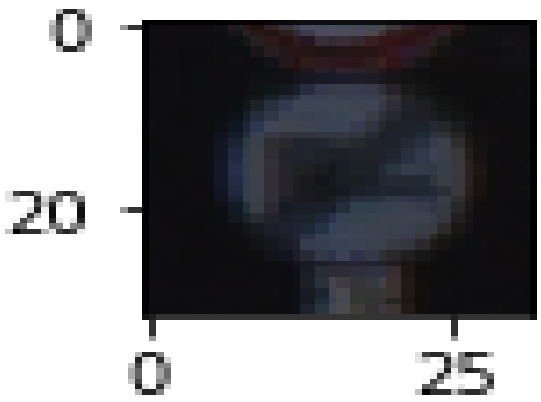
In order to do this problem we needed to first get the data of traffic sign images. This was obtained as a .p file of German traffic signs. It was already split into three files a train, test and validate. Since the data was already sized as a 32 x 32 that portion of the program was not need to be done.

CODE:

```
import random

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline


index = random.randint(0, len(X_train))

image = X_train[index].squeeze()


plt.figure(figsize=(1,1))

plt.imshow(image)

print(y_train[index])
```

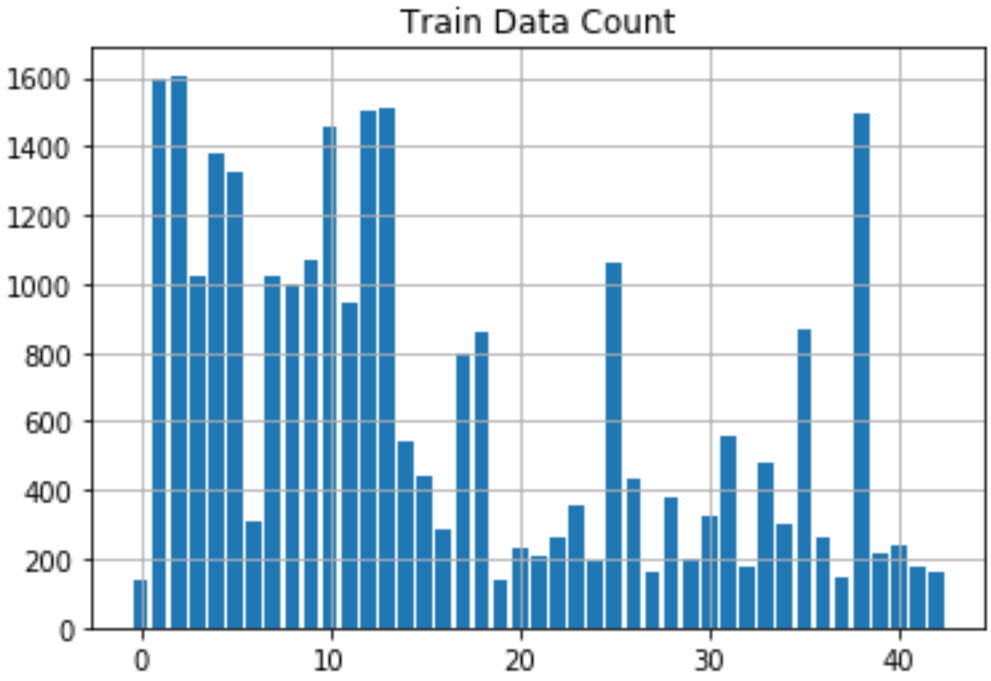


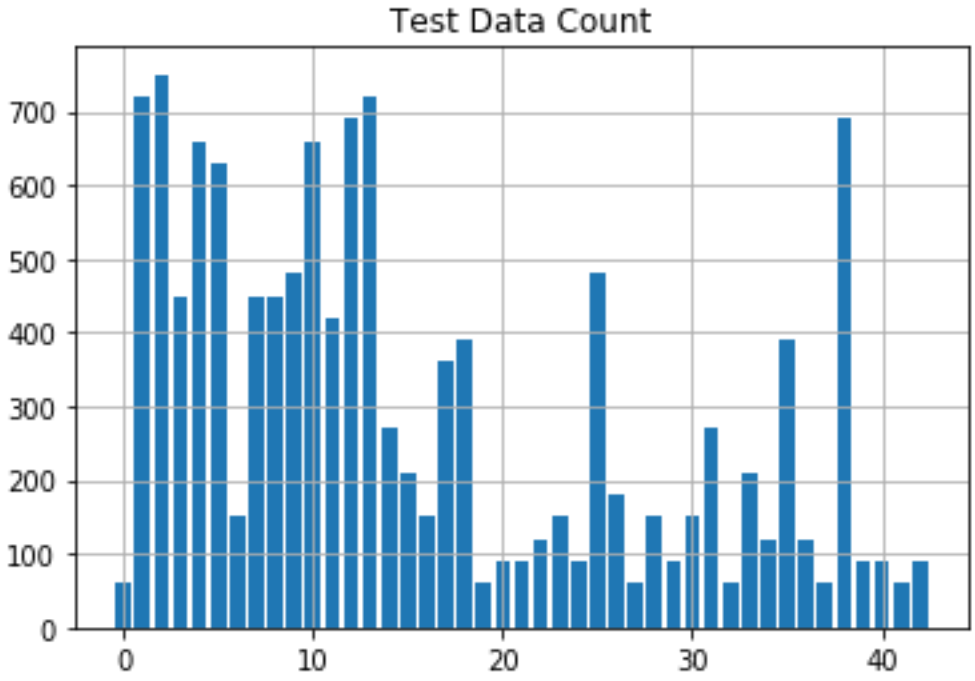
There are charts of the training data count were we see how many images of each class there were in the training set. We can also see this same information in the test and validation sets. As you can see by the charts they are closely related so there should be very little issues with this data sets. Since all three are show approximately the same information give or take a little data.

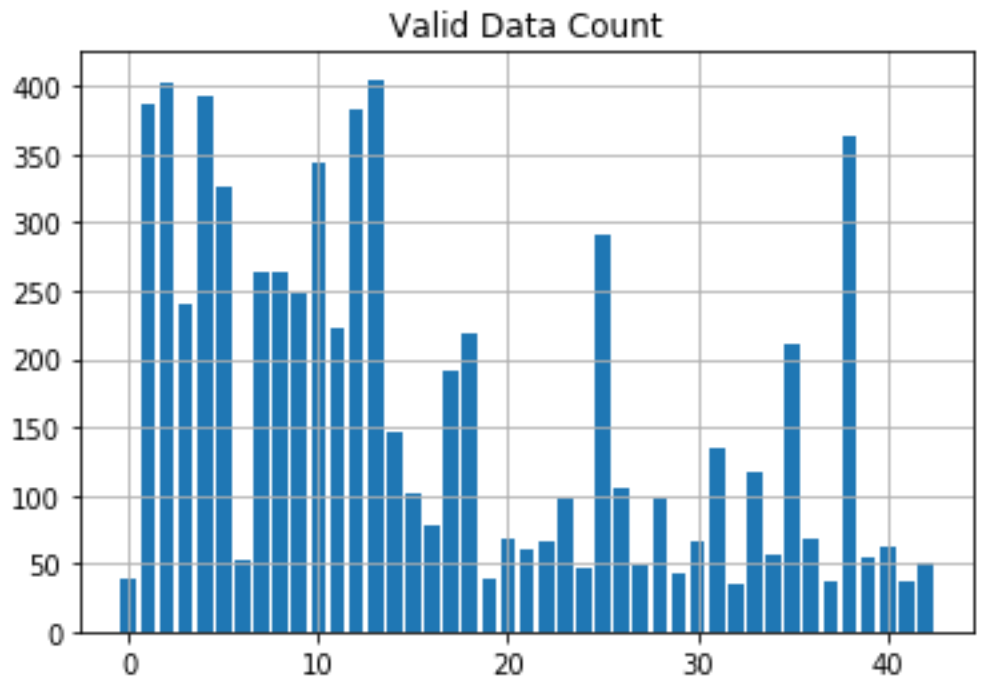
CODE:

```
import numpy as np
```

```
from sklearn.model_selection import train_test_split
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,test_size = 0.2, random_state=0)

print("Data Set size : {}".format(X_train.shape[0]))

unique, counts = np.unique(y_train, return_counts=True)
plt.bar(unique, counts)
plt.grid()
plt.title("Train Data Count")
plt.show()
unique, counts = np.unique(y_test, return_counts=True)
plt.bar(unique, counts)
plt.grid()
plt.title("Test Data Count")
plt.show()
unique, counts = np.unique(y_valid, return_counts=True)
plt.bar(unique, counts)
plt.grid()
plt.title("Valid Data Count")
plt.show()
```


Train Data Count
1600
1400
1200
1000
800
600
400
200
0
0
10
20
30
40


Test Data Count
700
600
500
400
300
200
100
0
0
10
20
30
40



The model used for this problem was LeNet so in order to prepare the data the first step was to convert the images to grayscale and then normalize the image. This model was created using tensor flow with a EPOCHS of 30 and a BATCH_SIZE of 30 the mu and sigma values were standard as used in the examples in class. The LeNet model uses the relu activation with the following layers. Layer 1 is a convolutional layer with an input of 32 x 32 x 3 meaning the image entered into the model is that of a 32 pixel by 32 pixel with 3 color channels. The output of the first layer is 28x28x6. So that is the input of layer 2 which is also a convolutional layer with an output of 10x10x16. Next a pooling function is used to output a 5x5x16. Then the data is flattened to give an output of 400. After the data has been flattened it then enters a convolutional layer where the output is 120. Then the data enters another convolutional layer where the input of 120. When the layer outputs 84. Then lastly the final layer takes the 84 and outputs 43 the number of classes for the sign data. In other words the model in bulleted form is this:

LeNet

- Layer 1 – Convolutional Layer, Input 32x32x3 Output 28x28x6
    - Activation is Relu
    - Max Pooling Layer Input 28x28x6 Output 14x14x6
- Layer 2 – Convolution Layer, Input 14x14x6 Output 10x10x16
    - Activation is Relu
    - Max Pooling Layer Input 10x10x16 Output 5x5x16
    - Flatten Input 5x5x16 Output = 400
- Layer 3 – Fully Connected Layer Input = 400 Output = 120
    - Activation is Relu
- Layer 4 - Fully Connected Input = 120 Output = 84
    - Activation is Relu

- Layer 5 – Fully Connected Input = 84 Output = 43

CODE:

```
import tensorflow as tf


EPOCHS = 30

BATCH_SIZE = 30

def LeNet(x):

    # Arguments used for tf.truncated_normal, randomly defines variables for the weights and biases for each layer

    mu = 0

    sigma = 0.1


    # SOLUTION: Layer 1: Convolutional. Input = 32x32x3. Output = 28x28x6.

    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 3, 6), mean = mu, stddev = sigma))

    conv1_b = tf.Variable(tf.zeros(6))

    conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv1_b


    # SOLUTION: Activation.

    conv1 = tf.nn.relu(conv1)


    # SOLUTION: Pooling. Input = 28x28x6. Output = 14x14x6.

    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')


    # SOLUTION: Layer 2: Convolutional. Output = 10x10x16.

    conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean = mu, stddev = sigma))

    conv2_b = tf.Variable(tf.zeros(16))

    conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VALID') + conv2_b
```

```
# SOLUTION: Activation.
conv2 = tf.nn.relu(conv2)

# SOLUTION: Pooling. Input = 10x10x16. Output = 5x5x16.
conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

# SOLUTION: Flatten. Input = 5x5x16. Output = 400.
fc0   = flatten(conv2)

# SOLUTION: Layer 3: Fully Connected. Input = 400. Output = 120.
fc1_W = tf.Variable(tf.truncated_normal(shape=(400, 120), mean = mu, stddev = sigma))
fc1_b = tf.Variable(tf.zeros(120))
fc1   = tf.matmul(fc0, fc1_W) + fc1_b

# SOLUTION: Activation.
fc1    = tf.nn.relu(fc1)

# SOLUTION: Layer 4: Fully Connected. Input = 120. Output = 84.
fc2_W  = tf.Variable(tf.truncated_normal(shape=(120, 84), mean = mu, stddev = sigma))
fc2_b  = tf.Variable(tf.zeros(84))
fc2    = tf.matmul(fc1, fc2_W) + fc2_b

# SOLUTION: Activation.
fc2    = tf.nn.relu(fc2)

# SOLUTION: Layer 5: Fully Connected. Input = 84. Output = 43.
fc3_W  = tf.Variable(tf.truncated_normal(shape=(84, 43), mean = mu, stddev = sigma))
fc3_b  = tf.Variable(tf.zeros(43))
logits = tf.matmul(fc2, fc3_W) + fc3_b
```

```
    return logits
```

Then comes the moment of truth does the model do as expected in order to train this model the learning rate has to be entered in this case it was 0.000795 that seemed to get the best results after many tries and testing. This model was optimized by using the Adam Optimizer from the notes. This model also employees one hot encoding to help to determine when to use the learning rate. After the model has been trained it is saved for later use as a lenet file. As the model trained it quickly got to a 91% validation rate and finished training with a 97% validation rate. Then it was tested on the test data and got a test rate of 90%. This is to be expected that test rate would be slightly lower than the training data since this is new material it has never seen. Also some of the images do look very similar. Although the real test of this model is how will it preform when given new images from the internet. The images are displayed and labeled in the notebook.

CODE:

```
import glob

import cv2


my_images = sorted(glob.glob('./mysigns/*.png'))

my_labels = np.array([1, 22, 35, 15, 37, 18])

name_values = np.genfromtxt('signnames.csv', skip_header=1, dtype=[('myint', 'i8'), ('mysring', 'S55')], delimiter=',')


figures = {}

labels = {}

my_signs = []

index = 0

for my_image in my_images:

    img = cv2.cvtColor(cv2.imread(my_image), cv2.COLOR_BGR2RGB)

    my_signs.append(img)

    figures[index] = img

    labels[index] = name_values[my_labels[index]][1].decode('ascii')

    index += 1
```

```
plot_figures(figures, 3, 2, labels)
```

Speed limit (30km/h)

Bumpy road

Ahead only

No vehicles

Go straight or left

General caution

To prepare the sign images for analysis they were normalized before tested on the model. Then the model is asked to predict what each image is. The model predicted the images with a predicted accuracy of 83% very close to the test data results.

CODE:

```
my_signs = np.array(my_signs)

my_signs_normalized = my_signs/127.5-1


with tf.Session() as sess:
    saver.restore(sess, "./lenet")
    my_accuracy = evaluate(my_signs, my_labels)
    print("My Data set Accuracy is = {:.3f}". format(my_accuracy))
```

When using the Softmax function I see what happened with the predictions when looking at what the software predicted it got most of the right with a high confidence but when looking at the blank circle it was not sure this could probably be fixed with more data to teach the program what this blank circle means, I feel that the reason it did not classify this one correctly I think it was focused more on the circle and not the entire sign. The white may have confused the model when it was trying to make its prediction.

CODE:

```
my_single_item_array = []

my_single_item_label_array = []


for i in range(6):
    my_single_item_array.append(my_signs[i])
    my_single_item_label_array.append(my_labels[i])


    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        saver.restore(sess, "./lenet")
        my_accuracy = evaluate(my_single_item_array, my_single_item_label_array)
```

```
    print('Image {}'.format(i+1))
    print("Image Accuracy = {:.3f}".format(my_accuracy))
    print()
```

Over all you can see in the notebook that this model did predict 5 of the 6 signs correctly and with some more data and training I feel it will again be able to predict all 6 correctly.

CODE:

```
k_size = 5
softmax_logits = tf.nn.softmax(logits)
top_k = tf.nn.top_k(softmax_logits, k=k_size)

with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  saver.restore(sess, "./lenet")
  my_softmax_logits = sess.run(softmax_logits, feed_dict={x :my_signs, keep_prob: 1.0})
  my_top_k = sess.run(top_k, feed_dict={x: my_signs, keep_prob: 1.0})

  for i in range(6):
    figures = {}
    labels = {}

    figures[0] = my_signs[i]
    labels[0] = "Original"

    for j in range(k_size):
      labels[j+1] = 'Guess {} : ({:.0f}%)'.format(j+1, 100*my_top_k[0][i][j])
      figures[j+1] = X_valid[np.argwhere(y_valid == my_top_k[1][i][j])[0]].squeeze()
    plot_figures(figures, 1, 6, labels)
```

The prediction based on using the model saved as lenet shows that the model was able to predict 5 of the 6 signs accurately. The program has a very high confidence (100%) on the first 5 signs. Although it did not have as high a confidence (88%) with the last image. When identifying the predictions of the softmax function you can see in the images below that the software shows the 5 best guesses of what it thinks the sign it is testing is. Although looking at this display we see that for the first 5 it shows the first prediction with a prediction of 100%. The last sign though it was not as accurate when it predicted the sign but when we compare it against the label of the actual sign we find that the prediction made an error, although it did only have a prediction of 88% so it wasn't as confidence.

I do find it interesting though different times I have run the same model and get conflicting results a couple of time I would see 100% accuracy and other time it get slightly less.