Eric Victorson

3/24/2017

SDC Term 1 Project 2 Traffic Sign Classifier

#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.

###Writeup / README

####1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. You can use this template as a guide for writing the report. The submission includes the project code.

You're reading it! and here is a link to my project: **<insert link>**

###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.

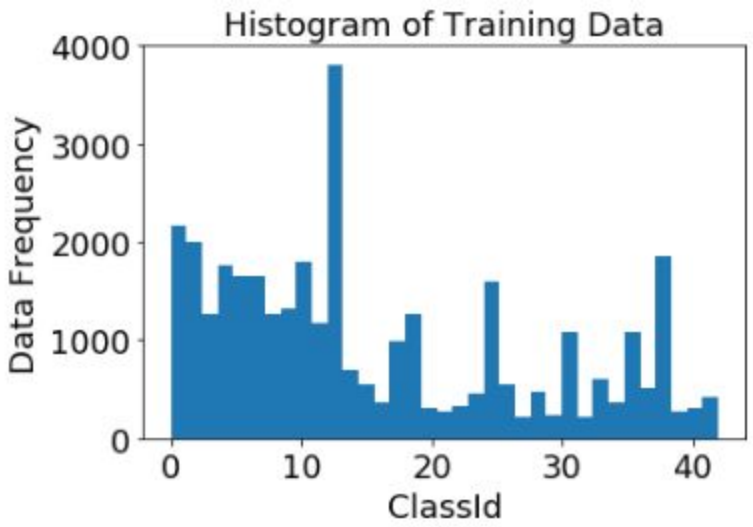
I used the pandas library to calculate summary statistics of the traffic signs data set:

- 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 second code cell of the IPython notebook.

Here is an exploratory visualization of the data set. It is a bar chart showing how the training samples are distributed.



###Design and Test a Model Architecture

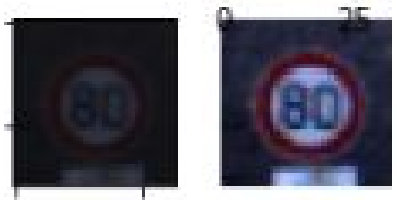
####1. Describe how, and identify where in your code, you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each

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.

Initially I tried many pre-processing steps but ultimately what ended up working the best was using opencv's normalize function with a beta = 1.0. From there I tried to normalize the distribution of training images by creating more images that were randomly translated by +/- 2 pixels rotated by +/-15 degrees and sheared via an affine transformation. Other approaches I tried were grayscale conversion followed by histogram equalization of the single intensity channel, as well as conversion to YUV color space, and then performing histogram equalization on the Y channel.

Here is an example of a traffic sign image before and after normalization.



####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)

The code for splitting the data into training and validation sets is contained in the fifth code cell of the IPython notebook.

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 146574 images (after modified images being added). My validation set and test set had 4410 and 12630 images. These last two numbers were unaltered because they were given to us as separate pickles. Interestingly when I used train_test_split to generate validation images instead of using the ones given to us I

achieved higher validation rates, which makes me question the validation image set that was given to us.

The fourth code cell of the IPython notebook contains the code for augmenting the data set. I decided to generate additional data because I wanted to get more training examples for classes that were under-represented. To add more data to the the data set, I translated images, rotated them and applied an affine transformation to shear them.

Here are some examples of modified images that were added to the data set:

And here is the histogram of how many images there are after processing for each class:

####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. It is essentially just the standard LeNet architecture we used in the lab with the addition of dropout operations between fully connected layers

My final model consisted of the following layers:

Layer	Description
Input	32x32x3 RGB image
Convolution 5x5	1x1 stride, valid padding, outputs 28x28x6
RELU	
Max pooling	2x2 stride, outputs 14x14x6
Convolution 5x5	1x1 stride, valid padding, outputs 10x10x16
RELU	
Max pooling	2x2 stride, outputs 5x5x16
Fully connected	400 inputs, 120 outputs
RELU	
Dropout	Keep prob = 0.5
Fully connected	120 inputs, 84 outputs
RELU	
Dropout	Keep prob = 0.5

Fully Connected	84 inputs, 43 outputs
Softmax	

####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 cell of the ipython notebook.

To train the model, I used a learning rate of 0.001, 100 epochs, although I could have used far less, a batch size of 128. I chose to use the Adam optimizer because from the papers I've read adaptive optimizers appeared to have the best performance and were the least computationally expensive.

####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 eighth cell of the Ipython notebook.

My final model results were:

- training set accuracy of 0.998
- validation set accuracy of 0.954
- test set accuracy of 0.939

If an iterative approach was chosen:

- What was the first architecture that was tried and why was it chosen?
    - I first chose the basic LeNet architecture.
- What were some problems with the initial architecture?

    - The initial training sets were achieving training accuracies of nearly 1.000 while the validation accuracy was only around 0.870. This indicated that the model wasn't able to generalize what it learned in the training set to the validation set well.
- 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.
    - I moved on to try adding layers of convolutions as well as in the fully connected classifier layer. I then moved on to try some inception modules. From there improved the training data set and added dropout because the training set was achieving very good accuracies whereas the validation set was still achieving relatively poor performance.
- Which parameters were tuned? How were they adjusted and why?
    - I played with the learning rate a bit but decided to leave it at 0.001. I increased the number of epochs based on where I saw the optimizer begin to stall.
- 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?
    - I think the dropout layers helped because it allowed the model to have backup methods of classification which further allowed the model to generalize to the validation set.

If a well known architecture was chosen:

- What architecture was chosen?
- Why did you believe it would be relevant to the traffic sign application?
- How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?

###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 took screenshots of 10 signs that I found in Stuttgart streets on google maps' streetview:

All of the images are likely going to be slightly difficult to classify because they were taken from odd angles, captured via screenshot from an already modified image, then resized to the appropriate dimensions. The 3rd image I expected to be especially difficult because it is different than the other 30 km/h signs as it's not round and has extra features. The 1st 4th, and 5th images all have glitches in the image where there is a portion that is much lighter than other areas of the image.

####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 tenth cell of the Ipython notebook.

Here are the results of the prediction:

Prediction	Truth
13	13
13	12
0	1
38	37

38	38
35	35
5	5
13	13
3	3
33	33

The model was able to correctly guess 7 of the 10 traffic signs, which gives an accuracy of 70%. This compares favorably to the accuracy on the test set of ...

####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 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

1st image:

Probability	**Prediction**
1.000	13
.175	12

.119	25
.029	38
.027	35

2nd image:

Probability	Prediction
0.936	13
0.029	9
0.023	15
.005	14
.002	12

3rd image:

Probability	Prediction
0.421	0
0.320	1
0.128	8
0.079	13

0.029	38

4th image:

Probability	Prediction
0.697	38
0.281	37
0.009	36
0.004	35
0.004	40

5th image:

Probability	Prediction
0.999	38
4e-7	34
1e-8	36
3e-12	32
2e-16	41

6th image:

Probability	Prediction
0.999	35
2e-7	37
8e-10	34
2e-11	33
1e-13	36

7th image:

Probability	Prediction
0.999	5
1e-5	7
8e-10	1
7e-12	12
7e-13	4

8th image:

Probability	Prediction

1.000	13
1e-8	38
4e-9	15
4e-10	12
2e-10	14

9th image:

Probability	Prediction
1.000	3
8e-14	5
1e-17	2
2e-22	34
8e-23	28

10th image:

Probability	Prediction
1.000	33
9e-15	35

4e-16	39
1e-16	34
6e-20	19