Traffic Sign Classifier Project
Tue 21-Jul-2020
John Hilbun

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 (this document)

Rubric Points:

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. The submission includes the project doe.

   This document is the Writeup.

   The project code is located here - https://github.com/johnhilbun/P3-Traffic-Sign-Classifier-SDCND/blob/main/Traffic_Sign_Classifier.ipynb

Dataset Set Summary and Exploration

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.

   The dataset was provided in the baseline project as Pickle files:

   ```
   training_file = "../data/train.p"
   validation_file= "../data/valid.p"
   testing_file = "../data/test.p"
   ```

   The Python, Numpy and Pandas libraries were used to calculate summary statistics of the traffic signs data set:
   - The size of the training set is 34,799 samples.
   - The size of the validation set is 4,410 samples
   - The size of the test set is 12,630 samples.
   - The image shape is 32x32x3 (RGB)
   - The training set has 43 unique classes

2. Include an exploratory visualization of the dataset.

   Each of the training, validation and test datasets contain 32x32 pixel RGB images of German Traffic Signs.

   Figure 1. 32x32 pixel RGB image - German Traffic Sign

   Here is an exploratory visualization of the data set. It is a bar chart showing how the 43 unique classes are distributed in the training, validation and test datasets.

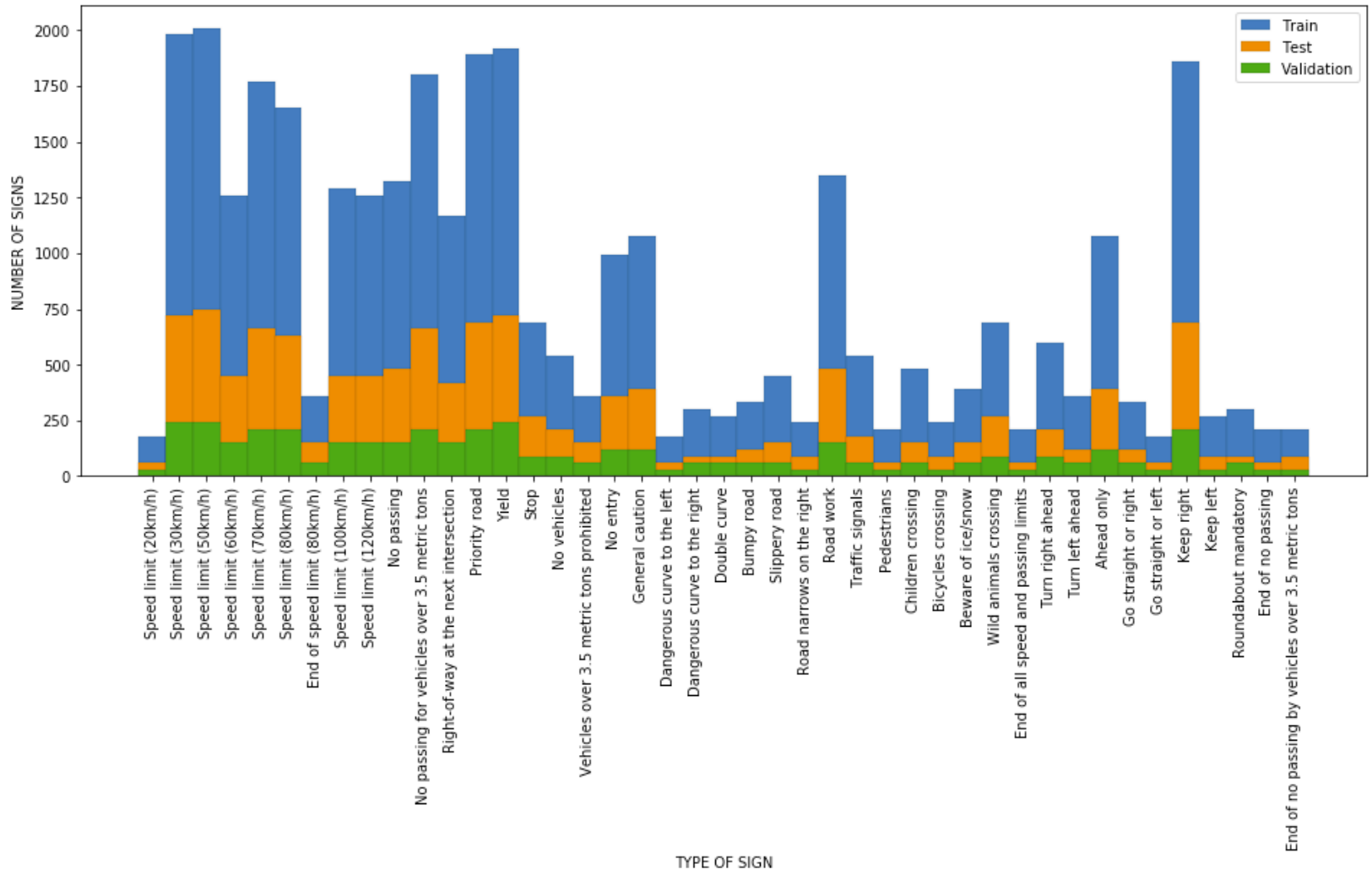
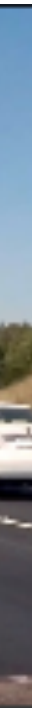


Figure 2. Types of signs in the Train, Validation and Test Datasets

Design and Test a Model Architecture



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.

   As a first step, testing was run with no normalization. Goal was to see how the system performed “as is”.

   In three runs, the results were:
   0.905
   0.886
   0.894
   For an average of 0.895.

   As a second step, grayscale was introduced.

   In three runs, the results were:
   0.880
   0.885
   0.927
   For an average of 0.897.

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

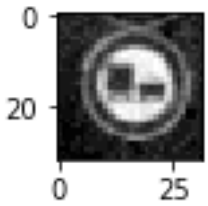


Figure 3. 32x32 pixel Grayscale image - German Traffic Sign

As a final step, I added normalization.

In three runs, the results were:
0.916
0.888
0.924
For an average of 0.909

So, using grayscale and normalizing the images resulting in marginal improvements. But, nothing that got the results past the 0.93 requirement.

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.

The final model consisted of the following layers:

Layer	Description
Input	32x32x1 Grayscale image
Convolution 1x1	1x1 stride, valid padding, outputs 28x28x6
Normalization	Batch Normalization
Tanh	
Max pooling	2x2 stride, outputs 14x14x6
Convolution 2x2	1x1 stride, valid padding, outputs 10x10x16
Normalization	Batch Normalization
Tanh	
Max pooling	2x2 stride, valid padding, outputs 5x5x16
Flatten	5x5x16 to 400

Dropout	Dropout layer (keep_prob=0.75 training / 1.0 prediction)
Fully Connected	Input 400. Output 120.
Normalization	Batch Normalization
Tanh	
Dropout	Dropout layer (keep_prob=0.75 training / 1.0 prediction)
Fully Connected	Input 120. Output 84.
Normalization	Batch Normalization
Tanh	
Dropout	Dropout layer (keep_prob=0.75 training / 1.0 prediction)
Fully Connected	Input 84. Output 43.

3. Describe how you trained your model. The discussion can include the type of optimizer, number of epochs and any hyperparameters such as learning rate.

    To train the model, the LeNet Architecture example provided by the instructor was used with the following changes:

        Reduced Epochs from 20 to 10.
        Added Batch Normalization in multiple locations.
        Changed activation function from ReLU to Tanh.
        Added Dropout layers.

    These Hyperparameters remained at the settings provided in the example:

        BATCH_SIZE = 128
        mu = 0
        sigma = 0.1

4. Describe the approach taken for finding 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.

    My final model results were:
    - training set accuracy of 0.991
    - validation set accuracy of 0.949
    - test set accuracy of 0.930

    A well known architecture was chosen:
    - What architecture was chosen?
        The LeNet Architecture provided by the example code was selected.

- Why did you believe it would be relevant to the traffic sign application?
  It made the most sense to me to start with the example code provided rather than start from scratch. This was chosen as the example code by the Instructor for a reason so I decided to take advantage of their experience.

- How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?
  The model reaches ~ 0.93 performance within 3-4 Epochs and remains above that threshold during the remaining Epochs. The result is performance greater than 0.93 (which meets the 0.93 minimum requirement).

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.

   The following site was used to grab a single png file with examples of the 43 different types of signs - https://www.researchgate.net/figure/An-example-of-the-43-traffic-sign-classes-of-GTSRB-dataset_fig9_311896388/download

Figure 4. German Traffic Sign Examples

The screen capture was used to select five images for prediction testing.

The first image was chosen because it looks pretty clear. Wanted to make sure to give the algorithm a good chance of getting at least one right.

Figure 5. German Traffic Sign Example 1 - General Caution

The second image was chosen because it looked a little dark in the summary view (Figure 4).

Figure 6. German Traffic Sign Example 2 - Keep Right

The third image was chosen because there is a bit of white bleeding into the red circle. Wanted to see how the algorithm handled that.

Figure 7. German Traffic Sign Example 3 - Speed limit (20km/h)

The fourth image was choses because it appeared dark. Thought that might give the algorithm a challenge.

Figure 8. German Traffic Sign Example 4 - Stop Sign

The fifth image was picked because it looked like a challenge too. There are multiple colors that show up where red would be expected on the left and right side of the sign.

Figure 9. German Traffic Sign Example 6 - Yield

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.

Image	Prediction
General Caution	General Caution
Keep Right	Keep Right
Speed Limit 20	Speed Limit 20
Stop	Stop
Yield	Yield

   The accuracy of these new predictions:
   The model was able to correctly estimate 5 of the 5 traffic signs, which gives an accuracy of 100%.

   Compare the accuracy to the accuracy on the test set.
   This compares favorably to the accuracy on the test set of 93%.

3. Describe how certain the model is when predicting 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.

   Complete this section.
   The code for making predictions on my final model is located in the 15th cell of the Ipython notebook.

   For the first image, the model is very sure this is a General Caution sign (probability .99) and the image does contain a General Caution sign. The top five softmax probabilities were:

Probability	Prediction
0.99	General Caution
0.001	Traffic signals
0.0003	Pedestrians
0.00005	Road narrows on the right
0.00001	Dangerous curve to the right

For the second image, the model is very sure this is a Keep Right sign (probability .99) and the image does contain a Keep Right sign. The top five softmax probabilities were:

Probability	Prediction
0.99	Keep right
0.00005	Turn left ahead
0.00001	Dangerous curve to the right
0.000006	Go straight or right
0.000004	End of no passing

For the third image, the model is very sure this is a Speed limit (20km/h) sign (probability .99) and the image does contain a Speed limit (20km/h) sign. The top five softmax probabilities were:

Probability	Prediction
0.99	Speed limit (20km/h)
0.008	Speed limit (30km/h)
0.0004	Speed limit (70km/h)

0.00009	End of all speed and passing limits
0.00007	Speed limit (120km/h)

For the fourth image, the model is very sure this is a Stop sign (probability .99) and the image does contain a Stop. The top five softmax probabilities were:

Probability	Prediction
0.99	Stop
0.002	Speed limit (70km/h)
0.0002	No entry
0.0001	Turn right ahead
0.00003	General caution

For the fifth image, the model is very sure this is a Yield sign (probability .99) and the image does contain a Yield sign. The top five softmax probabilities were:

Probability	Prediction
0.99	Yield
0.004	Ahead only
0.0002	Keep right
0.0001	Turn left ahead
0.0001	Road work

Suggestions to Make Your Project Stand Out

Augment the Training Data

Augmenting the training set might help improve model performance. Common data augmentation techniques include rotation, 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 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 is 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.

Visualize Layers of the Neural Network

See Step 4 of the Ipython notebook for details about how to do this.

Issues (notes for the writeup)
- Using Udacity workspace because the examples use TensorFlow 1.0. When I use TensorFlow 2.0 locally, running into an issue related to getting the MNIST example that doesn't seem worth the effort to track down right now.