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

Data Set Summary & 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.

Step 0 Load The Data (Use Step 4)

Firstly, I imported all the necessary libraries.

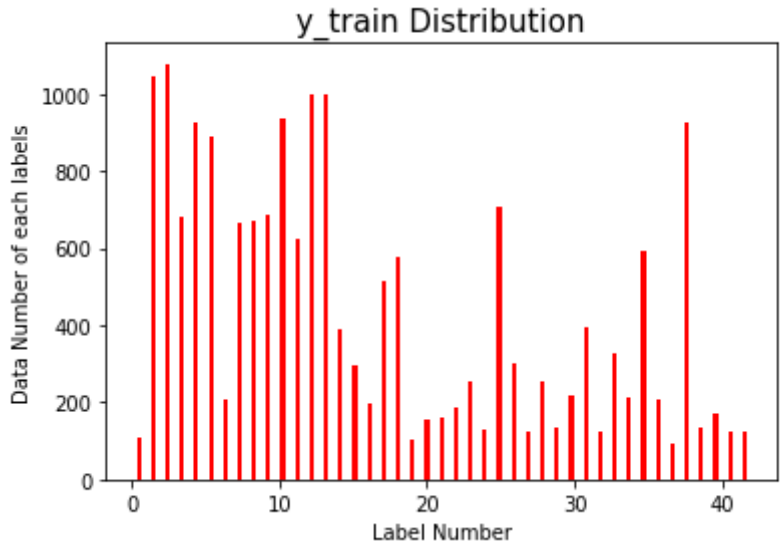
As the template shows, I opened training data, validation data, and test data from the default data directory. Although, I ended up not using it except the test data, which I will mention later.

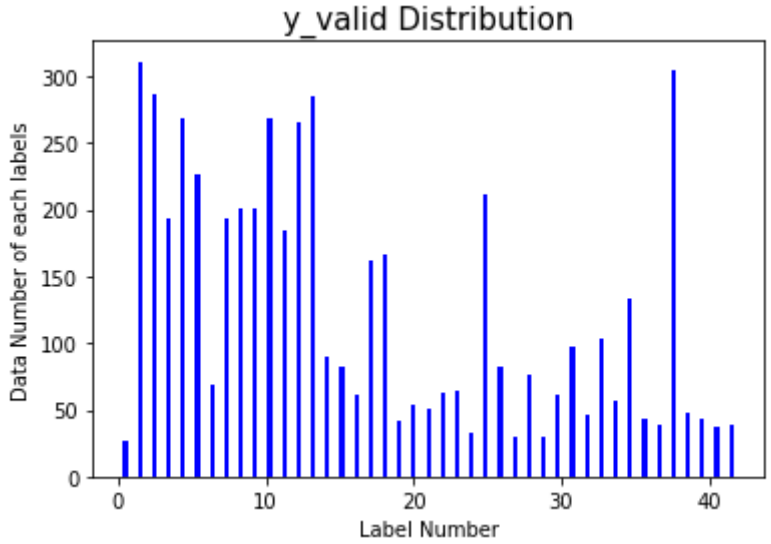
To use other data for this project, I downloaded it from https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/published-archive.html) , which was pointed out in readme file for the instruction of this project.
I used zipfile class to unzip the file, and then unfrozen it. The data is used for training data, validation data and test data. Dividing the data three parts with ratio of 70 %, 20 % and 10 % respectively, I split that data to; the values as X_data ; the labels as y_data.

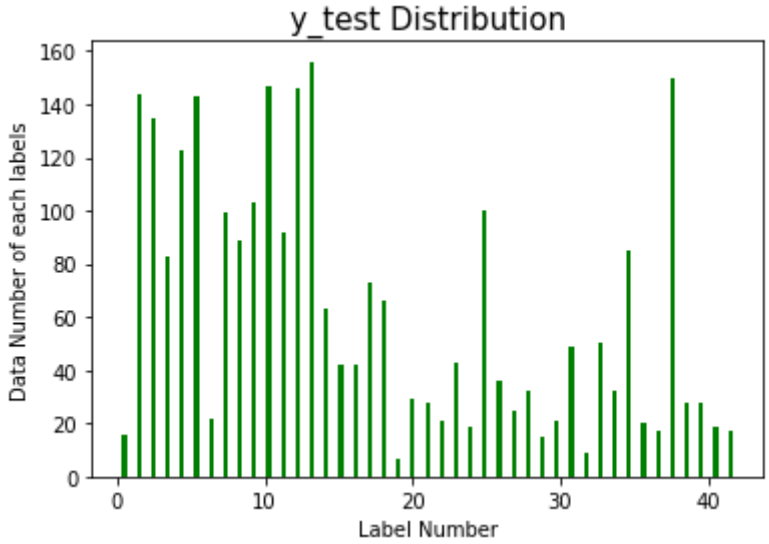
Step 1 Dataset Summary & Exploration
Exploring the data, I gained the following results about it.

* The size of training set is 18648
* The size of the validation set is 5328
* The size of test set is 2664
* The shape of a traffic sign image varies lots from about 27x28x3 to about 164x170x3.
* The number of unique classes/labels in the data set is 43

According to the way I split the data for training, validation and test, I yielded histograms for each to visualize the correspondence between the images and labels. I laid out the results below.







Here are some *examples* [1]of the input images. (They are all resized to 32x32x3, which I will talk about in the next step)

Step 2 Design and Test a Model Architecture

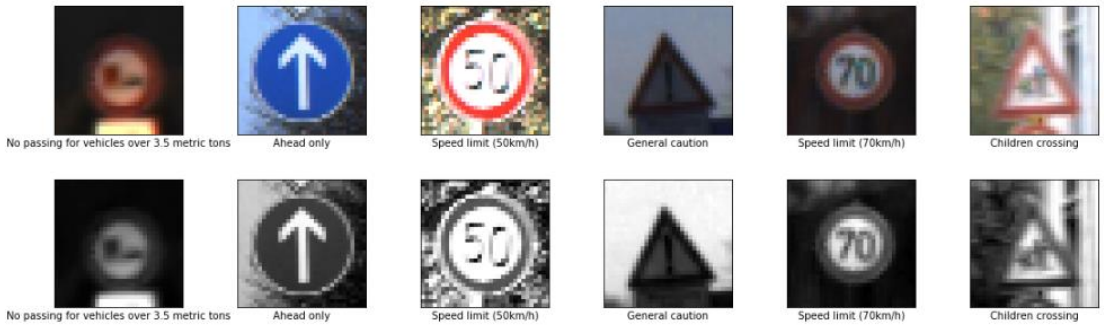
1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc.

Firstly, I resized all the images to 32x32x3, because they vary quite a bit as above. Then I converted the images to grayscales. The reason is as follows: considering the nature of the CNNs – to outline the feature of edges of images - the importance of the gradient is more predominant than the color. For example, I came across images from the dataset in RGB channels which were barely recognizable with bear eyes due to its low pixel values, but gray-scaling the image enabled to detect the gradients of these pixels' values such that the features of the images became a lot more obvious. Also, this process decreases computation through a

[1] These images were generated with the same code as my previous one, but are different compared to the ones in html.file because of the randomization. But this doesn't cause any trouble on the following codes.

network which results in a faster training over epochs.

Here is an *example*[2] of a traffic sign image before (same as the images example above) and after gray-scaling.



Secondly, I pre-processed the data using a standardization technic such that the data have a distribution with a mean value 0 and deviation value of 1. This is because the adjacent weights affect all the subsequent layers in the model, thus the small change of weights will be amplified through the network. More controlled distribution of the input layer is thus easier to compute, as the distribution with a wider range will be computationally more expensive and jumpier.

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.

My model architecture is as follows. I started building it on top of the LeNet architecture. I added batch normalization in convolutional layer, precisely between (convo and affin) and the activation function. I'll discuss the reason in the next chapter.

I initialized weights and biases in a manner of scandalization with the mean value of 0 and the sigma AKA variances of data. I listed the details of the model below.

---

[2] Same as the 1st footnote above.

Layer	Description
Input layer	32x32x1 Grayscale image
Convolution 5x5x1x6: layer 1	1x1 stride, same padding, output 28x28x6
Batch normalization: layer 1	outputs 28x28x6
RELU: layer 1	Outputs 28x28x6
Max pooling: layer 1	2x2 stride, outputs 14x14x6
Convolution 5x5x6x16: layer 2	output 10x10x16
Batch normalization: layer 2	output 10x10x16
RELU: layer 2	Output 10x10x16
Max pooling: layer 2	2x2 stride output 5x5x16
Flat: layer 3	Output 400
Fully connected: layer 4	output 120
Dropout[3]: layer 4	Output 120
Fully connected: layer 5	output 84
Fully connected: layer 6	Output 43

3. Describe how 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.

I trained the model using several different technics. My approach to train this model is stochastic gradient descent. Using weights as variables, I optimized the loss function of this network with the weights by examining gradient at each weight to the function. I initiated weights by using truncate function, and biases by using tensorFlow zeros functions. I also optimized the loss function with backpropagation and batch normalization(I'll talk about this later on). Batch size is 128, and the epoch number was 50 to ensure completing training. Learning rate is 0.001.

4. Describe the approach taken for finding a solution 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

[3] I ended up setting the ratio of dropout to 1, so this can be left out.

architecture. In this case, discuss why you think the architecture is suitable for the current problem.

I used LeNet architecture as a foundation, though I made a few of changes to make it suitable for the dataset I used in this project. Firstly, I changed the input layer to 32x32x1 because I used grayscale images to train, validate and test data which have only channel. I also used batch normalization because of the similar reason to pre-processing input data with standardization: that is, as each layer's weights adapt to the previous adjacent input layer, the change of weights closer to the input layer will be amplified during going through the multiple layers. This amplification can cause more computationally expensive process. On the other hand, if weights in each layer is regularized to a distribution with a mean value 0 and a variance 1, this effect of amplification decrease, thus would yield better result.

My final model results were:

* training set accuracy of 99.121%
* validation set accuracy of 97.241%

Here, I discuss some other possible architectures. As this architecture satisfies the required condition, I'd like to strip off some functions to evaluate how this affects on the accuracies.

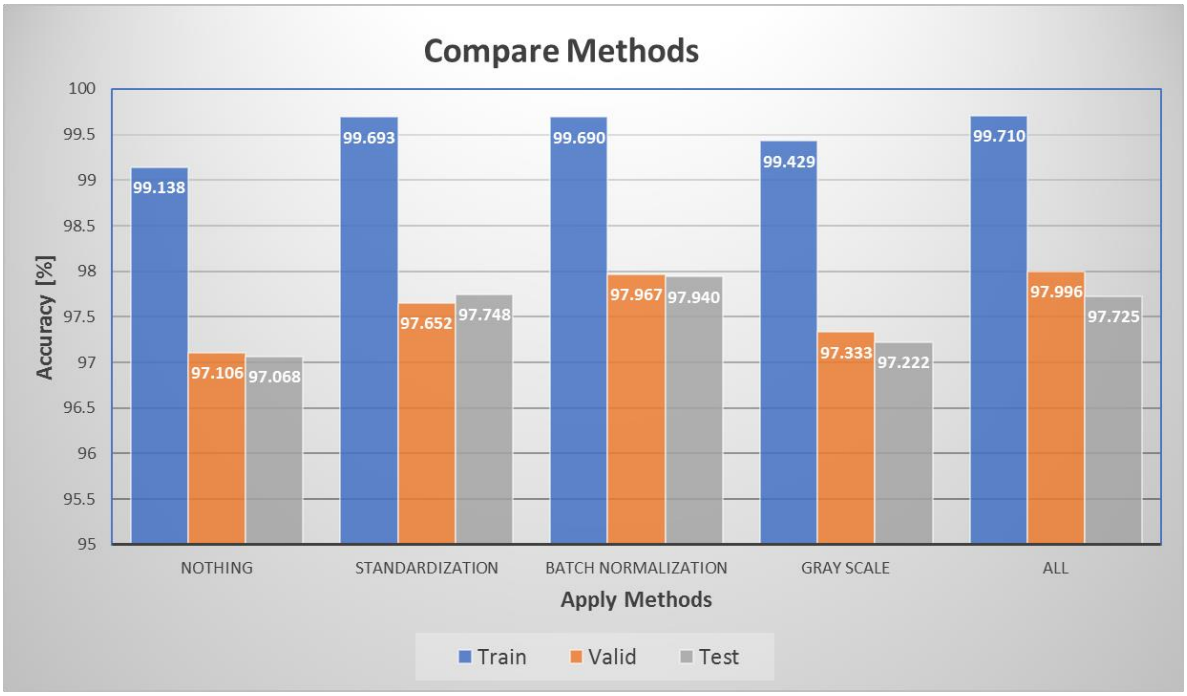
Firstly, I consider a case without batch normalization. As aforementioned, this will put more burden on a training. Also considering the mechanism of loss function, the shape becomes smoother with respect to weights with batch normalization. So otherwise, the function takes much more jumpy or rough shape such that the course of stochastic gradient descent will be more random thus hard to find its global minim.

Secondly, I consider a case without standardization. As I discussed the effect on this in pre-processing section, standardized distribution will yield more stable input dataset than otherwise.

Thirdly, I consider a case without grayscale. I mentioned the benefit of this process above, so this would yield worse results.

Finally, if all the steps are all omitted, theoretically this will yield the worst results (I labeled this as nothing in the graph below). I iterated each step for 10 times, stored the results of

accuracies and took the mean values for each step at the end. Here I put the results below.



As I expected, a model with all the functions – gray scale, standardization and batch normalization – yields the best result, and batch normalization itself achieves about the same result. Obviously, a step without any of the methods yields the worst result, but this still satisfies the required condition of 93 % accuracy.

Step 3: Test a Model on Test Data Set

Applying this model to the test dataset I prepared, I got the result of 97.110 %. Please refer to the number in the HTML code.

Step 4: Test a Model on New Images

After my resubmission, I gained 5 example images from the reviewer which I showed below. I cropped the data by using Snipping tool and resized it to 32x32 by using cv2.resize function. The data is from the random web images, and they all have essentially corresponding labels to train_dataset with some edgy information such as a graffiti, covering branches and so forth. Before testing how the model would classify the images, I did not give the labels to the images, therefore the images below don't have the labels yet.

I then gray scaled and applied standardization to them for the same reason I mentioned above.

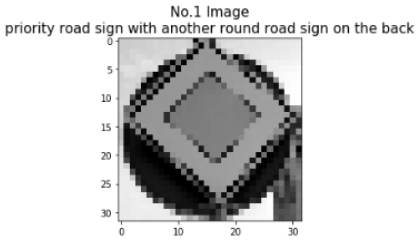
2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).
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.

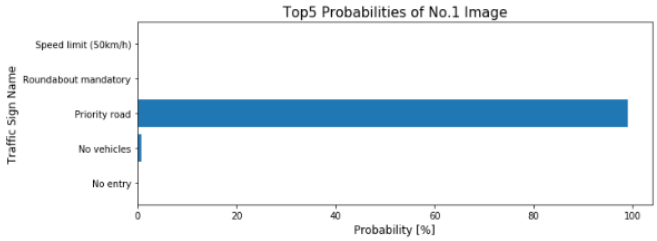
Here, I will address the 2 questions above at the same time below.

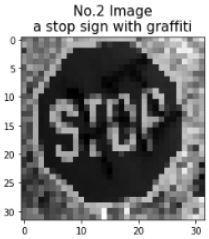
Firstly, I made my model predict the sign type for each image. I reloaded the model by the file path that I saved my model at, then I obtained the following result which shows the number of label on each image(This number corresponds to a certain label, and that is listed in signnames.csv file)

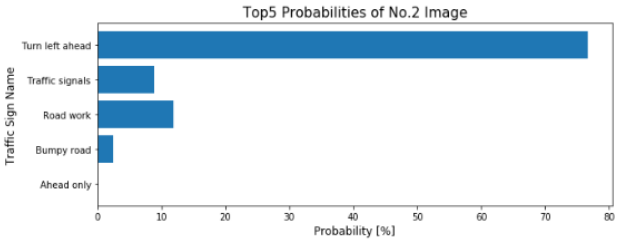
```
INFO:tensorflow:Restoring parameters from ./lenet_G
indices =       [12, 34, 1, 37, 1]
```

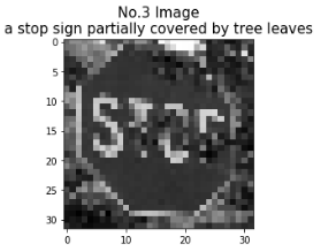
Now, I also output top 5 softmax probabilities to see how model predicts on the given images. Here is the result.

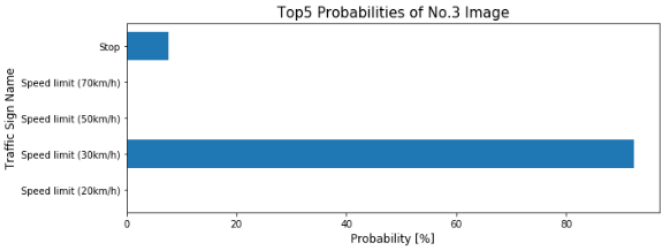


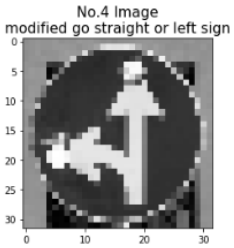


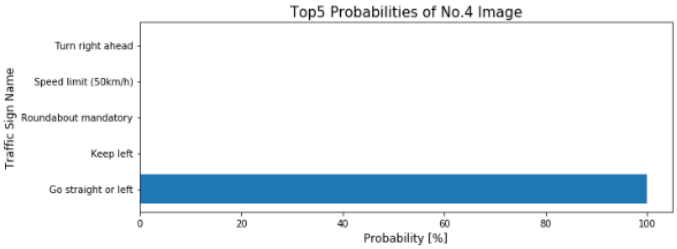


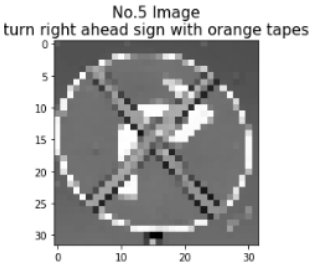


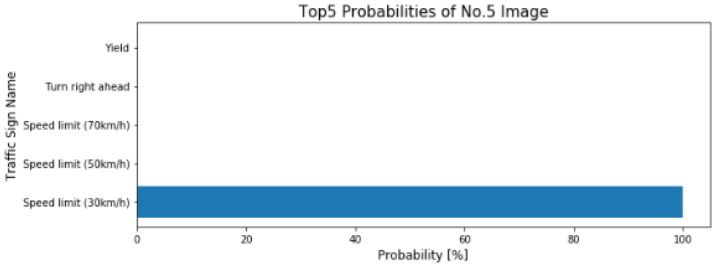












Overall, the model was able to predict the images with 40 % accuracy. As the graph above shows, some of the signs such as a priority road sign, go straight or left signs are predicted with high probabilities. On the other hand, stop signs and turn right ahead sign are misclassified. Although, for example, I can see the prediction on the third image has a low

probability on the stop sign. This indicates that either the given data is too noisy and thus indistinguishable, or the model is not robust enough for the edge cases, or it could even be both.

Analyze Performance

```
# Testing the 5 images above with my model
# Give labels of the examples
five_labels =[12,14,14,37,33]
acc = np.isclose(predict_indices, five_labels)
# calcurating the accuraty
acc_p = np.mean(np.float32(acc))
print("The certainty is {:.1%}".format(acc_p))
```

```
The certainty is 40.0%
```

Conclusion

The architecture and technics that I explained above enables to predict new traffic signs by using a large number of image dataset. As I discussed in the middle of the report, this approach reaches more than 97 % of accuracy on both validations sets and test sets. On the other hand, this model predicts poorly when given images with edgy information as I discussed in the last section. To combat this, including more edgy images to training data may help to build a more robust model.