Traffic Sign Recognition

Writeup

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

Writeup / README

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

You're reading it! and here is a link to my project code

Data Set Summary & Exploration

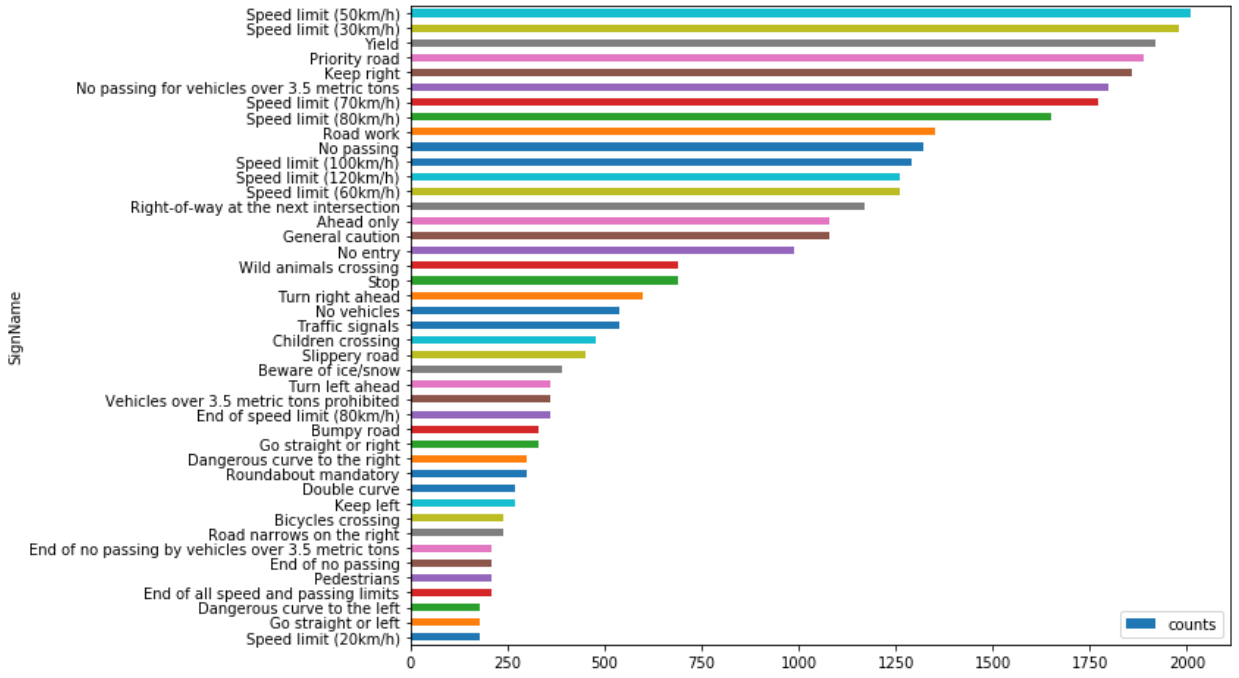
1. Provide a basic summary of the data set.

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

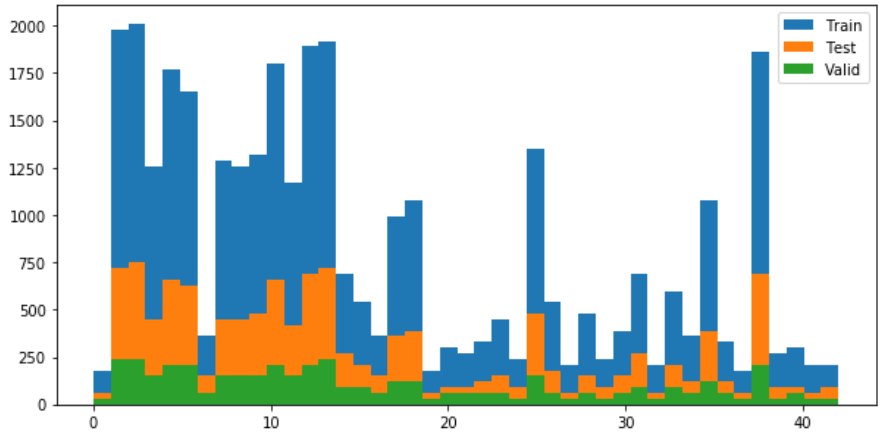
- The size of training set is 34,799
- The size of the validation set is 4,410
- The size of test set is 12,630
- The shape of a traffic sign image is (32, 32, 3)
- The number of unique classes/labels in the data set is 43

2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing the distribution of the classes in the training set sorted by frequency.



To examine the distribution of the classes distribution across different data sets (training, test, validation), I show them in a histogram, and it seems all the classes are evenly distributed across all the data sets.

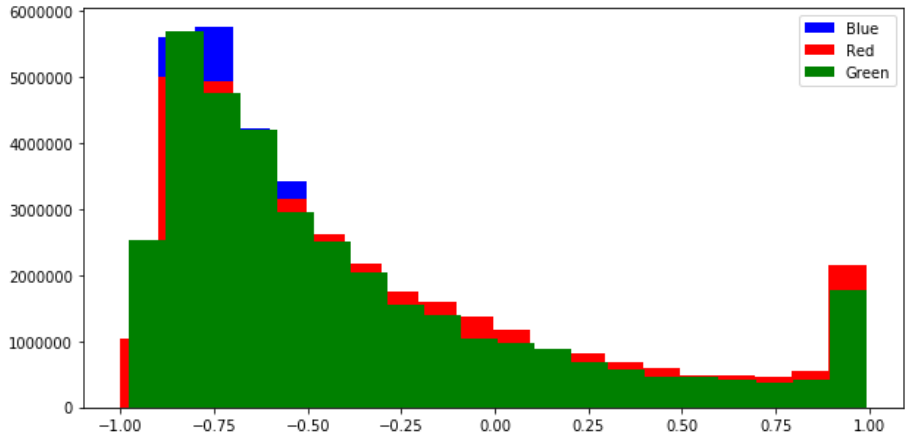


Design and Test a Model Architecture

1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques?

I decided not to convert the images to grayscale because the colors can include some useful information, and I hope the model can leverage those information.

NN models prefer normalized features, because it is easier to the optimizer to find a good solution. Therefore I normalized the image data using a simple formula, (X - 128.0) / 128.0. To check the distribution of the normalized values, I plot the histogram of each of the RGB bands. To my surprise, it is not really a normal distribution, but pretty skewed.



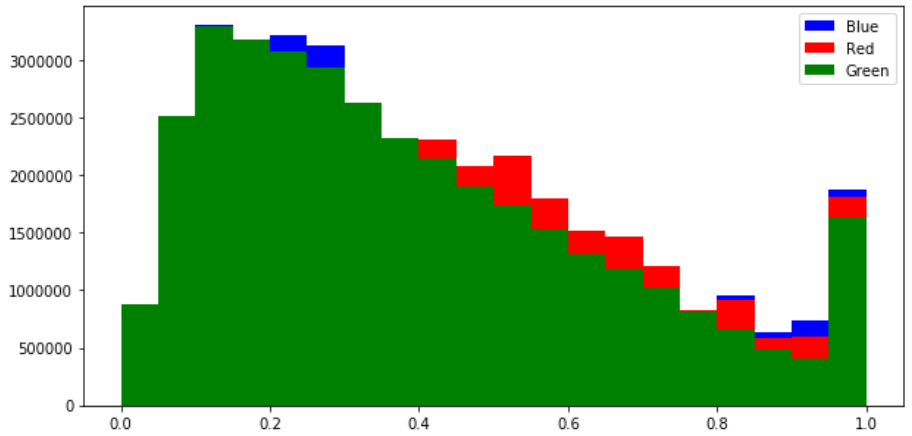
I checked the images and found many of them are very dark. I guess those images were taken from cameras in outdoor environment, so they usually look pretty dark due to the exposure. Below are some example images.

I tried to use some library to auto enhance the images so that they have better contrast, such as skimage.exposure.equalize_adapthist. You can find the difference below:

Before adjusting the contrast:

After adjusting the contrast:

It will also change the distribution of the pixel values.



However, the results didn't actually improve the model accuracy significantly, and it has some extra costs, such as I need to tune the clip_limit and it has to be applied to new images as well. To make the model more general, I still keep the original images without enhancement, and hope the model can generalize well for images with all kinds of exposure conditions.

2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.)

I adapted the LeNet with a dropout layer at the end as the final model. The 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, same padding, outputs 14x14x6
Convolution 5x5	1x1 stride, valid padding, outputs 10x10x16
RELU	
Max pooling	2x2 stride, same padding, outputs 5x5x16
Fully connected	Flatten and connect to 120 nodes. outputs 120
RELU	
Fully connected	Fully connect to 84 nodes. outputs 84
RELU	
Dropout	Dropout with keep_ratio 0.5
Logits	Fully connect to 43 output nodes. outputs 43
Softmax	

This model is very similar to the LeNet we implemented in the course lab. The only differences are:

- Input RGB with 3 bands instead 1-band gray images.
- Add a dropout at the last layer before logits.

3. Describe how you trained your model.

To train the model, I used the following setup:

- optimizer: Adam
- batch size: 128

- number of epochs: 100
- learning rate: 0.001 (I also tuned a few, and 0.001 can converge in a reasonable speed without loss of too much accuracy)

4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93.

My final model results were:

- training set accuracy of 0.998
- validation set accuracy of **0.951**
- test set accuracy of 0.928

Notes from the development:

- What was the first architecture that was tried and why was it chosen?
    - The first model doesn't have any regularization method, i.e., no dropout layer.
- What were some problems with the initial architecture?
    - The first model has very strong overfitting on the training data, and the validation set cannot achieve 0.93 accuracy, although it converges much faster.
- How was the architecture adjusted and why was it adjusted?
    - To reduce the overfitting, I tried to add dropouts in the network. I did some research and first tried to add 3 dropouts in the fully connected layers, then reduced it to 2 and 1. I found it is good enough even only use 1 dropout, therefore I only kept the last one.
- Which parameters were tuned? How were they adjusted and why?
    - I tuned batch size (64, 128, 256), and it didn't change too much.
    - I tuned number of epochs. It became pretty stable after 50 epoches, but to be consistent over different experiments, I used 100 epoches.
    - I also tuned the learning rate, such as 0.01, 0.001 and 0.0001. The smaller, the longer to run. If the learning rate is too big, the performance is not very stable. I found 0.001 is a good choice for both accuracy and speed.
    - I considered to change the kernel size of the convolution layers, but didn't really do it because 5x5 seems good enough now.
- 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?
    - Convolution layer work well because it can capture important spatial features of the signs. Dropout layer can help because it reduces the overfitting.
    - I also considered to use Incept-3, VGG, Resnet and transfer learning to train the model. But due to the time limit, I didn't really try them in this project.

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 8 German traffic signs that I found on the web:

Only the first one is downloaded from Google search. All the others are downloaded from the German Traffic Sign Dataset, but from another dataset (online competition test set) which should not have been used in the training.

The first image might be difficult to classify because it looks like a speed limit sign, but it is not from German (the board shape is a square instead of circle). I choose it to see how general this model can apply to similar traffic signs in other countries, or how much it depends on the shape of the board. All the others should look similar to the training set, but were not used in the training.

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 (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).

Here are the results of the prediction:

Image	Prediction	Correct?
Speed limit (50km/h)	Slippery road	No
No passing	No passing	Yes
No entry	No entry	Yes
General caution	General caution	Yes
Slippery Road	Slippery Road	Yes
Wild animals crossing	Wild animals crossing	Yes
Turn left ahead	Turn left ahead	Yes
Roundabout mandatory	Roundabout mandatory	Yes

The model was able to correctly guess 7 of the 8 traffic signs, which gives an accuracy of 87.5%. Only the first image was misclassified. As I mentioned, it is not really a German sign, so I guess the model depends a lot on the shape of the sign board (round vs. square makes a big difference).

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.

The code for making predictions on my final model is located in the second to the last cell of the Ipython notebook.

To my surprise, the top 5 softmax results are all the same: [1, 0, 0, 0, 0]!

```
TopKV2(values=array([
       [ 1.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
```

```
        [ 1.,  0.,  0.,  0.,  0.],
        [ 1.,  0.,  0.,  0.,  0.],
        [ 1.,  0.,  0.,  0.,  0.],
        [ 1.,  0.,  0.,  0.,  0.],
        [ 1.,  0.,  0.,  0.,  0.],
        [ 1.,  0.,  0.,  0.,  0.]], dtype=float32), indices=array([[31,  0,  1,  2,  3],
        [ 9,  0,  1,  2,  3],
        [17,  0,  1,  2,  3],
        [18,  0,  1,  2,  3],
        [23,  0,  1,  2,  3],
        [31,  0,  1,  2,  3],
        [34,  0,  1,  2,  3],
        [40,  0,  1,  2,  3]], dtype=int32))
```

To check what's going on, I also print the top 5 logits, and now I can see the difference:

```
TopKV2(values=array([
        [   843.19311523,    254.91714478,     14.66727829,  -1485.24060059,
          -1655.02880859],
        [ 12349.40722656,  -2805.61938477,  -3848.00415039,  -4046.9140625 ,
          -5570.66015625],
        [ 11339.296875  ,    805.45062256,  -1389.74169922,  -5665.57128906,
          -6301.11035156],
        [  4045.49194336,   -303.00888062,   -503.70346069,   -661.65734863,
          -1110.19787598],
        [ 10850.96777344,  -3129.26464844,  -4413.51904297,  -4654.375     ,
          -4954.00244141],
        [  8658.140625  ,  -3175.34179688,  -3478.6340332 ,  -4063.6418457 ,
          -4211.08349609],
        [  5341.04150391,    766.97418213,   -707.09759521,  -1371.35656738,
          -1419.49511719],
        [  1459.53259277,    252.11105347,     68.08274078,   -129.10275269,
           -423.4949646 ]], dtype=float32), indices=array([[31,  0,  1, 19,  3],
        [ 9, 16,  3, 12, 32],
        [17, 14, 30, 12, 26],
        [18, 36, 11, 26, 37],
        [23, 31, 20, 30,  3],
        [31, 22, 25, 23, 29],
        [34, 33, 38,  3, 11],
        [40, 32, 16,  1, 11]], dtype=int32))
```

The problem is the values of logits are already too big, so the softmax pushes the values to extreme (either very close to 1 or 0). I guess the problem is the weights in the nets (especially the fully connected layers) are too large. One possible solution is to use L2 regularization on the weights for the optimization to shrink the weights.