Traffic Sign Recognition by Subrat Kamat

Main goal of the project - 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

Files Attached to this project:

1. Traffic_Sign_Classifier.ipynb – Project jupyter notebook file containing the code
2. report.html – An html copy of the jupyter notebook code file\
3. signnames.csv – The file that was provided containing the class ID and signnames
4. New_German_Signs folder containing new german signs picked up from the internet to test the trained model on them:
    a. STOP_SIGN.png
    b. General_Caution.jpg
    c. bumpy_road.jpg
    d. speed_limit_60.jpg
    e. ahead_only.jpg
5. lenet.meta , lenet.index and lenet.data-00000-of-00001 files that were created after I saved the trained model.
6. Writeup.pdf

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.

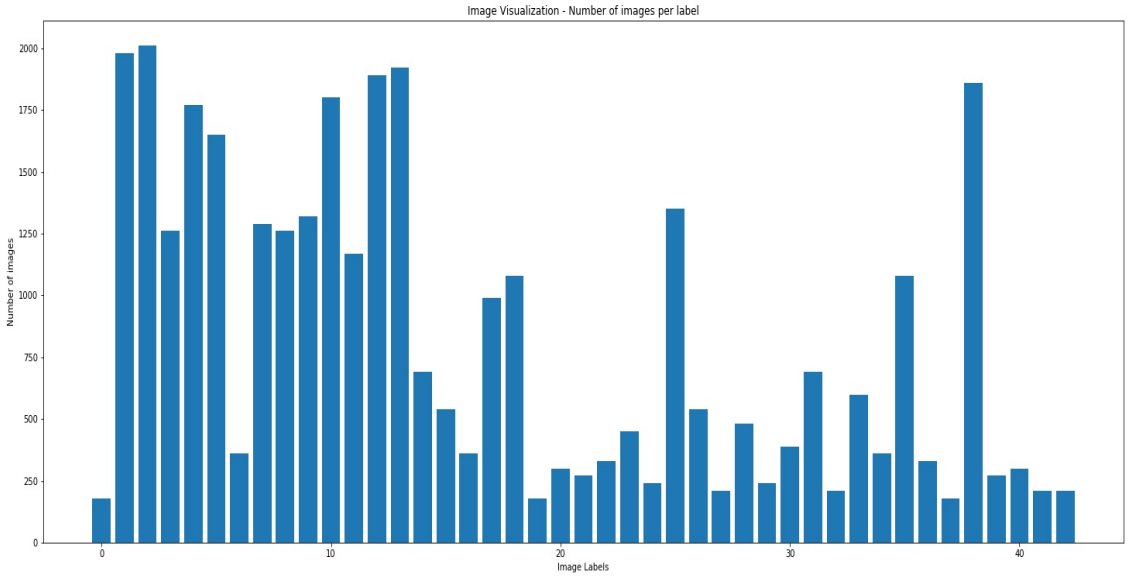
The code for this step is contained in the second code cell of the IPython notebook. I used normal python commands like len(), set() and .shape[] to calculate the summary statistics of the traffic:

* The size of training set is 34799
* The size of test set is 12630
* The shape of a traffic sign image is (32, 32, 3)
* 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. I had randomly selected three images and displayed them on this cell which are available on the report.html for you to see

Here is an exploratory visualization of the data set. It is a bar chart showing how many images pertaining to each label (Class ID) are there within the training set:



Design and Test a Model Architecture

1. Describe how, and identify where in your code, you preprocessed the image data. What tecniques 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 and fifth code cell of the IPython notebook.
Since the dataset contains colored traffic signs I chose not to convert the image to grayscale because there could be certain number of images that look similar but have different colors. I instead choose to only normalize the data by subtracting 128 from the RGB layers and then dividing by 128. This was a suggested normalization technique in one of the lessons earlier so that the training model won't have to work harder to find the global minima or in this case the appropriate values of the weights and biases of the model.

I also shuffled the data in the fifth code cell so as to improve the performance of the model.

2. 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)

This question does not stand valid as we were already given a training, validation and data set after an announcement that the training set and validation set were split for us. So this part was not a requirement within the rubric anymore. But since the question how much data is present was asked, I have mentioned it below:

- The size of training set is 34799
- The size of validation set is 4410
- The size of test set is 12630

3. 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 sixth cell of the ipython notebook.

I set the number of epochs and batch size in this cell i.e. 16 and 64.

My final model consisted of the following layers:

Layer	Description
Input	(32x32x3)
Convolutional	1x1 stride, valid padding, outputs 28x28x6
RELU	
Max Pooling	2x2 stride, outputs 14x14x6
Convolutional	1x1 stride, valid padding, outputs 10x10x16
RELU	
Max Pooling	2x2 stride, outputs 5x5x16
Flattening	outputs 400
Fully connected	outputs 120
RELU	
Fully connected	outputs 84
RELU	
Fully connected	outputs 43

A part of the training pipeline which will be discussed in the next question also contained the cross entropy calculation which contains the softmax operation. That also can be considered as a part of my last layer but I prefer it to be a part of the training pipeline because softmax helps to decide the accuracy of the model with the help of its probability calculation and is not really a part of the main model that requires tuning/training (weights and biases).

4. 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.

The code for training the model is located in the seventh, eighth, ninth and tenth cell of the ipython notebook.

To train the model, I first created two placeholders for x and y where x is a placeholder for a batch of input images and y is a placeholder for the same batch of output labels that would be used in training the model.

In eighth cell I created a command to calculate the cross entropy by using the tensorflow function that calculates cross entropy by calculating softmax and from logit output. I then created a command to calculate the loss operation by finding the mean of the cross entropy this is then passed through an Adam optimizer and minimized ('training_operation' will be the final object). This further is used to calculate the accuracy using argmax function within ninth cell.

The batch size used was 64, the number of epochs was 16 and the learning rate is 0.001 (sixth cell and eighth cell for learning rate). I used a small batch size of 64 because I realized that if I decreased the batch size then I would get better accuracy because it would be able to consider nuances within the images too. I initially started off with epoch size 10 but realized I needed more to get better validation accuracy so increased it to 16.

In the ninth cell I created a function that would calculate the accuracy for each EPOCH of trained model. This function finds the accuracy by calculating the accuracy for a batch of data according to the batch size.

The tenth cell contains the tensor flow session that trains the model with the help of the training data and then validates it with the validation set. The tensor flow session first initializes the global variables then shuffles the data and trains the model through all the layers mentioned earlier by using the training_operation object to calculate the weights and biases of all the layers and through the adam optimizer and the cross entropy functions. The trained model is then passed through a validation set to find its accuracy using the function created before. I also calculate the training accuracy. All this training is repeated for the set number of epochs. Please note that the global variables were initialized only once so after every epoch the weights biases would be leveraged from the previous step.

5. 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 tenth and eleventh cell of the Ipython notebook. The tenth cell calculates the accuracy of the training set and validation set. The eleventh cell calculates the accuracy of the test set.

My model final results after 16 epochs were:

* training set accuracy of 0.996 (99.6% accuracy)
* validation set accuracy of 0.952 (95.2% accuracy)

* test set accuracy of 0.916 (91.6% accuracy)

Though the model results for the first 15 epochs were:

EPOCH 1 ...
Training Accuracy = 0.913

Validation Accuracy = 0.840

EPOCH 2 ...
Training Accuracy = 0.964

Validation Accuracy = 0.891

EPOCH 3 ...
Training Accuracy = 0.964

Validation Accuracy = 0.894

EPOCH 4 ...
Training Accuracy = 0.982

Validation Accuracy = 0.907

EPOCH 5 ...
Training Accuracy = 0.984

Validation Accuracy = 0.923

EPOCH 6 ...
Training Accuracy = 0.994

Validation Accuracy = 0.926

EPOCH 7 ...
Training Accuracy = 0.986

Validation Accuracy = 0.926

EPOCH 8 ...
Training Accuracy = 0.994

Validation Accuracy = 0.918

EPOCH 9 ...
Training Accuracy = 0.988

Validation Accuracy = 0.924

EPOCH 10 ...

Training Accuracy = 0.995

Validation Accuracy = 0.937

EPOCH 11 ...
Training Accuracy = 0.995

Validation Accuracy = 0.945

EPOCH 12 ...
Training Accuracy = 0.995

Validation Accuracy = 0.936

EPOCH 13 ...
Training Accuracy = 0.994

Validation Accuracy = 0.930

EPOCH 14 ...
Training Accuracy = 0.997

Validation Accuracy = 0.933

EPOCH 15 ...
Training Accuracy = 0.996

Validation Accuracy = 0.945

I used the Le-Net architecture that was taught in the previous lessons. The data set provided had color images in them (I chose not to convert them to grayscale), since the Le-Net architecture contained 2 convolutional layers that would help to better train the model for the various features of the images and since the model contained three fully connected layers with RELU functions between every function thus making the number of parameters to tune less but also train the mode for any non-linearity of the image I felt Le-Net would be the best fit for training the German traffic signs. The Le-Net architecture has 13 layers and such a complexity also warrants the need of using on a traffic sign classifier.

I get a final training accuracy of 99.6%, validation accuracy of 95.2% and test accuracy of 91.6 % which are pretty high accuracy for the model as it guarantees at least 41 of the 43 images would be identified accurately all the time.

Test a Model on New Images

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

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

i.e. STOP sign


i.e. bumpy road sign

i.e. speed limit of 60 km/h sign


i.e. Ahead only sign

 i.e. General Caution sign

The first image might be difficult to classify because the image was shot from below, the second image could be slightly difficult to classify due to the trees in the background but the model should be able to classify it, the third image could be difficult to classify as it seems to be not entirely clear because of the clouds in the background having the same shade of gray as the sign due to less sunlight, the forth image was taken from the side thus giving the signal an oval shape thus making it slightly difficult to classify and the fifth image has grass in the background but I think it should be able to identify it relatively easily to the above images.

2. 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 thirteenth, fourteenth and fifteenth cell of the Ipython notebook.

Here are the results of the prediction:

Image	Prediction
Stop Sign	Stop Sign
Bumpy Road	Bumpy Road
Speed Limit - 60 km/h	Speed limit (120km/h)
Ahead only	Ahead only
General Caution	General Caution

|

The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%. This is 10% less to the accuracy on the test set of 91.6% which could be considered favorable as we just 5 images where each image constitutes 20% of the set.

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 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 located in the 16th cell of the Ipython

notebook.

For the first image, the model is absolutely sure that this is a stop sign (probability of 1.0), and the image does contain a stop sign. The top five soft max probabilities were

Probability	Prediction
1	Stop
4.49E-08	Bicycles crossing
2.88E-10	Speed limit (60km/h)
1.00E-11	Speed limit (30km/h)
3.36E-13	Speed limit (80km/h)

For the second image the model is absolutely sure that this is a Bumpy road sign (probability of 1.0), and the image does contain a Bumpy road sign. The top five soft max probabilities were

Probability	Prediction
1	Bumpy road
4.87E-25	Dangerous curve to the right
7.65E-28	Traffic signals
1.50E-29	Road work
3.53E-32	Bicycles crossing

For the third image the model incorrectly predicts the image as Speed limit (120km/h) sign (probability of 0.857) but the image contains a Speed limit of 60 km/h sign. The top five soft max probabilities were

Probability	Prediction
8.57E-01	Speed limit (120km/h)
1.42E-01	Roundabout mandatory
2.39E-04	Vehicles over 3.5 metric tons prohibited
4.12E-05	No passing
2.52E-05	Dangerous curve to the right

For the forth image the model is absolutely sure that this is an Ahead only sign (probability of 1.0), and the image does contain an Ahead only sign. The top five soft max probabilities were

Probability	Prediction
1.00E+00	Ahead only
1.38E-04	Go straight or right
9.39E-16	Turn right ahead
1.19E-17	Roundabout mandatory
1.03E-17	Go straight or left

For the fifth image the model is absolutely sure that this is a General caution sign (probability of 1.0), and the image does contain a General caution sign. The top five soft max probabilities were

Probability	Prediction
1.00E+00	General caution
1.07E-06	Traffic signals
6.85E-16	Pedestrians
1.17E-22	Road narrows on the right
4.82E-27	Right-of-way at the next intersection