**#Traffic Sign Recognition**

##Writeup Template

### It is easy for human naked eye to understand and to identify the traffic signs. On the other hand, for the self-driving car it is not easy. Therefore, the traffic sign recognition is important in the field of self-driving car. In order for self-driving car to understand the traffic condition and environment around it, the self-driving car must understand and recognize the traffic signs that post on the road or anywhere that the car navigate to.  If the self-driving car could not recognize the traffic signs, it would be unsafe and very dangerous and the self-driving car is not applicable on the road and in the real world.

This project is about the traffic sign classification in helping self-driving car to identify the traffic signs. This project is based on machine learning and CNN. Start with preparing the training data and building the model to testing the real traffic sign, this project is aim to make sure the applicable of self-driving car in the real world.

**Build a Traffic Sign Recognition Project**

The steps of this project are the following:

- Load the 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. In this writeup, I decided to use pdf file. I also used this writeup as a readme file in github project. You're reading it! and here is a link to my project code

### Data Set Summary & Exploration

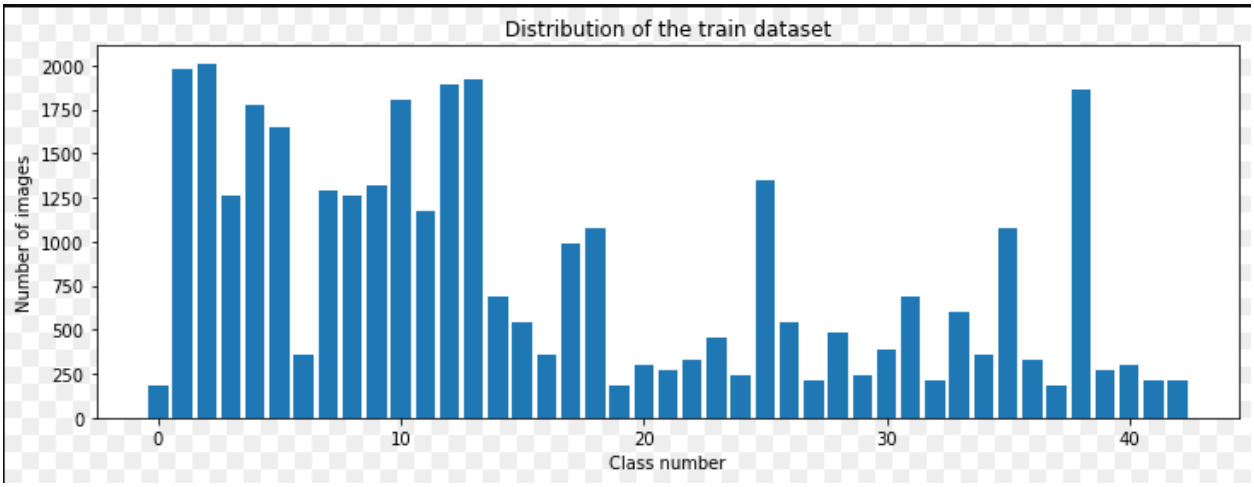
####1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

I used the numpy library to calculate the dataset. Here is the summary of the traffic signs data set:

- The size of training set is 34799
- The size of the validation set is not provided so I will split it from the training set later.
- The size of test set is 12630
- 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 how the data is distribute for each class off traffic signs:

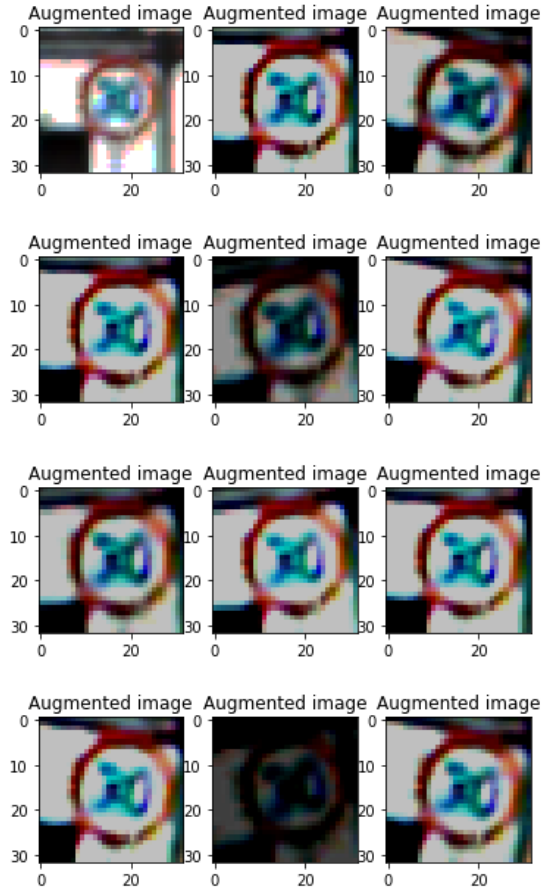



###Design and Test a Model Architecture

####1. Firstly, I used original dataset for training, validating, and testing. The result is not satisfied because the percentage of accuracy is around 80 percent. Then I try to create more fake data by creating more images per classes that has images less than 500. Using augment_img() function to change image contrast, size, color, and transform image from the

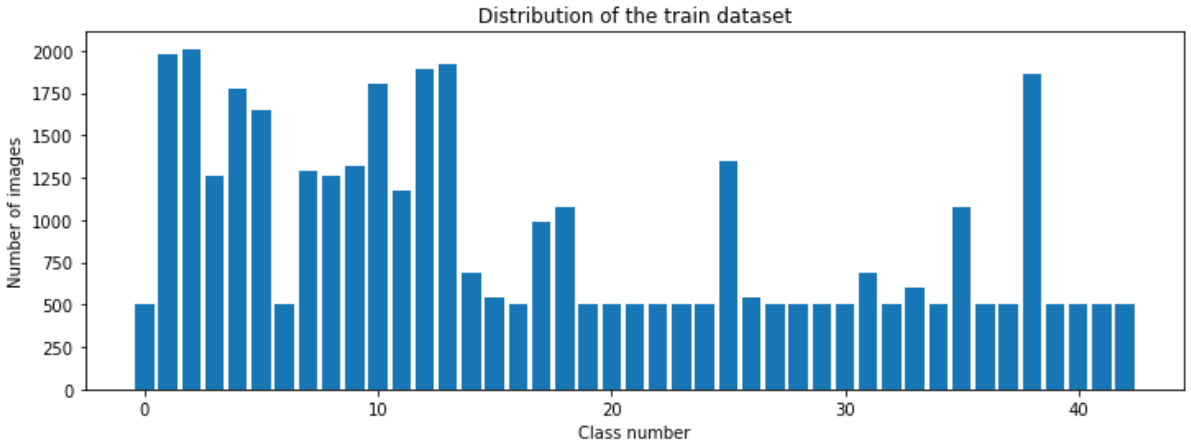
original attribute to different attribute. The result still could not satisfy the requirement because the accuracy is below 90 percent. Finally, I decide to convert the images to grayscale and normalize the dataset; and I see the improvement on the result and I could get the accuracy above 93 percent.

Here is an example of a traffic sign images that have additionally created for classes that have less than 500 images.



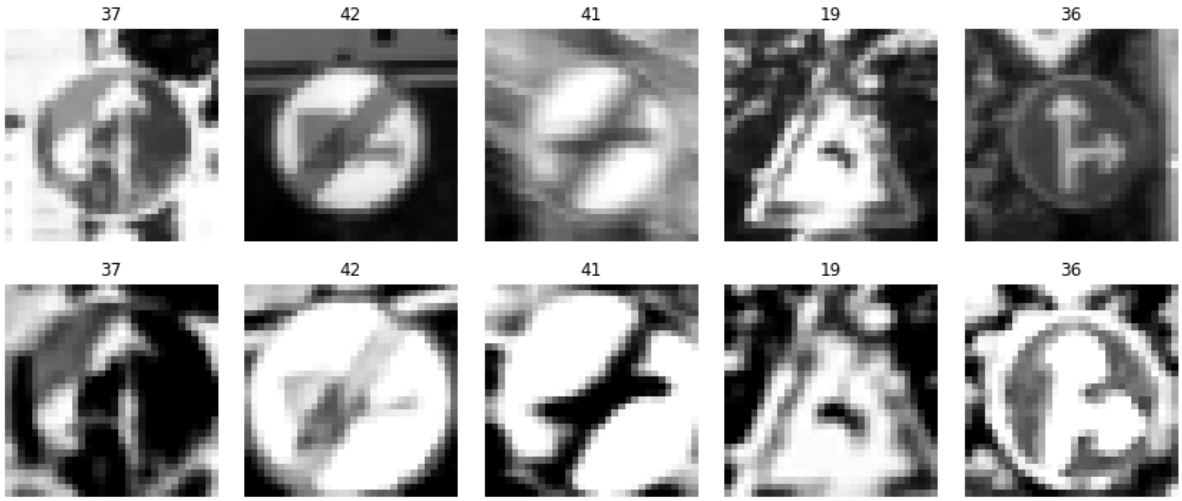

```
Original image size is (32, 32, 3)
Final image size is (32, 32, 3)
```

This is the new dataset




- The new size of training set is 39239

Below first line of images are the converted grayscale images and the bottom line is the normalized image:




####2. The model based on LeNet model with some tweak.

My final model consisted of the following layers:

| **Layer** | **Description** |
|---|---|
| Input | 32x32x1 Grayscale image |

| Layer | Description |
|---|---|
| Convolution 5x5 | 1x1 stride, VALID padding, outputs 28x28x6 |
| RELU | |
| Max pooling | 2x2 stride, VALID padding, outputs 5x5x16 |
| Convolution 5x5 | 1x1 stride, VALID padding, outputs 1x1x400 |
| ReLu | |
| Flatten layers | Concatenate flattened layers |
| Dropout layer | |
| Fully connected layer | 800 in, 42 out |

####3. This model is based on LeNet model. As it always, using the combination of the right parameters would be critical for model to perform well and improve accuracy. In this training, I have used the following parameters:

- Batch size = 100
- Epochs = 25
- mu = 0
- sigma = 0.1
- rate = 0.001
- This model is using AdamOptimizer

####4. To get accuracy to be at lease 0.93, I have used LeNet model with some customizations. While using original LeNet, the accuracy is below 0.9. With some customization, the accuracy is above 0.93.

Comparing Original LeNet with new Model

| LeNet | Customized LeNet |
|---|---|
| # Arguments used for tf.truncated_normal, randomly defines variables for the weights and biases for each layer<br>mu = 0<br>sigma = 0.1 | # Hyperparameters<br>mu = 0<br>sigma = 0.1 |
| # SOLUTION: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.<br>conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 6), mean = mu, stddev = sigma))<br>conv1_b = tf.Variable(tf.zeros(6))<br>conv1 = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv1_b | # TODO: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.<br>W1 = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 6), mean = mu, stddev = sigma), name="W1")<br>x = tf.nn.conv2d(x, W1, strides=[1, 1, 1, 1], padding='VALID')<br>b1 = tf.Variable(tf.zeros(6), name="b1")<br>x = tf.nn.bias_add(x, b1)<br>print("layer 1 shape:",x.get_shape()) |
| # SOLUTION: Activation.<br>conv1 = tf.nn.relu(conv1) | # TODO: Activation.<br>x = tf.nn.relu(x) |
| # SOLUTION: Pooling. Input = 28x28x6. Output = 14x14x6.<br>conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID') | # TODO: Pooling. Input = 28x28x6. Output = 14x14x6.<br>x = tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')<br>layer1 = x |
| # SOLUTION: Layer 2: Convolutional. Output = 10x10x16.<br>conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean = mu, stddev = sigma))<br>conv2_b = tf.Variable(tf.zeros(16))<br>conv2 = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VALID') + conv2_b | # TODO: Layer 2: Convolutional. Output = 10x10x16.<br>W2 = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean = mu, stddev = sigma), name="W2")<br>x = tf.nn.conv2d(x, W2, strides=[1, 1, 1, 1], padding='VALID')<br>b2 = tf.Variable(tf.zeros(16), name="b2")<br>x = tf.nn.bias_add(x, b2) |
| # SOLUTION: Activation.<br>conv2 = tf.nn.relu(conv2) | # TODO: Activation.<br>x = tf.nn.relu(x) |
| # SOLUTION: Pooling. Input = 10x10x16. Output = 5x5x16.<br>conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID') | # TODO: Pooling. Input = 10x10x16. Output = 5x5x16.<br>x = tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')<br>layer2 = x |
| | |

| | |
|---|---|
| # SOLUTION: Flatten. Input = 5x5x16. Output = 400.<br>fc0 = flatten(conv2) | |
| # SOLUTION: Layer 3: Fully Connected. Input = 400. Output = 120.<br>fc1_W = tf.Variable(tf.truncated_normal(shape=(400, 120), mean = mu, stddev = sigma))<br>fc1_b = tf.Variable(tf.zeros(120))<br>fc1 = tf.matmul(fc0, fc1_W) + fc1_b | # TODO: Layer 3: Convolutional. Output = 1x1x400.<br>W3 = tf.Variable(tf.truncated_normal(shape=(5, 5, 16, 400), mean = mu, stddev = sigma), name="W3")<br>x = tf.nn.conv2d(x, W3, strides=[1, 1, 1, 1], padding='VALID')<br>b3 = tf.Variable(tf.zeros(400), name="b3")<br>x = tf.nn.bias_add(x, b3) |
| # SOLUTION: Activation.<br>fc1 = tf.nn.relu(fc1) | # TODO: Activation.<br>x = tf.nn.relu(x)<br>layer3 = x |
| # SOLUTION: Layer 4: Fully Connected. Input = 120. Output = 84.<br>fc2_W = tf.Variable(tf.truncated_normal(shape=(120, 84), mean = mu, stddev = sigma))<br>fc2_b = tf.Variable(tf.zeros(84))<br>fc2 = tf.matmul(fc1, fc2_W) + fc2_b | # TODO: Flatten. Input = 5x5x16. Output = 400.<br>layer2flat = flatten(layer2)<br>print("layer2flat shape:",layer2flat.get_shape()) |
| | # Flatten x. Input = 1x1x400. Output = 400.<br>xflat = flatten(x)<br>print("xflat shape:",xflat.get_shape()) |
| # SOLUTION: Activation.<br>fc2 = tf.nn.relu(fc2) | |
| # SOLUTION: Layer 5: Fully Connected. Input = 84. Output = 10.<br>fc3_W = tf.Variable(tf.truncated_normal(shape=(84, 10), mean = mu, stddev = sigma))<br>fc3_b = tf.Variable(tf.zeros(10))<br>logits = tf.matmul(fc2, fc3_W) + fc3_b<br><br>return logits | # Concat layer2flat and x. Input = 400 + 400. Output = 800<br>x = tf.concat([xflat, layer2flat], 1)<br>print("x shape:",x.get_shape()) |
| | # Dropout<br>x = tf.nn.dropout(x, keep_prob) |
| | # TODO: Layer 4: Fully Connected. Input = 800. Output = 43.<br>W4 = tf.Variable(tf.truncated_normal(shape=(800, 43), mean = mu, stddev = sigma), name="W4")<br>b4 = tf.Variable(tf.zeros(43), name="b4")<br>logits = tf.add(tf.matmul(x, W4), b4)<br><br>return logits |

My final model results were:

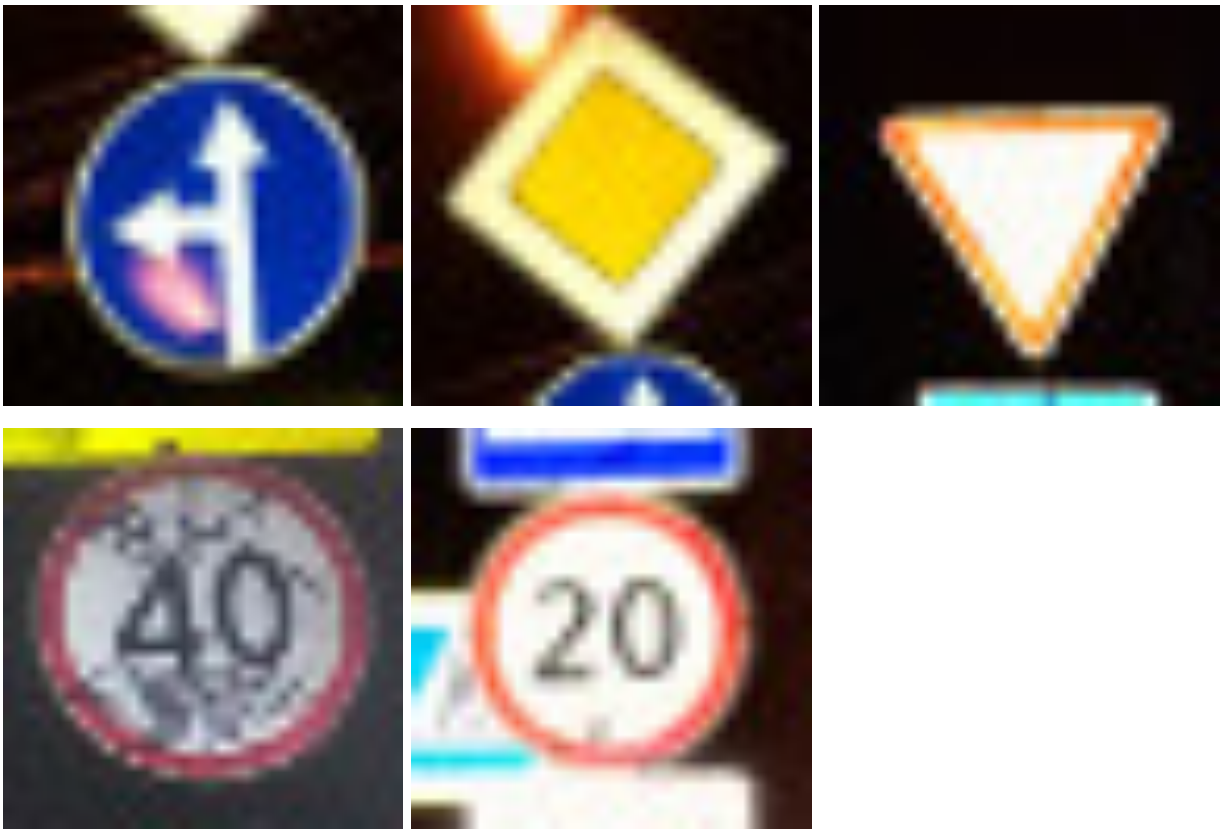
- validation set accuracy of 0.988
- test set accuracy of 0.935

If a well known architecture was chosen:

- I used LeNet for the architecture of this training. On the other hand, I could not get the accuracy above 0.9 while using the original LeNet. I have to customize some of the function in order to improve the model accuracy.
- Follow by the instruction from the lesson in the classroom; I think that LeNet is a good model that can be used for the traffic sign classification.

###Test a Model on New Images

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



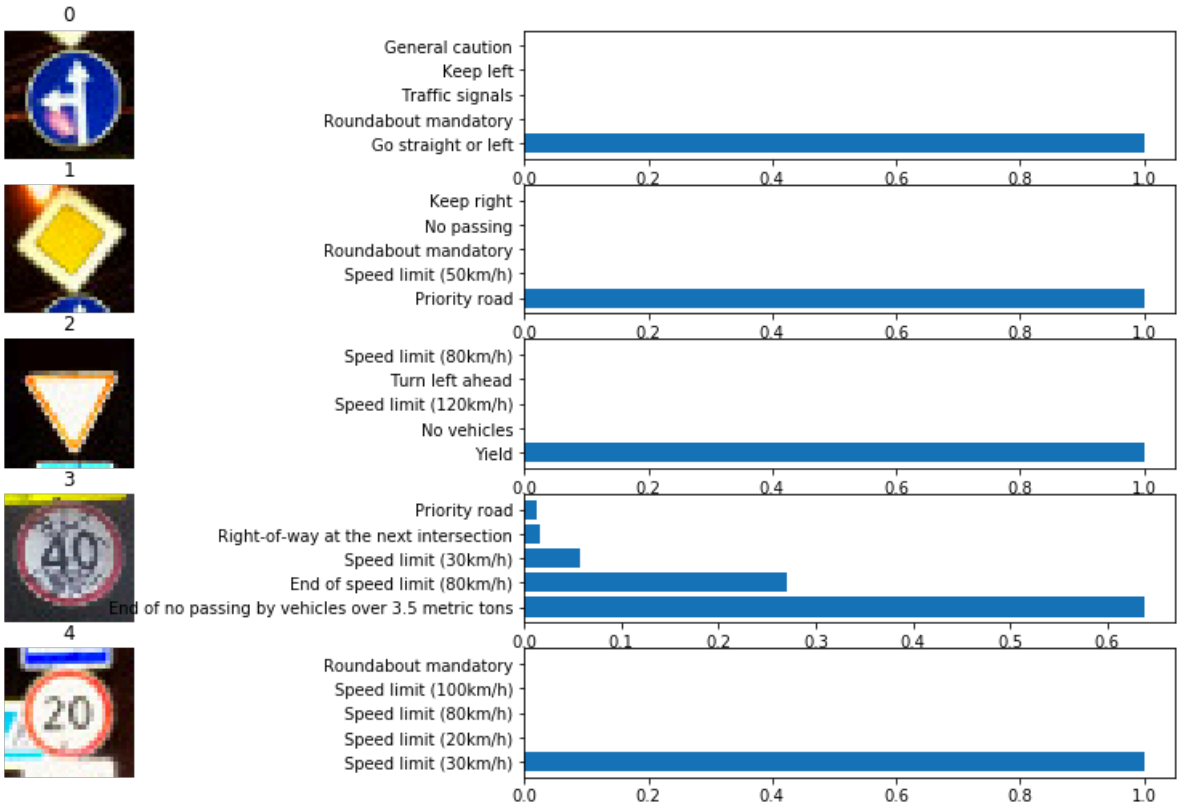

The fourth image is difficult to classify because it doesn't have in any class in the training dataset. Therefore, the training doesn't know what class is the sign would be in.

####2. The model can predict four of the five images correctly. Therefore, the accuracy of the prediction is 80% compare to the training set of 93.4%. The fourth image is outside of the classes in the training set, therefore, it make sense that the model could not predict it while we doesn't train it to recognize this image.

Here are the results of the prediction:

| **Image** | **Prediction** |
|---|---|
| Go straight or left | Go straight or left |
| Priority road | Priority road |
| Yield | Yield |
| Speed limit 40 km/h | End of no passing by vehicle over 3.5 metric tons |
| Speed limit 20 km/h | Speed limit 30 km/h |

#### 3. For these real traffic sign images, the model predicts 60% correct. Only fourth and fifth image that are images that the model did not predicted accurately; Picture below shows how the result of the model's prediction to real traffic signs.




For the first image, the model is relatively sure that this is a Go straight or left (probability of 1), and the image do contain a Go straight or left sign. The top five soft max probabilities were

| **Probability** | **Prediction** |
|---|---|
| 1 | Go straight or left |
| 1 | Priority road |
| 1 | Yield |
| 1 – Wrong prediction | End of no passing by vehicles over 3.5 metric tons |
| 1 – Wrong prediction | Speed limit (30km/h) |

Summary:

In the field of self-driving car, traffic sign classification would be the first step to make sure that the car does follow the law. In this project, the prediction of traffic sign accuracy is about 93% but in the real world this result is not applicable to implement while the error is two large to take risk. This mean that there are sill many more traffic sign that the self-driving car could not recognize so the car could not make the decision in responding to the traffic signs and the situation. There are still many more to improve and to learn in this area in order to make the model work to get more accuracy to which we can implement in the real world situation. These improvements could be providing more training data, remodeling the model, augmenting the images and tweaking some more parameter what could improve accuracy of the prediction.