIFICATIONS
	Section #6
Solution Design	The project thoroughly discusses the approach taken for deriving and designing a model architecture fit for solving the problem given. Section #7

Test a Model on New Images

CRITERIA	MEETS SPECIFICATIONS
Acquiring New Images	The submission includes five new German Traffic signs found on the web, and the images are visualized. Discussion is made as to any particular qualities of the images or traffic signs in the images that may be of interest, such as whether they would be difficult for the model to classify. Section #8
Performance on New Images	The submission documents the performance of the model when tested on the captured images. The performance on the new images is compared to the accuracy results of the test set. Section #9
Model Certainty - Softmax Probabilities	The top five softmax probabilities of the predictions on the captured images are outputted. The submission discusses how certain or uncertain the model is of its predictions. Section #10

Suggestions to Make Your Project Stand Out!

Here are a few ideas for going beyond the requirements outlined in the rubric.

Augment the Training Data

Augmenting the training set might help improve model performance. Common data augmentation techniques include rotatiion, translation, zoom, flips, and/or color perturbation. These techniques can be used individually or combined.

Analyze New Image Performance in More Detail

Calculating the accuracy on these five German traffic sign images found on the web might not give a comprehensive overview of how well the model is performing. Consider ways to do a more detailed analysis of model performance by looking at predictions in more detail. For example, calculate the precision and recall for each traffic sign type from the test set and then compare performance on these five new images..

If one of the new images is a stop sign but was predicted to be a bumpy road sign, then we might expect a low recall for stop signs. In other words, the model has trouble predicting on stop signs. If one of the new images is a 100 km/h sign but was predicted to be a stop sign, we might expect precision to be low for stop signs. In other words, if the model says something is a stop sign, we're not very sure that it really is a stop sign.

Looking at performance of individual sign types can help guide how to better augment the data set or how to fine tune the model.

Create Visualizations of the Softmax Probabilities

For each of the five new images, create a graphic visualization of the soft-max probabilities. Bar charts might work well.

Section #10

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.

This is the readme file ☺

The code is included in this submission as a jupyter notebook file and a copy saved as in HTML is also included.

Data Set Summary & Exploration

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

The code for this step is contained in the second code cell of the IPython notebook.

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 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

2. Include an exploratory visualization of the dataset and identify where the code is in your code file.

The code for this step is contained in the third code cell of the IPython notebook.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data distribution of y-labels and its frequency.

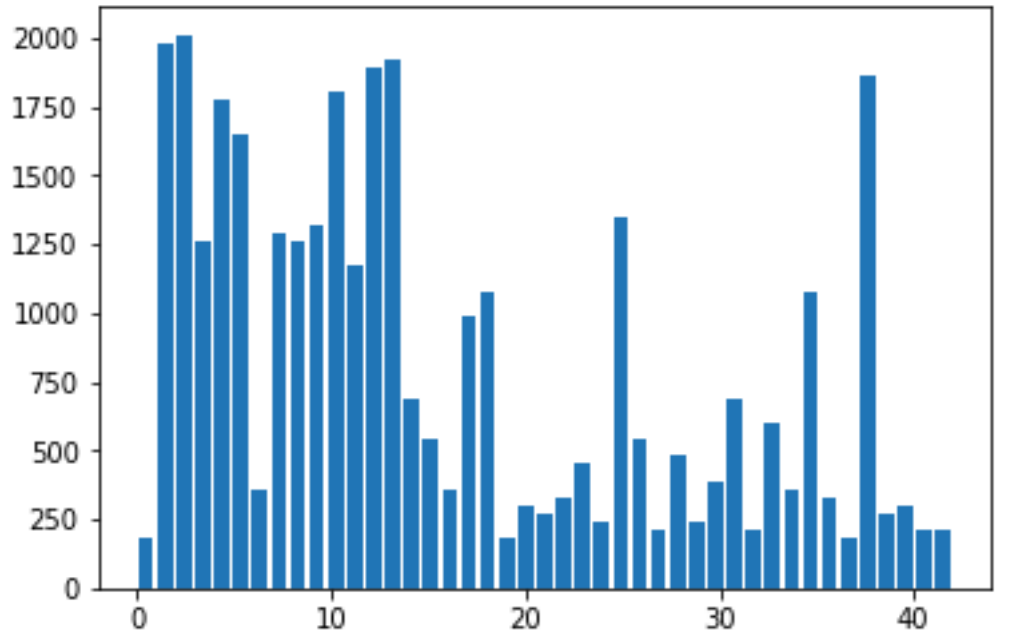
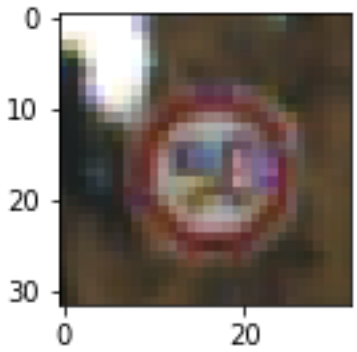


Figure above shows the classIDs with # of occurances.



also in the same code cell, the most frequent traffic sign is displayed

Design and Test a Model Architecture

3. Describe how, and identify where in your code, 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 code for this step is contained in the fourth code cell of the IPython notebook.

As a first step, I decided not to convert the images to grayscale because ...

In image recognition, it is often assumed the method used to convert color images to grayscale has little impact on recognition performance. However, this is not true based on the research below. Besides with greater GPU capacity the case for computational advantage using greyscale is weakened.

Also promise of neural networks are to work with minimal preprocessing.

http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0029740

~~I normalized the image data because ...neural networks train well and the gradient descent does not get stuck in local minimum using small numbers and variance.~~

Above normalization was turned off because results became worse

4. Describe how, and identify where in your code, you set up training, validation and testing data. How much data was in each set? Explain what techniques were used to split the data into these sets. (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, identify where in your code, and provide example images of the additional data)

The code for loading of the data is in the first cell itself. The dataset I got from Udacity actually consisted of three files:

1. Train.p (34,799)
2. Valid.p (4410)
3. Test.p (12630)

Therefore, I am confused about the question on how to setup data because all three datasets were provided. Apologies if I missed something here. Ideally (as Prof Ng also prescribes) we select a dataset of 60/20/20%

But in this case the zip file I got from Udacity already had the three files split individually.

~~To cross validate my model, I randomly split the training data into a training set and validation set. I did this by ...~~

~~My final training set had X number of images. My validation set and test set had Y and Z number of images.~~

~~The sixth code cell of the IPython notebook contains the code for augmenting the data set. I decided to generate additional data because ... To add more data to the the data set, I used the following techniques because ...~~

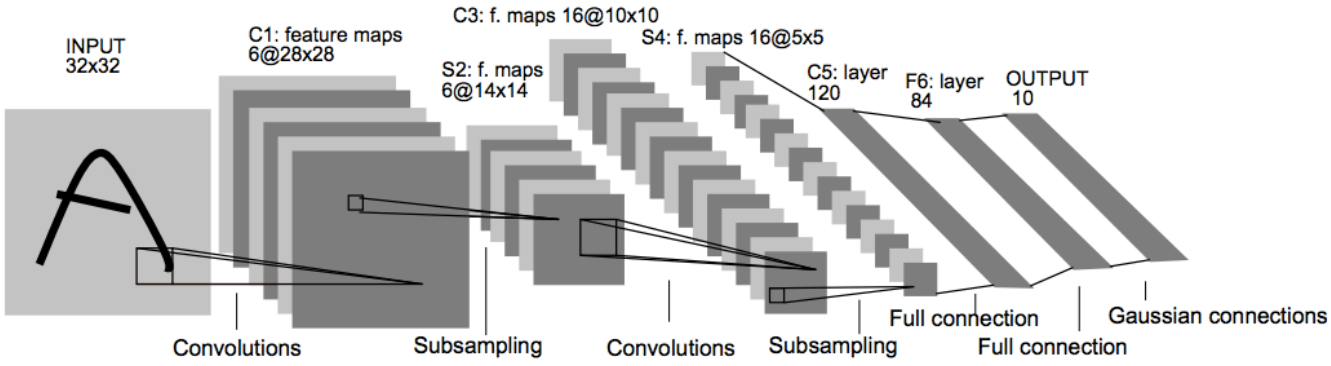
~~Here is an example of an original image and an augmented image:~~

5. Describe, and identify where in your code, 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 code for my final model is located in the 13 cell of the ipython notebook.

My final model based on the lenet architecture consisted of the following layers:

Layer	Description
Input	32x32x3 RGB image
Convolution (layer 1)	`5x5 convolution (32x32x3 in, 28x28x6 out)`
RELU	
Max pooling	`2x2 (28x28x6 in, 14x14x6 out)`
Convolution (layer 2)	`5x5 convolution  14x14x6 in, 10x10x16 out`
RELU	
Max pooling	2x2 `(10x10x16 in, 5x5x16 out)`
Flatten	`5x5x6 in, 400 out`
RELU	
Fully Connected	Input = 400. Output = 120. (Layer 3)
RELU	
Fully Connected	Input = 120. Output = 84 (Layer 4)
RELU	
Fully Connected	Input = 84. Output = 43. (Layer 5)



6. Describe how, and identify where in your code, 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.

1. The code for training the model is located in the ninth cell of the ipython notebook.

2. Adam Optimizer was used

3. My batch size is 128

4. 22 epochs (chose 22 because going further doesn't buy much – from graph in section)

The parameters were set based on trial and error. Initially I started with 15 then 20…until I saw 30 was the best. However, I feel there must be a more elegant and automated way of finding the right epochs and batch size (along with other parameters). I am looking forward to the same in the next session to find a way to automate this rather then re-invent the wheel or do this manually.

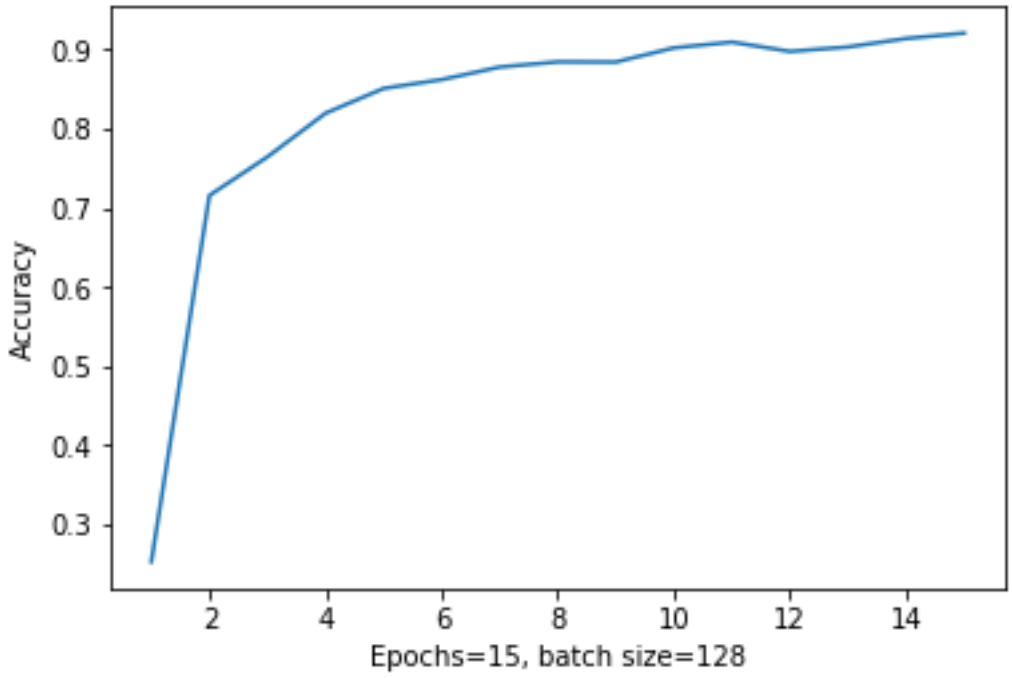
7. Describe the approach taken for finding a solution. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution 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.

The code for calculating the accuracy of the model is located in the seventh cell of the Ipython notebook.

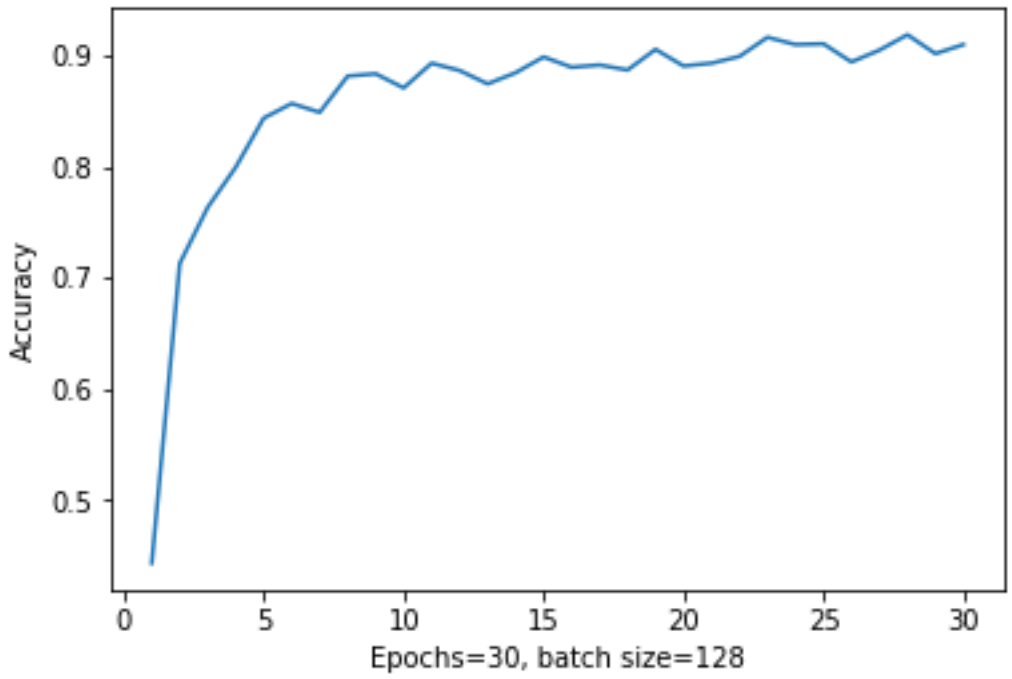
My final model results were:

- training set accuracy of 0.996
- validation set accuracy of 0.913
- test set accuracy of 0.907

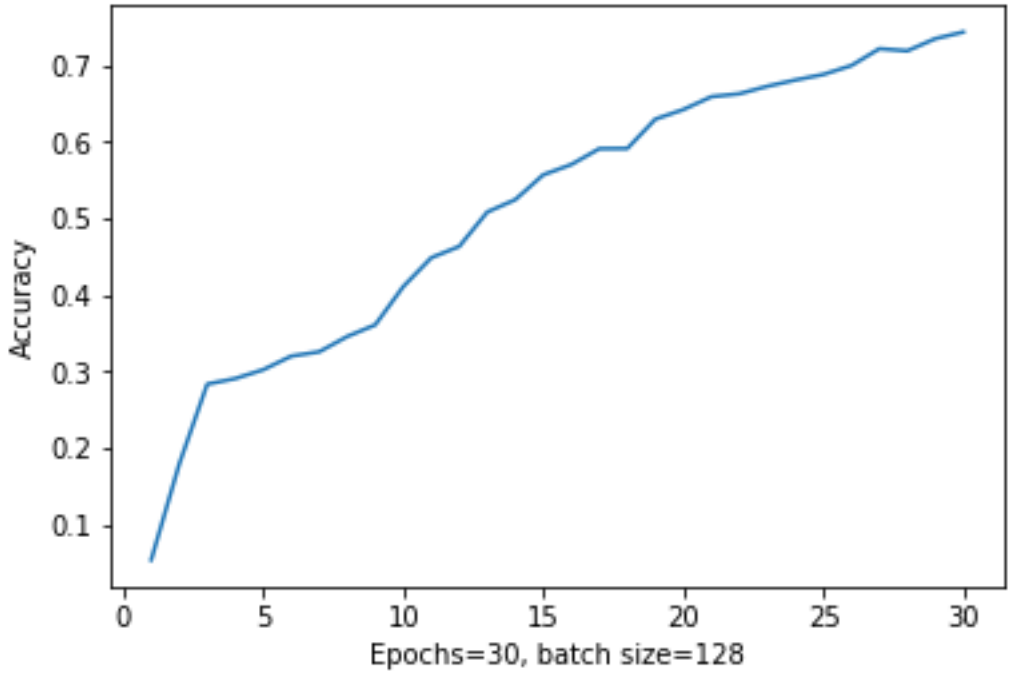
- Below is the accuracy graph for straight data (training data not normalized)



-
-



-
-
- training with data normalized (in cell 4)
-
- As we can see accuracy is worse than non-normalized data. Therefore, normalization was turned off.



-
-

~~If an iterative approach was chosen:~~

- ~~What was the first architecture that was tried and why was it chosen?~~
- ~~What were some problems with the initial architecture?~~
- ~~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 over fitting or under fitting. 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.~~
- ~~Which parameters were tuned? How were they adjusted and why?~~
- ~~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?~~

If a well known architecture was chosen:

- What architecture was chosen? LeNet5 was chosen (To be open about it ☺) I saw David Silver use the Lenet to get an accuracy of more than 0.9. Also this architecture is proven and therefore didn't want to re-invent the wheel. Convolutional Neural Networks are are a special kind of multi-layer neural networks. Like almost every other neural networks they are trained with a version of the back-propagation algorithm. Where they differ is in the architecture. Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing. (credits http://yann.lecun.com/exdb/lenet/)
- Also results on MNIST-sized images (28x28) are usually in the 5x5 range on the first layer, while natural image datasets (often with hundreds of pixels in each dimension) tend to use larger first-layer filters of shape 12x12 or 15x15.
- Why did you believe it would be relevant to the traffic sign application? Because it able to detect images fairly accurately.

  Credit: deeplearning.net for below

  Motivation

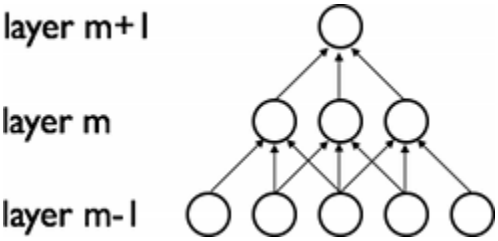
  Convolutional Neural Networks (CNN) are biologically-inspired variants of MLPs. From Hubel and Wiesel's early work on the cat's visual cortex [Hubel68], we know the visual cortex contains a complex arrangement of cells. These cells are sensitive to small sub-regions of the visual field, called a *receptive field*. The sub-regions are tiled to cover the entire visual field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.

Additionally, two basic cell types have been identified: Simple cells respond maximally to specific edge-like patterns within their receptive field. Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern.

The animal visual cortex being the most powerful visual processing system in existence, it seems natural to emulate its behavior. Hence, many neurally-inspired models can be found in the literature. To name a few: the NeoCognitron [Fukushima], HMAX [Serre07] and LeNet-5 [LeCun98], which will be the focus of this tutorial.

Sparse Connectivity

CNNs exploit spatially-local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. In other words, the inputs of hidden units in layer **m** are from a subset of units in layer **m-1**, units that have spatially contiguous receptive fields. We can illustrate this graphically as follows:

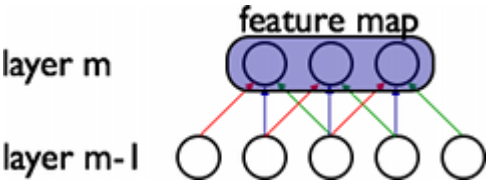


Imagine that layer **m-1** is the input retina. In the above figure, units in layer **m** have receptive fields of width 3 in the input retina and are thus only connected to 3 adjacent neurons in the retina layer. Units in layer **m+1** have a similar connectivity with the layer below. We say that their receptive field with respect to the layer below is also 3, but their receptive field with respect to the input is larger (5). Each unit is unresponsive to variations outside of its receptive field with respect to the retina. The architecture thus ensures that the learnt "filters" produce the strongest response to a spatially local input pattern.

However, as shown above, stacking many such layers leads to (non-linear) "filters" that become increasingly "global" (i.e. responsive to a larger region of pixel space). For example, the unit in hidden layer **m+1** can encode a non-linear feature of width 5 (in terms of pixel space).

Shared Weights

In addition, in CNNs, each filter h_i is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a *feature map*.



In the above figure, we show 3 hidden units belonging to the same feature map. Weights of the same color are shared—constrained to be identical. Gradient descent can still be used to learn such shared parameters, with only a small change to the original algorithm. The gradient of a shared weight is simply the sum of the gradients of the parameters being shared.

Replicating units in this way allows for features to be detected *regardless of their position in the visual field.* Additionally, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learnt. The constraints on the model enable CNNs to achieve better generalization on vision problems.

- How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well? Of course there is much scope for improvement, but I feel there are quite a few moving parts and I believe there must be a more elegant solution and waiting to discover in the next few lessons. If there aren't any then I will work to automate some of the parameter assignments.

Test a Model on New Images

8. 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~~ six German traffic signs that I found on the web: One was deliberately chosen as it was not there as a classid (railway).

The first image (most popular with German drivers ☺) I feel should be moderately difficult to predict. Because there are quite a few similar signs which stands of end of limit for various speed limits.

The second one should be fairly easy to predict because of the solid lines around the triangle and the man with shovel

The 3rd should be fairly easy but the number 6 could be confused with 5 as well

The 4th sign should be straightforward because of two distinct arrows

The 5th no-entry sign also has easy to detect lines and patterns and convnets should find it easily from layer 1 itself

The 6th sign I chose deliberately because it does not exist in the classification set. It should be difficult to detect

9. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. Identify where in your code predictions were made. 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).

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

Here are the results of the prediction (* ones are right):

Image	Prediction
*Go straight or left	Go straight or left
Railway crossing ahead (not in training set)	Dangerous curve to the right
Road Work	Right-of-way at the next intersection
*No Entry	No Entry
*End of all speed and passing limits	End of all speed and passing limits
Speed limit (60km/h)	Speed limit (50km/h)

The model was able to correctly guess 3 of the 6 traffic signs (actually 3 out of 5 since one image is not even in the training set), which gives an accuracy of 50%. This compares unfavorably to the accuracy on the test set of ...0.89

10. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction and identify where in your code softmax probabilities were outputted. 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 in the 11th cell of the Ipython 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:

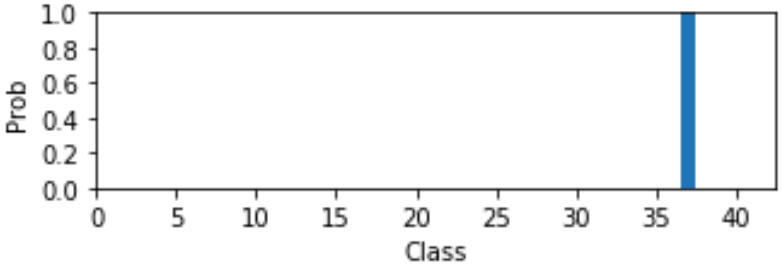
(very cumbersome to write in the table format) therefore copy pasted from code output

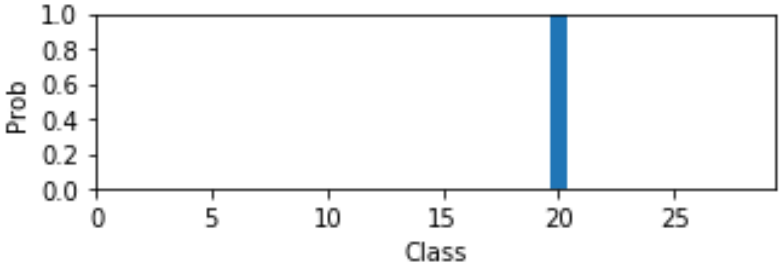
```
Image: 1
Probabilty: 1.0 Prediction: Go straight or left
Probabilty: 6.97796e-15 Prediction: Roundabout mandatory
Probabilty: 0.0 Prediction: Speed limit (20km/h)
Probabilty: 0.0 Prediction: Speed limit (30km/h)
Probabilty: 0.0 Prediction: Speed limit (50km/h)
-----------------------------------
Image: 2
Probabilty: 0.999542 Prediction: Dangerous curve to the right
Probabilty: 0.000283609 Prediction: No passing
Probabilty: 0.000173827 Prediction: Traffic signals
Probabilty: 4.85086e-07 Prediction: Children crossing
Probabilty: 1.49301e-08 Prediction: No passing for vehicles over 3.5 metri
c tons
-----------------------------------
Image: 3
Probabilty: 1.0 Prediction: Right-of-way at the next intersection
Probabilty: 7.68819e-13 Prediction: Children crossing
Probabilty: 1.01727e-14 Prediction: Road work
Probabilty: 1.03806e-15 Prediction: Beware of ice/snow
Probabilty: 3.26071e-22 Prediction: Slippery road
-----------------------------------
Image: 4
Probabilty: 1.0 Prediction: No entry
Probabilty: 6.74466e-09 Prediction: Stop
Probabilty: 1.98259e-20 Prediction: Right-of-way at the next intersection
Probabilty: 7.33229e-21 Prediction: Traffic signals
Probabilty: 2.21429e-22 Prediction: Speed limit (20km/h)
-----------------------------------
Image: 5
Probabilty: 0.978972 Prediction: End of all speed and passing limits
Probabilty: 0.0210057 Prediction: End of speed limit (80km/h)
Probabilty: 2.25987e-05 Prediction: End of no passing
Probabilty: 4.16842e-14 Prediction: End of no passing by vehicles over 3.5
metric tons
```

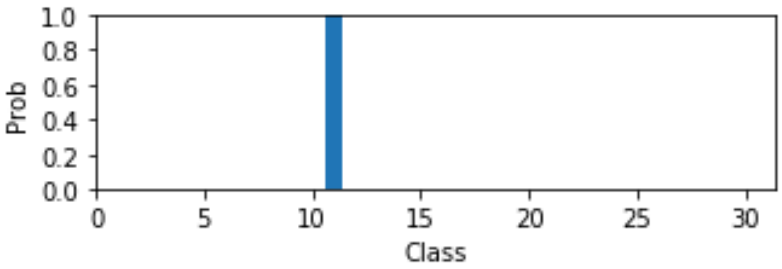
```
Probabilty: 2.11962e-15 Prediction: Priority road
-----------------------------------
Image: 6
Probabilty: 0.999949 Prediction: Speed limit (50km/h)
Probabilty: 5.10353e-05 Prediction: Speed limit (30km/h)
Probabilty: 1.33654e-08 Prediction: Speed limit (80km/h)
Probabilty: 4.15807e-16 Prediction: Speed limit (60km/h)
Probabilty: 2.13067e-21 Prediction: Priority road
-----------------------------------
```

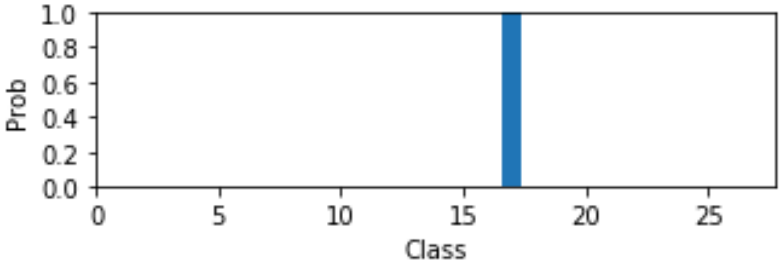
(OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)

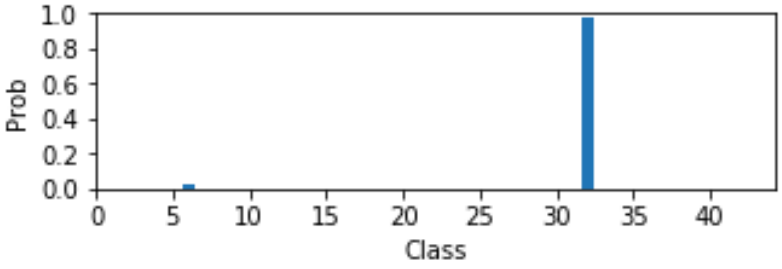
from cell 12 in notebook


Prob
Class
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
35
40

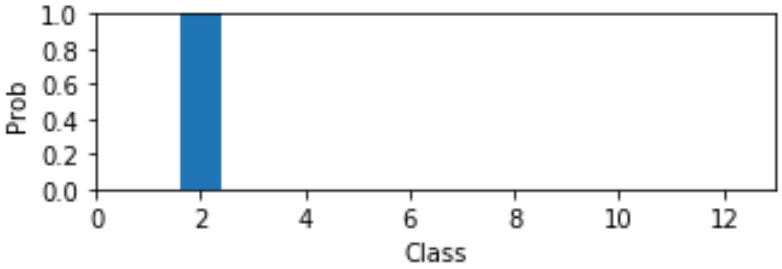

Prob
Class
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25


Prob
Class
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30


Prob
Class
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25


Prob
Class
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
35
40


60


Prob
Class
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12