Introduction

The goals / steps of this project are the following:

1) Load the data set
2) Explore, summarize and visualize the data set
3) Design, train and test a model architecture
4) Use the model to make predictions on new images
5) Analyze the softmax probabilities of the new images
6) Summarize the results with a written report

The project is graded according to these Rubric Points, which have been addressed in my project. A table of contents for the rubric point is below:

Rubric Point Table of Contents

i) Submission Files

The project submission includes all required files.

- Ipython notebook with code ("Traffic_Sign_Classifier.ipynb")
- HTML output of the code ("Traffic_Sign_Classifier.html")
- A writeup report (*This document*)

ii) Dataset Exploration

Dataset Summary

The submission includes a basic summary of the data set.

The code for the dataset summary is in "Step 1: Dataset Summary & Exploration", under the "Provide a Basic Summary of the Data Set Using Python, Numpy and/or Pandas" sub-section.

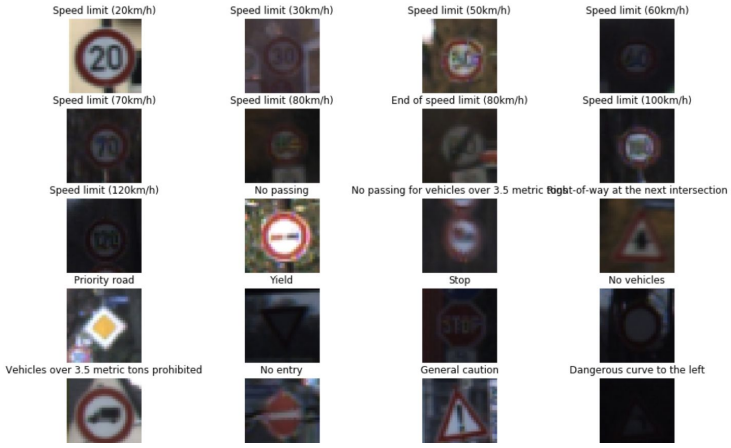
The output (summary) is:

```
# of training examples = 34799
# of validation examples = 4410
# of testing examples = 12630
Image data shape = (32, 32, 3)
# of classes = 43
```

Exploratory Visualization

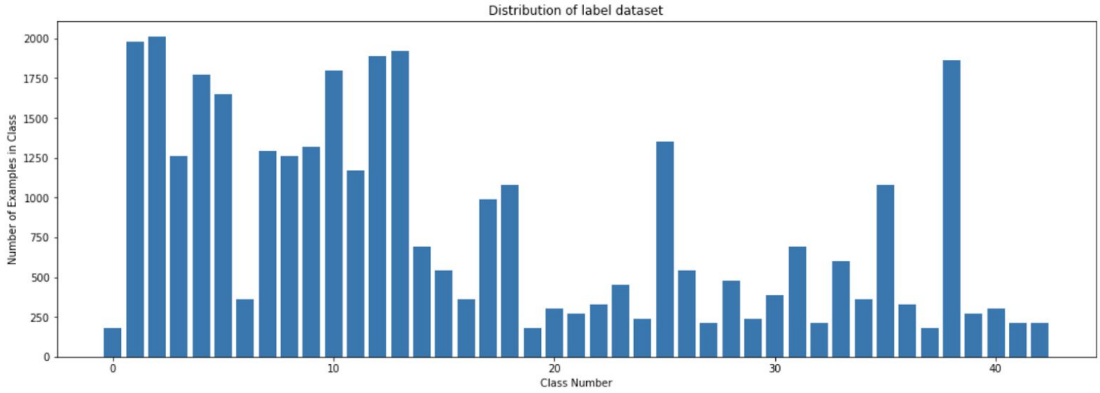
The submission includes an exploratory visualization on the dataset.

The code for the exploratory visualization of the data set is in "Step 1: Dataset Summary & Exploration", under the "Include an exploratory visualization of the dataset" sub-section. The output is:



In addition, I inspect the training data using a histogram of the labels:

```
Min # of images per class = 180
Max # of images per class = 2010
```



iii) Design and Test a Model Architecture

1. Preprocessing

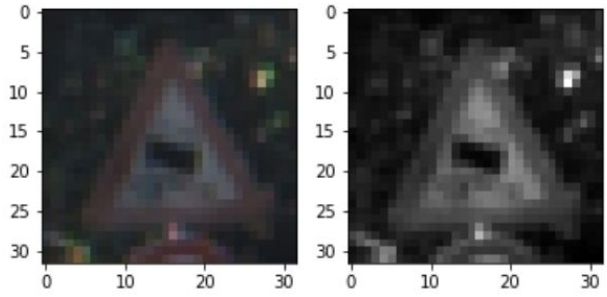
The submission describes the preprocessing techniques used and why these techniques were chosen.

The code to preprocess the image is in the section “Step 2: Design and Test a Model Architecture”, under the “Pre-process the Data Set (normalization, grayscale, etc.)” sub-section.

In my pipeline, we transform the image to grayscale and ignore colors in this problem because signs in our training set are differentiable from their contents and shapes. In addition, I resize the images to 32 x 32.

Here is a before and after of my image preprocessing pipeline:

```
Out[6]: <matplotlib.image.AxesImage at 0x7fb517e27ef0>
```



2. Model Architecture

The submission provides details of the characteristics and qualities of the architecture, including the type of model used, the number of layers, and the size of each layer. Visualizations emphasizing particular qualities of the architecture are encouraged.

The code defining the model architecture is in the section “Step 2: Design and Test a Model Architecture”, under the “Model Architecture” sub-section. I define helper functions for creating a convolution layer (*conv_layer()* function) and linear layer (*linear_layer() function*).

The model architecture is an Adapted LeNet with 2 x convolutional layers, followed by 1 x flatten layer and 3 x fully connected linear layers.

The Detailed Architecture is:

- Layer 1 - convolution 1: 32x32x1 -> 28x28x12 -> relu -> 14x14x12 (pooling)
- Layer 2 - convolution 2: 14x14x12 -> 10x10x25 -> relu -> 5x5x25 (pooling)
- Layer 3
    - Flatten: 5x5x25-> 625
    - Drop out: 625 -> 625
    - Linear layer 1: 625 -> 300
- Layer 4 - Linear layer 2: 300 -> 150
- Layer 5 - Linear layer 3: 150 -> 43

The code is here:

```
def classifier_CNN_model(input, keep_prob):
    ## Layer1: convolution layer: 32x32x1 -> 28x28x12 + Pooling
    conv1 = conv_layer(input, 32, 1, 28, 12)
    conv1 = tf.nn.relu(conv1)
    # pooling: 28x28x12 -> 14x14x12
    conv1 = tf.nn.max_pool(conv1, ksize=[1,2,2,1], strides=[1,2,2,1], padding='VALID')

    ## Layer2: Convolution layer 14x14x12 -> 10x10x25 + Pooling
    conv2 = conv_layer(conv1, 14, 12, 10, 25)
    conv2 = tf.nn.relu(conv2)
    # pooling: 10x10x25 -> 5x5x25
    conv2 = tf.nn.max_pool(conv2, ksize=[1,2,2,1], strides=[1,2,2,1], padding='VALID')

    ## Layer3: Dropout + 1st linear layer
    flat = tf.contrib.layers.flatten(conv2)
    dropped = tf.nn.dropout(flat, keep_prob)
    layer3 = linear_layer(dropped, 625, 300)
    layer3 = tf.nn.relu(layer3)

    ## Layer4: 2nd linear layer
    layer4 = linear_layer(layer3, 300, 100)
    layer4 = tf.nn.relu(layer4)

    ## Layer5: 3rd linear layer
    layer5 = linear_layer(layer4, 100, n_classes)
    return layer5
```

3. Model Training

The submission describes how the model was trained by discussing what optimizer was used, batch size, number of epochs and values for hyperparameters.

The code for the training of the model is in the section “Step 2: Design and Test a Model Architecture”, under the “Train, Validate and Test the Model” sub-section.

I trained the model using 10 epochs, batch size of 64 and learning rate of 0.001. I used the Adam Optimizer for the loss algorithm.

The code for training the model is located here:

```
#Set training parameters
epoch = 10
batch_size = 64
rate = 0.001

#Build training pipeline
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=one_hot_y)
loss = tf.reduce_mean(cross_entropy)
train_step  = tf.train.AdamOptimizer(learning_rate=rate).minimize(loss)

# Fuction to train model in TF
def train_model(X_data, y_data):
    #Run Tensorflow
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        print("------Begin training-----")
        for i in range(epoch):
            X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.1, random_state=int(time.t
            start_time = time.time()
            for offset in range(0, len(X_train), batch_size):
                end = offset + batch_size
                features, labels = X_train[offset:end], y_train[offset:end]
                sess.run(train_step, feed_dict={x : features, y : labels, keep_prob: 0.8})
            validation = evaluate_model(X_test, y_test)
            print("[{3:.1f}s] epoch {0}/{1}: validation = {2:.3f}".format(i+1, epoch, validation, time.time()-start_ti

        saver.save(sess, model_file);
        print("model saved into {}".format(model_file))
        print("------Finish training-----")

# Run the training functoin on training data
train_model(features_train, labels_train)
```

```
------Begin training-----
[3.6s] epoch 1/10: validation = 0.862
[3.0s] epoch 2/10: validation = 0.946
[2.9s] epoch 3/10: validation = 0.950
[2.9s] epoch 4/10: validation = 0.980
[2.9s] epoch 5/10: validation = 0.983
[2.9s] epoch 6/10: validation = 0.991
[2.9s] epoch 7/10: validation = 0.987
[2.9s] epoch 8/10: validation = 0.995
[2.9s] epoch 9/10: validation = 0.997
[2.9s] epoch 10/10: validation = 0.998
model saved into ./model
------Finish training-----
```

4. Solution Approach

The submission describes the approach to finding a solution. Accuracy on the validation set is 0.93 or greater.

The code for evaluating the model is in the section “Step 2: Design and Test a Model Architecture”, under the “Train, Validate and Test the Model” sub-section. The code is:

```
In [13]: # evaluate model with the validate set
         with tf.Session() as sess:
             saver.restore(sess, model_file)
             accuracy = evaluate_model(features_valid, labels_valid)
             print("Model accuracy using validation set: {:.3f}".format(accuracy))

         INFO:tensorflow:Restoring parameters from ./model
         Model accuracy using validation set: 0.939

In [14]: #Evaluate model with test set (only done at the end)
         with tf.Session() as sess:
             saver.restore(sess, model_file)
             accuracy = evaluate_model(features_test, labels_test)
             print("Model accuracy using test set: {:.3f}".format(accuracy))

         INFO:tensorflow:Restoring parameters from ./model
         Model accuracy using test set: 0.930
```

My final model performs with:

- Training set accuracy of 0.998
- Validating set accuracy of 0.939
- Test set accuracy of 0.930

To get to this final model, I started with the LeNet architecture since it has shown to be able to detect handwriting well. Using the original LeNet architecture, my test set accuracy was much lower (88-90% range). To improve this accuracy, I increase the filter depth in the first 2 layers to 12 and 25 respectively. This increased accuracy to 91-93% range.

To prevent overfitting, I added a dropout layer - which improved my test set accuracy to the 93%+ range.

iv) Test a Model on New Images

1. Acquiring New Images

The submission includes five new German Traffic signs found on the web, and the images are visualized. Discussion is made as to particular qualities of the images or traffic signs in the images that are of interest, such as whether they would be difficult for the model to classify.

I found eight images of traffic signs from the web. The images were a mix of natural background (i.e., in the real world) and white/black backgrounds as I wanted to test the generalizability of my model. The signs are fairly well lit and front-facing, meaning that they should be easy to detect by the model.

The code for importing and preprocessing my images are found in the “Step 3: Test a Model on New Images” section, under the “Load and Output the Images” sub-section.

The displays of my signs are below in the softmax probabilities section.

2. Performance on New Images

The submission documents the performance of the model when tested on the captured images. The performance on the new images is compared to the accuracy results of the test set.

The code for making predictions and analyzing the performance are found in the “Step 3: Test a Model on New Images” section, under the “Predict the Sign Type for Each Image” and “Analyze Performance” sub-sections.

The output for the prediction code is here:

```
#Run function
predict(my_features, True)
```

```
INFO:tensorflow:Restoring parameters from ./model
```

```
Out[32]: ['Stop',
          'General caution',
          'Yield',
          'Priority road',
          'Ahead only',
          'Speed limit (30km/h)',
          'Speed limit (60km/h)',
          'Speed limit (50km/h)']
```

The output for the analysis code is here:

```
INFO:tensorflow:Restoring parameters from ./model
Model accuracy is 0.875
```

The accuracy of my model on new images is 0.875, which is lower than the 0.93 on the test set. This can be a sign of underfitting and high variance (but low-ish bias). However, the difference between 0.875 and 0.93 is small enough that the variance may not be statistically significant. This is because I only have 8 new test images.

3. Model Certainty - Softmax Probabilities

The top five softmax probabilities of the predictions on the captured images are outputted. The submission discusses how certain or uncertain the model is of its predictions.

The code for visualizing the top 5 softmax probabilities are found in the “Step 3: Test a Model on New Images” section, under the “Output Top 5 Softmax Probabilities For Each Image Found on the Web” section.

The output for the code is here:

Looking at this code, I see that the code struggled with the 50 speed limit sign - suggesting that texts are difficult to grasp by the model. The other predictions have high certainty.