This is the write-up report for project2 Traffic Sign Classification.

I. Dataset Exploration

First of all, all cells in this section is combined into two cells as project developes to reduce the number of “shift + enter” for easier execution and used as an experimental debugging place.

Dataset Summary & Exploratory Visualization

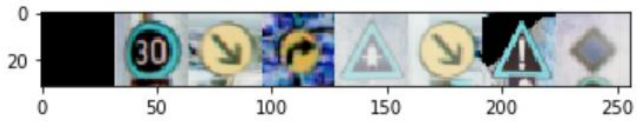
1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

```
Number of training examples = 34799
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
<class 'numpy.ndarray'>
(34799, 32, 32, 3)
(34799,)
(34799, 2)
(34799, 4)
1 38 33 11 38 18 12
```

Above figures are the outcomes of following print format describe various datasets.

```
print("Number of training examples =", n_train)
print("Number of testing examples =", n_test)
print("Image data shape =", image_shape)
print("Number of classes =", n_classes)
print(type(X_train))
print(X_train.shape)
print(y_train.shape)
print(size_train.shape)
print(crd_train.shape)
```

The number series “1 38 33 11 38 18 12” at the end is actually the class id of the sign images following.



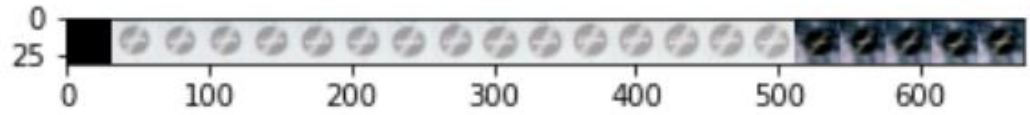
2. Include an exploratory visualization of the dataset.

This image series is created with following “gallery” function to allow easier exploration of signs than simple matplotlib using subplot. This function takes a 4D feature set (standard X feature we use in this project), first index number of desired image (fst_n), and the number of total images in series (n).

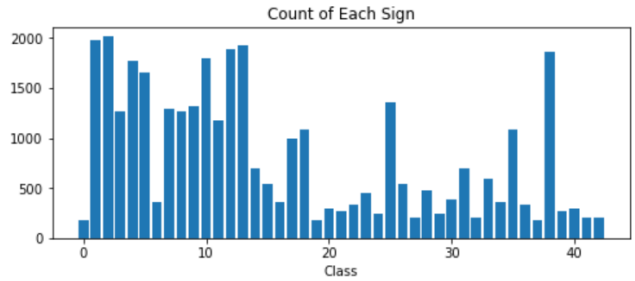
```
def gallery(x, fst_n, n):
    stack = np.zeros(x.shape[1:])
    #stack = [[] for row in range(x.shape[1])]
```

Keep in mind, there is a minor usability bug that first block is always black, filled with zeros.
Also, it returns some dimension error sometimes.
In that case, initial stack call statement has to be switched by commenting out/uncommenting.

One of the weird thing I noticed was, using this function, the training features in original data set appear to have series of similar signs. (I double checked the several times, make sure I did not corrupt it) Each of them were slightly different, but almost too similar to hardly notice the small variation. This makes me to suspect if the original was overwritten somehow.



And the last graph is the number of sign counts in each class.



In next cell, it returns index number of unique to each class, and randomly output one of signs.

II. Design and Test a Model Architecture

Preprocessing

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.

There were only two types of preprocessing used - “grayscale” and “normalization.”
A few of different “grayscale” and “normalization” techniques were explored.

For example, some of techniques explored include:
Weighted grayscale

```
def weightedAverage(pixel): # weighted grey
    return 0.299*pixel[0] + 0.587*pixel[1] + 0.114*pixel[2]
```

(explanation of specific figure found here:
https://samarthbhargav.wordpress.com/2014/05/05/image-processing-with-python-rgb-to-grayscale-conversion/)

Normalization of grayscale with min-max scaling at [0.1, 0.9]

```
def normalize_grayscale(image_data):
    return 0.1+0.8*image_data/255
```

In addition, grayscale conversion using cv2, PIL were also considered, but ended up using simple averaging of RGB for grayscale and (x-128)/128 was used for normalization because other methods were either too slow or seemed to somehow distort datasets I suffered more than a week to fix.

Model Architecture

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.

There were two architecture models were tried/modified from: LeNet and AlexNet.

LENET LAYER	ALEXNET LAYER	DESCRIPTION
Input	Input	Can be either 32x32x1 or 32x32x3 with small modification
Conv 5x5: 32x32x1 -> 28x28x6	Conv 3x3: 32x32x1 -> 30x30x64	1x1 strides for every Conv, and 2x2 strides for every maxpooling.
ReLu	MaxPool -> 14x14x64	
MaxPool ->14x14x6	ReLu	
Conv 5x5: 14x14x6 -> 10x10x16	Conv 3x3: 14x14x64 -> 12x12x64	
ReLu	MaxPool -> 6x6x64	
MaxPool ->5x5x16	ReLu	
Fully Connected (FC) 400 -> 120	FC 2304 -> 800	
ReLu	ReLu	
FC 120 -> 84	FC 800 -> 120	
ReLu	ReLu	
FC 84 -> 43	FC 120 -> 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.

Parameters were chosen base on trial and error. Also, dropout was also tried with different values, but commented out because seem to have adverse affect on the result somehow.

Model Training

Parameter I used were:

```
EPOCHS = 15
BATCH_SIZE = 128
(learning) rate = 0.001
mu = 0
sigma = 0.1
```

The accuracy for both models become stable after about 10 epochs at 0.001 rate, so I kept 15. Also, another line of codes I added were to reshape the feature sets because grayscale turned the shape to be Nx32x32 instead of Nx32x32x1 with following code, which created dimension conflict.

```
X_train = array(X_train).reshape(len(X_train),32,32,1)
```

Everything else was just ordinary, including standard AdamOptimizer.

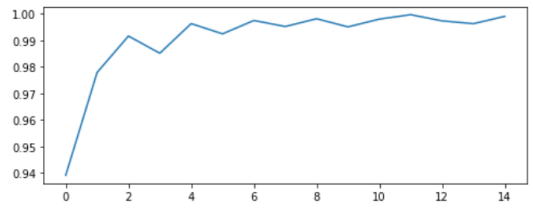
Solution Approach

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.

LeNet model is yielding about 88~90% validayion accuracy and AlexNet yield 99% training set accuracy and 93~94% validation accuracy quite stable with test accuracy well around 92~93%.

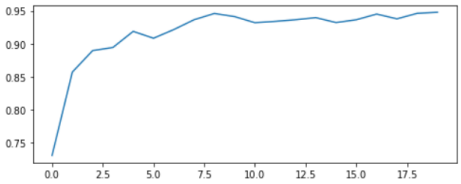
```
Training...
EPOCH 1 ... Training Accuracy = 0.939
EPOCH 2 ... Training Accuracy = 0.978
EPOCH 3 ... Training Accuracy = 0.991
EPOCH 4 ... Training Accuracy = 0.985
EPOCH 5 ... Training Accuracy = 0.996
EPOCH 6 ... Training Accuracy = 0.992
EPOCH 7 ... Training Accuracy = 0.997
EPOCH 8 ... Training Accuracy = 0.995
EPOCH 9 ... Training Accuracy = 0.998
EPOCH 10 ... Training Accuracy = 0.995
EPOCH 11 ... Training Accuracy = 0.998
EPOCH 12 ... Training Accuracy = 1.000
EPOCH 13 ... Training Accuracy = 0.997
EPOCH 14 ... Training Accuracy = 0.996
EPOCH 15 ... Training Accuracy = 0.999
Model saved
```

```
: [<matplotlib.lines.Line2D at 0x2a09f05dfd0>]
```




```
Training...
EPOCH 1 ... Validation Accuracy = 0.730
EPOCH 2 ... Validation Accuracy = 0.857
EPOCH 3 ... Validation Accuracy = 0.889
EPOCH 4 ... Validation Accuracy = 0.894
EPOCH 5 ... Validation Accuracy = 0.919
EPOCH 6 ... Validation Accuracy = 0.908
EPOCH 7 ... Validation Accuracy = 0.921
EPOCH 8 ... Validation Accuracy = 0.936
EPOCH 9 ... Validation Accuracy = 0.946
EPOCH 10 ... Validation Accuracy = 0.941
EPOCH 11 ... Validation Accuracy = 0.932
EPOCH 12 ... Validation Accuracy = 0.934
EPOCH 13 ... Validation Accuracy = 0.936
EPOCH 14 ... Validation Accuracy = 0.939
EPOCH 15 ... Validation Accuracy = 0.932
EPOCH 16 ... Validation Accuracy = 0.936
EPOCH 17 ... Validation Accuracy = 0.945
EPOCH 18 ... Validation Accuracy = 0.938
EPOCH 19 ... Validation Accuracy = 0.946
EPOCH 20 ... Validation Accuracy = 0.948
```

```
Out[7]: [<matplotlib.lines.Line2D at 0x7f786c56ee10>]
```



Training set accuracy and Validation set accuracy for AlexNet

```
from numpy import array
X_test = array(norm_gray3).reshape(len(norm_gray3),32,32,1)
assert len(X_test)==len(y_test)
#print(len(np.unique(y_test)), X_test.shape)

saver = tf.train.Saver()
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    saver.restore(sess, './model.ckpt')
    test_acc = evaluate(X_test, y_test)
    print("Test Accuracy = {:.3f}".format(test_acc))
```

```
INFO:tensorflow:Restoring parameters from ./model.ckpt
Test Accuracy = 0.937
```

Test set Accuracy for AlexNet

I chose these two simple models because they were more familiar and well known.
I just simply didn't have enough time to explore any other models.

I began with basic LeNet model I went through during the course lesson. Later on, AlexNet was chosen because it doesn't seem to be that much difficult and known to deliver better result than LeNet.

Just like project 1, the biggest challenge was simply implementing and make it work.

At the beginning I ran into memory run out message, found there were multiple versions of python installed, and the system was using 32-bit version. I had to reinstall entire python/anaconda on the course making it work.

Moreover, there were two particular issues I struggled hard, spent almost a week to solve each issue.

First was getting a very low validation accuracy and hanging there for no reason.

```
Training...
EPOCH 1 ...
Validation Accuracy = 0.054
EPOCH 2 ...
Validation Accuracy = 0.053
EPOCH 3 ...
Validation Accuracy = 0.054
EPOCH 4 ...
Validation Accuracy = 0.054
EPOCH 5 ...
Validation Accuracy = 0.046
Model saved
```

This was the same symptom some people from previous cohorts experienced.
(reference forum discussion here: https://discussions.udacity.com/t/extremely-low-accuracy/220912) However, issue was persistent and none of discussion helped to solve my case. I did everything I could think of - changed literally every code line, re-download the dataset, remove checkpoints and pickle files - to try fixing this issue.

Eventually, I fixed it by removing a hidden folder under working directory named ".git" which must be a legacy folder from git cloning. I still don't understand how it affected the code execution because it looks completely irrelevant.

The second issue was that I was stuck on setting up AWS EC2 instance environment because of the strange page loop. (see the screenshot of e-mail below)

Amazon Web Services <webservices@amazon.com> Jan 8 (10 days ago)

to me

Hello,

Samirah here from AWS support.

I apologize for all the trouble you having accessing your account and the frustration this has caused.

I researched this a bit more in depth as this is a rare issue. I found that you have two Amazon.com accounts with the same email address. The Amazon.com linked to your AWS account by password has not been verified.

I have sent you a verification email. Please verify the email, once you verify the email you will be prompted to change the password. After verifying the email and changing the email, you will be able to access the AWS account.

As always, if you have any further questions please let us know.

Wishing you a great day!

Of course, I could've just worked on my laptop, but that was quite frustrating because the laptop doesn't have GPU. Compare to these issues, optimization of parameter or other issues were not really a challenge for me.

III. Test a Model on New Images

Acquiring 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.

When I searched keyword German traffic signs, most useful ones seemed to be the output of Traffic Sign Classification project from previous cohorts, or at very best, samples from GTSRB published on INI website, which was provided as project data set. Therefore, I looked for ones that seemed to be from ordinary pictures, not much processed in any mean.


Every sign seemed to be adequately challenging because of lack of lighting at night, shadows, slight vandalism (sausage sticker), different display method (LED sign) and normal sign to test the model performance.

Performance on New Images

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

The result slightly varies and is not 100% consistent in every run, but the prediction usually gets 3~4 signs correctly out of 6. Thus, it would be safe to say the accuracy is somewhere 50~70%. Above result is actually luckier than usual, classified 4 signs correctly, yield accuracy of 67%.

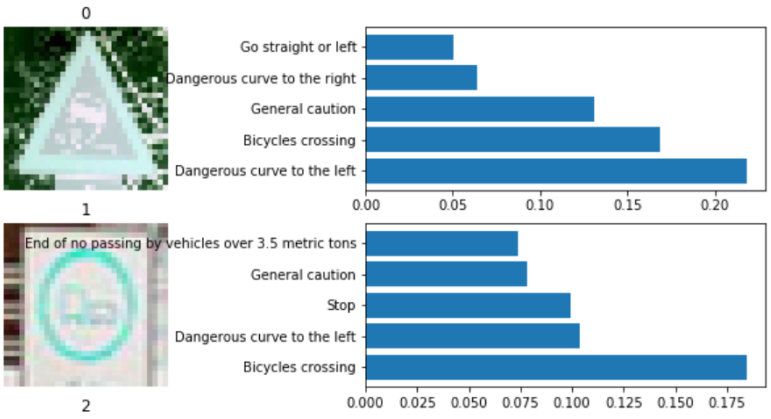
Also, two wrong classified signs were close enough, "No passing" instead of "No passing for 3.5 metric ton" and "speed limit 60" instead of "speed limit 80."

One thing to notice is, both failed ones were cropped out from the same picture and were not from standard signs but from LED signs. Perhaps lack of LED style sign samples influence make this difference in outcome.

Model Certainty - Softmax Probabilities

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.

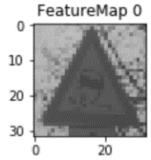
```
INFO:tensorflow:Restoring parameters from ./model.ckpt
[ 0.21838632  0.1683901   0.13082337  0.06417026  0.05072438] [19 29 18 20 37]
[ 0.1847122   0.10347389  0.09927614  0.07841742  0.07393947] [29 19 14 18 42]
[ 0.1882793   0.10990912  0.10937683  0.0896071   0.08062494] [18 29 19 35 14]
[ 0.15861748  0.12535882  0.11640382  0.0846433   0.07534718] [19 14 29 42 18]
[ 0.180904    0.0971111   0.07219674  0.07164107  0.06864091] [29 18 33 19 14]
[ 0.17539303  0.08610147  0.07213514  0.06418073  0.04981668] [18  2 19 12 14]
```

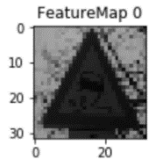


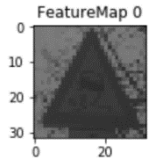
For some reason the softmax probability result returned from tensorflow seem to be different from models prediction in previous section. I'm still working on this looking for possible mix up.

IV. (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?







Currently working in progress trying to figure out how to trace the changes. It returns the initial image but not sure how to trace the training procedure.

Conclusion

This was very challenging project, and there were several unexpected obstacles made it even more difficult to finish. I will continue work on this since there are many bugs and room to improve I'm aware of.