Readme for the Self-Driving Cars Nanodegree Program 3rd Project Traffic Sign Recognition

Goals / Steps of this project:

- 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.

Rubric Points

Dataset Exploration

Dataset Summary

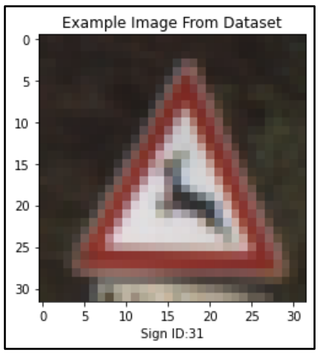
- German Traffic Sign Dataset was used to train the model used for the Traffic Sign Recognition.

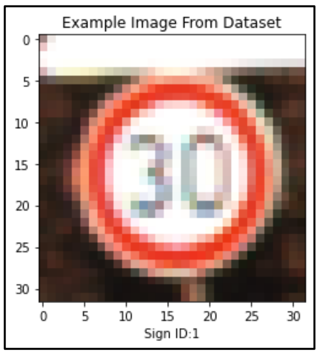
Size of **Training** set is:	34,799
Size of **Validation** set is:	4,410
Size of **Testing** set is:	12630
Image data shape is:	(32, 32, 3)
Number of Classes:	43*

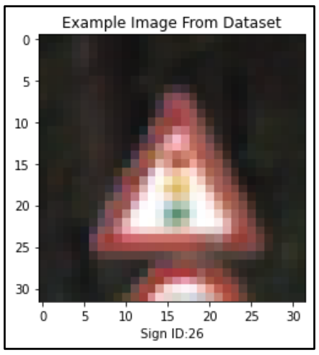
*The different Classes identification can be seen in the "signnames.csv" included in the project delivery.

Exploratory Visualization

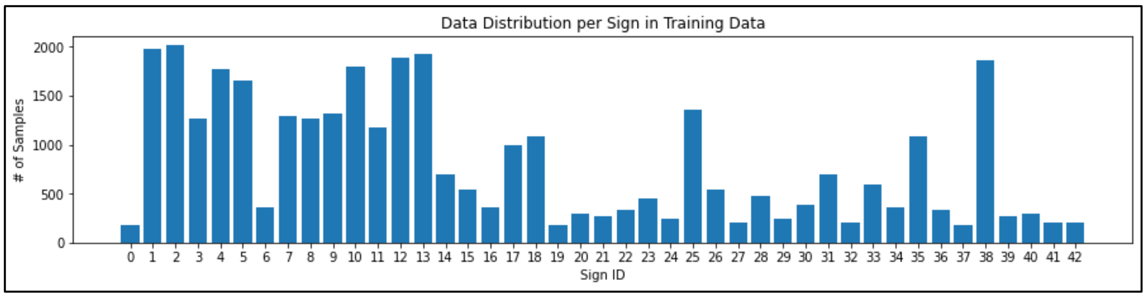
- The Following images were taken from the dataset.

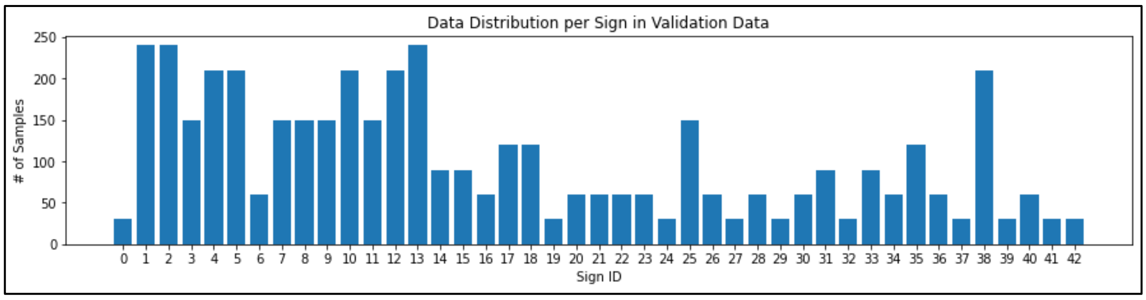


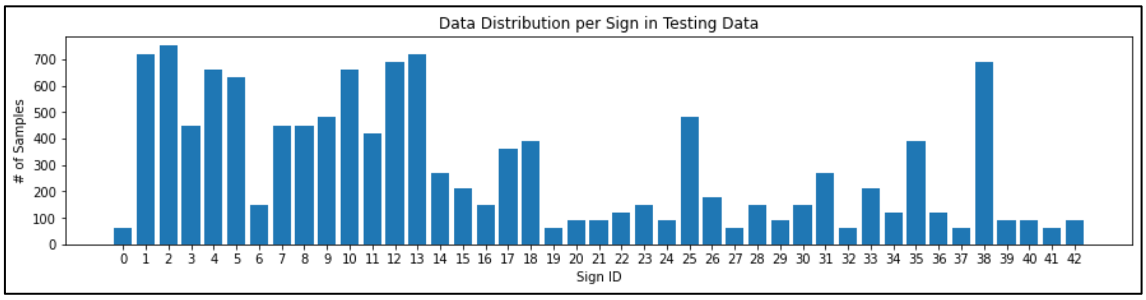




- In the following charts is shown how the training, validation and testing data was divided.







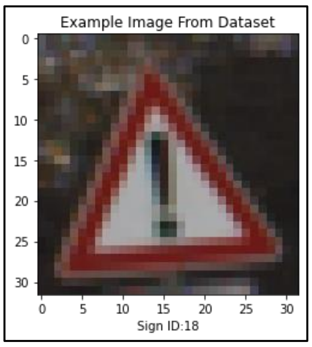
Design and Test a Model Architecture

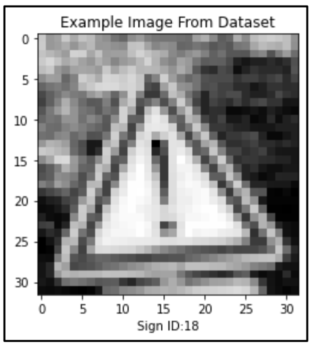
<u>Pre-processing</u>

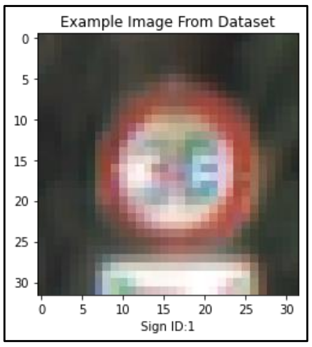
- The pre-processing of the data was done in 4 steps:
    - <u>Shuffle the training data</u>: This step is done to increase randomness and variety in the training dataset.
    - <u>Convert Images to Grayscale</u>: This step is done to all the input data. The use of grayscale images instead of colour ones improves the Convnets accuracy.
    - <u>Apply Local Histogram Equalization</u>: This step increases the contrast of the images. This is helpful since the images are taken from real world and many of them have low contrast.
    - <u>Normalize the Data</u>: Once the Images are converted to grayscale and the depth is just 1, I normalize the image by dividing every pixel [0,255] by 255; the result is a value between [0,1].

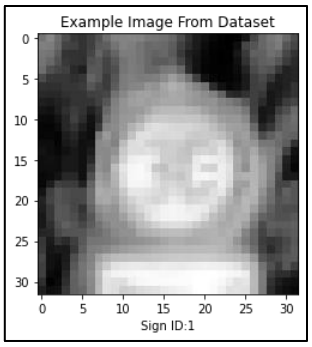
Note: In the code a function to increase the dataset can be found “add_rotate”. This Functions takes random images from the dataset and randomly rotates it and adds it again to the dataset. Finally, this function was not used since during the test I saw it did not contribute too much to the CNN Accuracy.

- The following images are examples of Traffic signs before/after being pre-processed.









Model Architecture

- The table describes the layers of the Model.

Layer	Description
Input	Grayscale Image 32x32x1
Convolution	Stride: 1x1, Padding: Valid, Output: 28x28x6
RELU	
Max Pooling	Stride: 2x2, Padding: Valid, Output: 14x14x6
Convolution	Stride: 1x1, Padding: Valid, Output: 10x10x16
RELU	
Max Pooling	Stride: 2x2, Padding: Valid, Output: 5x5x16
Flatten	Output: 400
Fully-Connected	Output: 120
RELU	
Dropout	Training keep_prob: 0.5
Fully-Connected	Output: 84
RELU	
Dropout	Training Keep_prob: 0.5
Fully-Connected	Output: 43

Model Training

- Hyperparameters
  - Epochs: 30
  - Batch_size: 128
  - Learning rate: 0.001

- To train the model, the the LeNet CNN was used. Modifications in the output layer; and dropouts were added in the fully connected layers to avoid overfitting since it is a serious problem in deep neural networks.
  - The output was modified to give an output of 43 classes instead of ten as used during the exercises of the course.

- The final model results were the following:

Training set accuracy:	0.992
Validation set accuracy:	0.951
Testing set accuracy:	0.933

- The following piece of code was used to train the network:

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    num_examples = len(X_train)

    print("Training...")
    print()
    for i in range(EPOCHS):
        X_train, y_train = shuffle(X_train, y_train)
        for offset in range(0, num_examples, BATCH_SIZE):
            end = offset + BATCH_SIZE
            batch_x, batch_y = X_train[offset:end], y_train[offset:end]
            sess.run(training_operation, feed_dict={x: batch_x, y: batch_y, keep_prob: 0.5})

        training_accuracy = evaluate(X_train, y_train)
        validation_accuracy = evaluate(X_valid, y_valid)
        print("EPOCH {} ...".format(i+1))
        print("Training Accuracy = {:.3f}".format(training_accuracy))
        print("Validation Accuracy = {:.3f}".format(validation_accuracy))
        print()

    saver.save(sess, './lenet')
    print("Model saved")
```

Note: During training the values for Dropouts was keep_prob = 0.5 while for the evaluation was 1.0.

```
###Model Evaluation
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_y, 1))
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
saver = tf.train.Saver()

def evaluate(X_data, y_data):
    num_examples = len(X_data)
    total_accuracy = 0
    sess = tf.get_default_session()
    for offset in range(0, num_examples, BATCH_SIZE):
        batch_x, batch_y = X_data[offset:offset+BATCH_SIZE], y_data[offset:offset+BATCH_SIZE]
        accuracy = sess.run(accuracy_operation, feed_dict={x: batch_x, y: batch_y, keep_prob: 1.0})
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples
```

Solution Approach

- The first step I took was to reuse the LeNet Model designed during the course.
    - Modifications to the input and output layers were done to fit the input images (RGB) and the output (43 Classes).
    - The Validation Accuracy was around 87%, not so far from the requested Accuracy that was 93%.
- During the process to get the requested accuracy I tried many things.
    - Normalize the data with the following formula (pixel-128)/128 getting pixels with values from [-1,1].
    - Using Sigmoid Activations instead of RELU; I tried this because of the normalized data [-1,1]. RELU might not be the best activation if the inputs can be negative.
    - Augment data to the dataset with the same images but randomly rotated.
    - Grayscale input.
    - Add Dropouts to all the layers (including Convolutions).
- Some of the accuracies that I got are:
    - Just adding Dropouts to all the layers: 0.637
    - Normalize dataset, Add Dropouts and augment 20% the dataset: 0.654

    - Grayscale the input: 0.895
    - Grayscale, Normalize data [0,1]: 0.911
    - **Grayscale, Normalize data [0,1], Dropout (Just Fully-connected layers): 0.930**
        - This was the first try that fulfilled the requested Validation Accuracy.
    - **Grayscale and Local Equalization, Normalize data [0,1], Dropout (Just Fully-connected layers): 0.951**
        - This one was the selected architecture (previously explained).

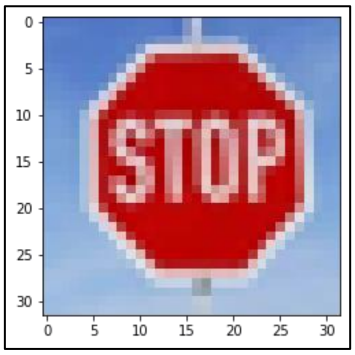
Note: I tried around 20 different models before obtaining a good result.

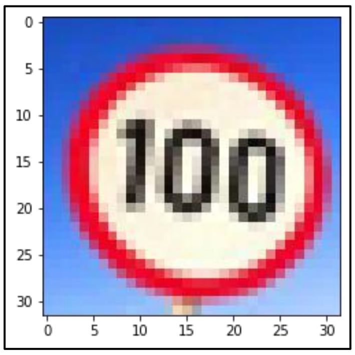
- During this tries I made a deep research to find what could improve my model, two tips that I consider very valuables from this research are:
    - When a CNN is overfitting, Dropouts are a good option to apply but just in the fully-connected layers.
    - CNN responds better to Grayscale images than Colour ones.

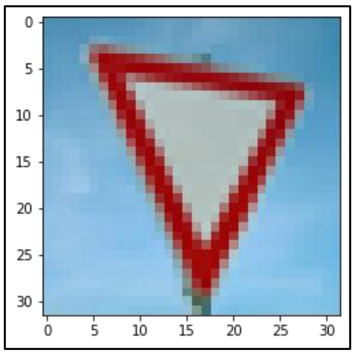
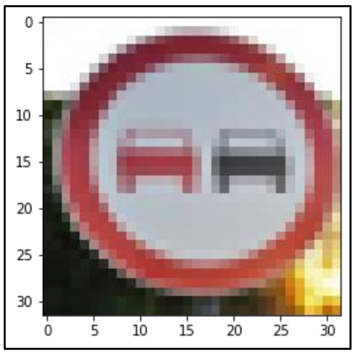
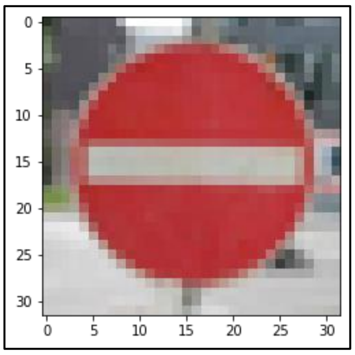
Test a Model on New Images

<u>Acquiring New Images</u>

- The selected images from the web are shown below:





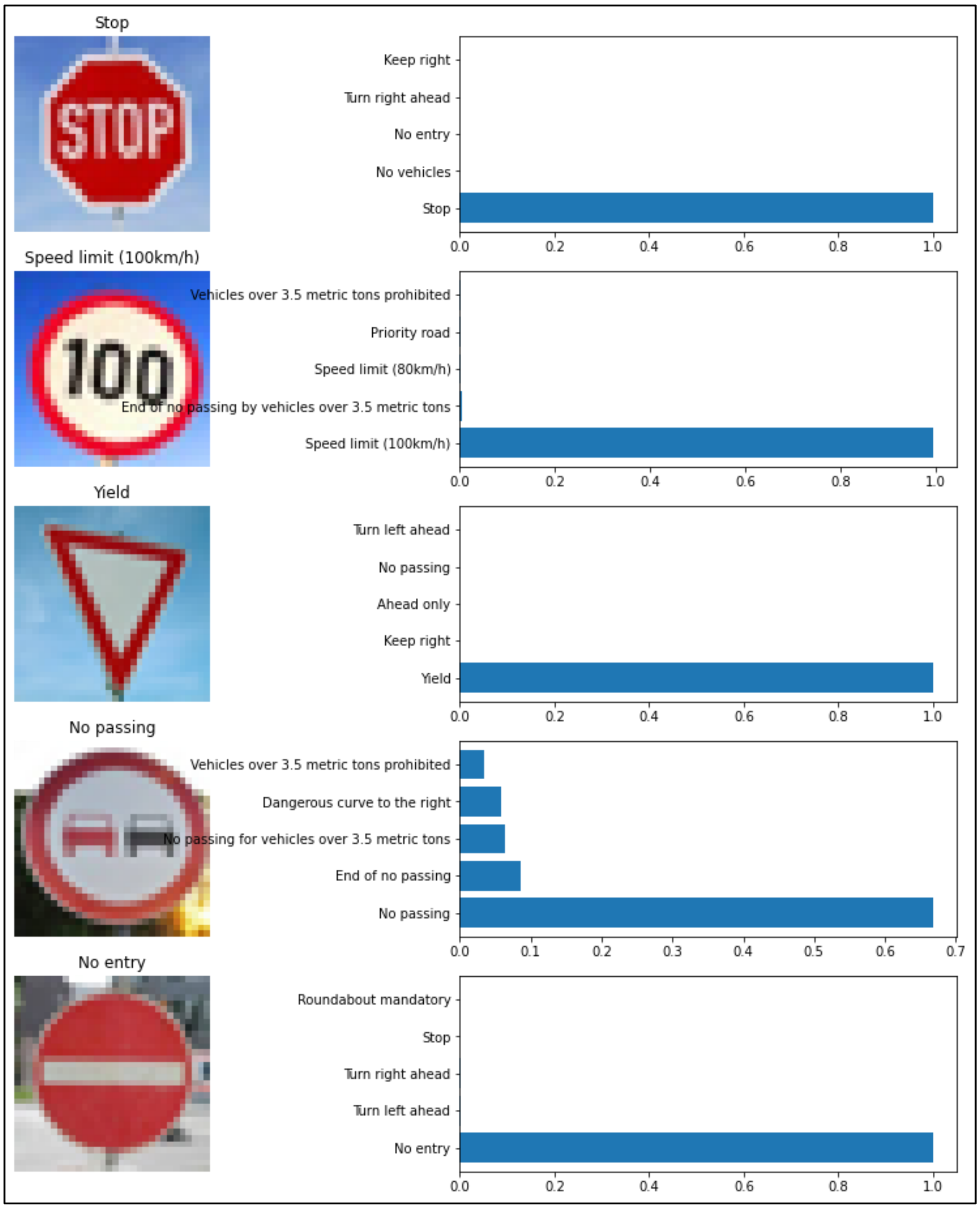
- Before testing the model with the previous images, I thought it would be difficult to correctly classify the "speed limitation" and the "No Passing" images.
    - The 100km/h limitation sign because it is very similar to all the signs that limit the speed.
    - The No Passing sign because the background of the picture is not very clean.

Performance on New Images

- The model was abled to correctly guess the 5 traffic signs, which gives an accuracy of 100%.
    - The Test set accuracy was: 93.3%

Model Certainty – Softmax Probabilities

- The top five softmax probabilities of the predictions on the New Images are shown below.
    - For 4 out of 5 images, the model is relatively sure of the sign, as guessed the "No Passing" Sign makes the model wonder a little bit since the are other traffic signs that are similar, like "End of no Passing" and "No Passing for vehicles over 3.5 metric tons".


Stop
Keep right
Turn right ahead
No entry
No vehicles
Stop
Speed limit (100km/h)
Vehicles over 3.5 metric tons prohibited
Priority road
Speed limit (80km/h)
End of no passing by vehicles over 3.5 metric tons
Speed limit (100km/h)
Yield
Turn left ahead
No passing
Ahead only
Keep right
Yield
No passing
Vehicles over 3.5 metric tons prohibited
Dangerous curve to the right
No passing for vehicles over 3.5 metric tons
End of no passing
No passing
No entry
Roundabout mandatory
Stop
Turn right ahead
Turn left ahead
No entry