CHUCK-P / Traffic_Sign_Classifier

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

Branch: master Traffic_Sign_Classifier / writeup_TSC_CHUCK-P.md Find file Copy path

CHUCK-P Review 2 corrections 5c5043a 42 seconds from now

1 contributor

215 lines (145 sloc) 12.3 KB Raw Blame History

#Traffic Sign Recognition

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

Thanks for reading it! Here is a link to my project code

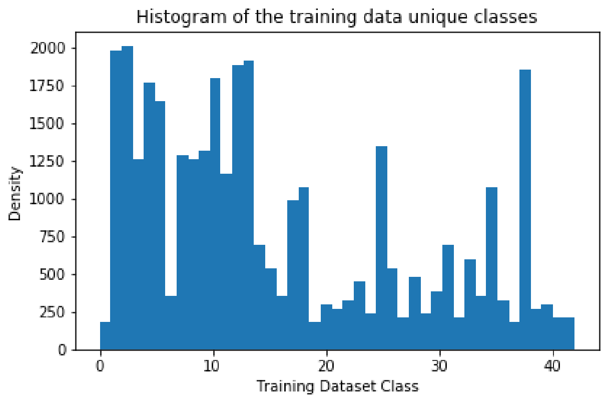
###Data Set Summary & Exploration

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

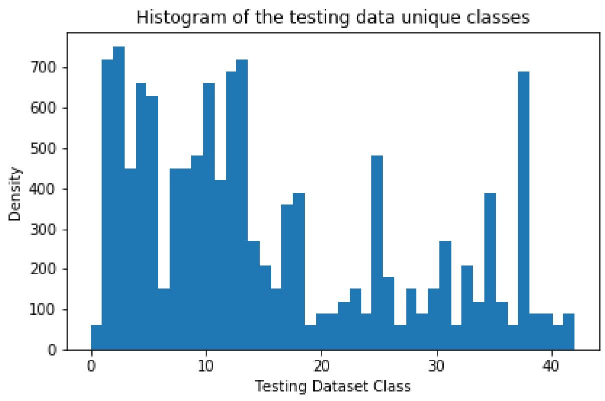
- The size of training set is 34799 examples
- The size of the validation set is 4410 examples
- The size of test set is 12630 examples
- The shape of a traffic sign image is 32 x 32 x 3 (pixels x pixels x color channels)
- The number of unique classes/labels in the data set is 43

####2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the training data set. It is a bar chart showing the distribution of number of samples per unique class



Here is a similar exploratory visualization of the testing data set.



NOTE: The testing data has very similar deficiencies as the training data. This might make it harder to have statistically significant results when the verification step is so lightly populated.

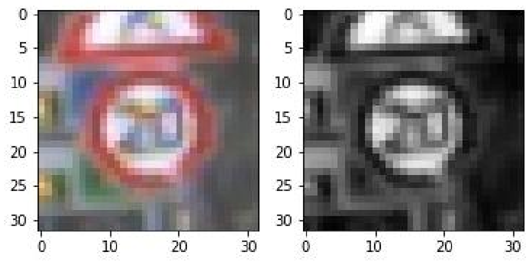
###Design and Test a Model Architecture

###Preprocessing Steps

I created a pipeline to be reused on every image that was for training, validation, testing, and supplementary testing. A summary of the steps that I used are as follows: # 1. Converted input RGB image to grayscale in order to reduce the affects of lighting conditions # 2. Normalize image range -1.0 to 1.0 to properly condition the data for Gradient Descent Method # 3. Crop image to remove unnecessary pixels to avoid confusing the training model # 4. Sharpen the cropped image in order to create more distinct features

As a first step, I decided to convert the images to grayscale because lighting conditions may affect the different channels in RGB, adversely. I don't have any experimentation to support that assumption, though. Just through experience on other projects.

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



Next, I normalized the image data to between -1.0 and 1.0 to ensure that the data was properly conditioned for gradient descent. I used pixel = (pixel - 127.5) / 127.5 instead of 128.0.

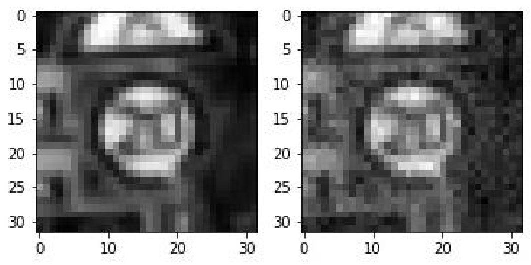
Then, I cropped the image to eliminate unnecessary features that might cause confuse the model

Finally, I sharpened the image using a Gaussian blur overlayed with the original image in order to reduce noise that are not relevant features

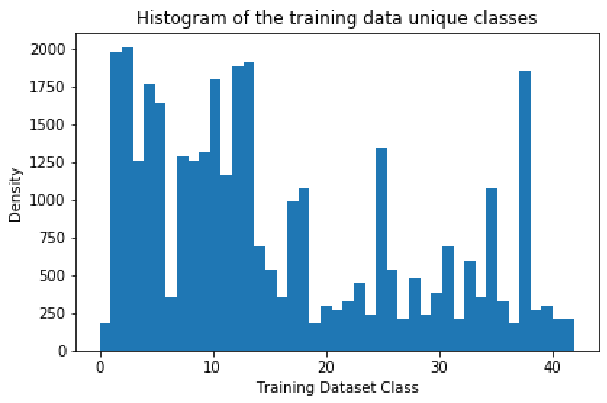
After visualizing the dataset via the histogram plot, I decided to generate additional data because a siginificant number of the traffic sign classes had less than 1200 samples available.

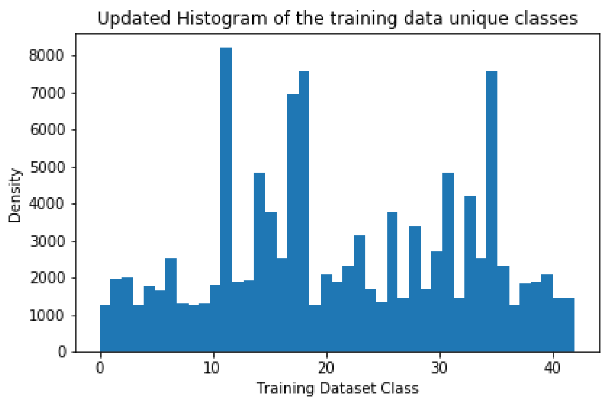
To add more data to the the data set, I sought out classes with less than 1200 samples and then used a rotation transform. The additional data was simply appended to the original data set.

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

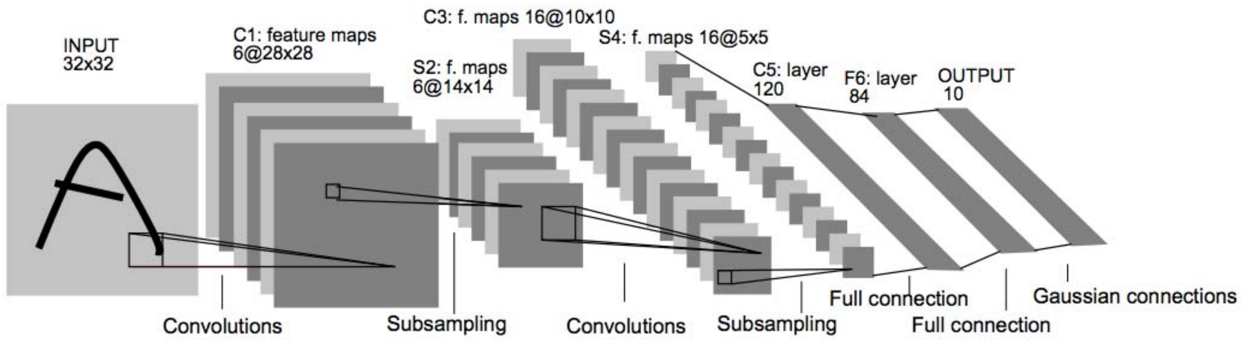


The difference between the original data set and the augmented data set is the following ...





####2. MODEL ARCHITECTURE My model architecture is heavily based upon the LeNet example presented for the MNIST example. The only small modifications that I made were to add dropout after the first and second RELU activations.

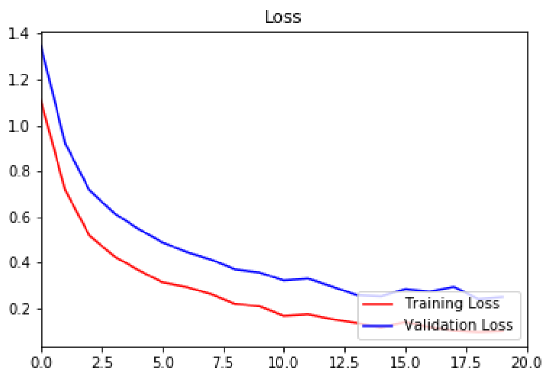


My final model consisted of the following layers:

Layer	Description
Input	32x32x1 Grayscale image
Convolution 5x5	5x5x1 filter, stride 1x1, VALID padding, Output 28x28x6
RELU	
Dropout 1	Used for training ONLY
Pooling	2x2 kernel, 2x2 stride, VALID padding, Output 14x14x6
Convolution 5x5	5x5x6 filter, 1x1 stride, VALID padding, Output 10x10x16
RELU	
Dropout 2	Used for trianing ONLY
Pooling	2x2 kernel, 2x2 stride, Output 5x5x16
Flatten	Output 400
Fully connected	Output 120
Fully connected	Output 84
RELU	
Fully connected	Output 43

####3. MODEL TRAINING Using the AdamOptimizer available in TensorFlow, I stuck with a generic batch size of 128 and really didn't experiment with larger batches nor did I deviate from the default learning rate. Prior to adding more training data via the rotations -20, -15, -10, 10, 15, 20, I did experiment with the learning rate and batch size - to no avail.

To check for overfitting, I monitored the loss of both the training and validation sets. The convergence seemed relatively



stable.

####4. SOLUTION In order to get to a solution with a validation set accuracy of at least 0.93, I added a six-fold amount of data to the classes that had less that 1200 samples. Regardless of the number of samples, a passing accuaracy would not be possible if the data were not normalized between -1 and 1 and also preprocessed using the pipeline. Most of the techniques that I used were well know implementations. The only change to the LeNet architecture that I made was to add dropouts after the first and second RELU activation functions.

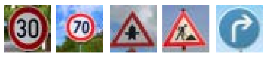
My final model results were:

- training set accuracy of 99%
- validation set accuracy of 94.7%
- test set accuracy of 93.2%

Q1: What was the first architecture that was tried and why was it chosen? A1: I tried the vanilla version of the LeNet model except changed the first convolution to have a three (3) channel input filter, originally. This may have been fine, but during my experimentation, I simply changed to grayscale (one (1) channel) and continued prototyping along that branch. Q2: What were some problems with the initial architecture? A2: I saw almost immediately that it was overfitting. The loss curves between the training and the validation curves were diverging. Q3: How was the architecture adjusted and why was it adjusted? A3: Initially, I started by changing the filter to single channel. Then I added in dropout after the first and second convolutions. This did not work - so then I tried later on after the fully conencted convolutions. That didn't work. I then tried again after convolution 1 and 2, but after the RELU. This seemed to work better. Q4: Which parameters were tuned? How were they adjusted and why? A4: I tried adjusting the learning rate and the batch size. Neither seemed to help. I then tried a large number of epochs to no avail. I ended up with all the same parameters values that were the default. The big mistake that I was making was setting the test and validation Evaluate function to have a keep_prob of .5. This eliminated some confusing results. Q5: What are some of the important design choices and why were they chosen? A5: Dropout seemed to be the most relevant. It quickly addressed the overfitting problem. Changing from 3 channels to 1 channel (grayscale), may have been important, but I haven't proven that.

Test a Model on New Images

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



DISCUSSION: These images exhibit varying brightness and contrast. Their backgrounds might not be as complicated as, say, a city scene - but they are not clear, either. The additional five traffic signs are also varied in angle. Most significantly, the 70kph sign is angled clockwise, slightly. This was the sign that was most difficult to classify.

Here are the results of the prediction:

ClassID	Image	Prediction
1	30kpm	30kpm
4	70kpm	20kpm
11	RoW Next Inter	RoW Next Inter
25	Road Work	Road Work
33	Turn Right Ahead	Turn Right Ahead

Originally, the model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%. This compares poorly to the accuracy on the test set of 93%. I believed that I was cropping the original image too closely to the edges, so I resubmitted the same image at 32x32 with slightly larger borders.

Even after re-cropping the image that was failing (70kpm), my accuracy stayed at 80%. One possibility may be that there was not enough quality training data to classify the 70kpm versus 20kpm - since they are very similar. Both the 20kpm and 70kpm had under 1200 original samples submitted. Additional samples were generated by applying small rotation transforms to these originals, so perhaps that was not sufficient. More importantly, the 0 only had about 200 samples to begin with and the 4 had just over 1200 (the limit for adding samples). So both ended up with a low amount of final samples.

SOFTMAX PREDICTIONS

All of the different new samples except for the 70kpm (ClassID 4) reported a prediction probability of over 96% (of those, all were 99% except for the "Turn Right Ahead" ClassID 33). This time, the reported probability was almost 37% that the 70kpm (classID 4) was 20kpm (classID 0). The second highest probability was for 30kpm (34%) and then 70kpm (9%).

Probability	Prediction
.998	30kpm
.001	80kpm

Probability	Prediction
.532	70kpm
.338	20kpm
.103	General Caution
.011	Slippery Road
.007	Children Crossing

Probability	Prediction
.997	Right of Way Next Intersection
.003	Beware of Ice/Snow

Probability	Prediction
1.000	Road Work

Probability	Prediction
.952	Turn Right Ahead
.017	Traffic Signals
.006	Stop
.005	Priority Road
.005	100kpm