Traffic_Sign_Classifier

November 30, 2017

1 Self-Driving Car Engineer Nanodegree

1.1 Deep Learning

1.2 Project: Build a Traffic Sign Recognition Classifier

In this notebook, a template is provided for you to implement your functionality in stages, which is required to successfully complete this project. If additional code is required that cannot be included in the notebook, be sure that the Python code is successfully imported and included in your submission if necessary.

> **Note**: Once you have completed all of the code implementations, you need to finalize your work by exporting the iPython Notebook as an HTML document. Before exporting the notebook to html, all of the code cells need to have been run so that reviewers can see the final implementation and output. You can then export the notebook by using the menu above and navigating to ","**File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

In addition to implementing code, there is a writeup to complete. The writeup should be completed in a separate file, which can be either a markdown file or a pdf document. There is a write up template that can be used to guide the writing process. Completing the code template and writeup template will cover all of the rubric points for this project.

The rubric contains "Stand Out Suggestions" for enhancing the project beyond the minimum requirements. The stand out suggestions are optional. If you decide to pursue the "stand out suggestions", you can include the code in this Ipython notebook and also discuss the results in the writeup file.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

1.3 Step 0: Load The Data

```
In [2]: # Load pickled data
        import pickle
```

```
# TODO: Fill this in based on where you saved the training and testing data

training_file = './TestData/train.p'
validation_file= './TestData/valid.p'
testing_file = './TestData/test.p'

with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)

X_train, y_train = train['features'], train['labels']
X_valid, y_valid = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']
```

1.4 Step 1: Dataset Summary & Exploration

The pickled data is a dictionary with 4 key/value pairs:

- `'features'` is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).
- `'labels'` is a 1D array containing the label/class id of the traffic sign. The file `signnames.csv` contains id -> name mappings for each id.
- `'sizes'` is a list containing tuples, (width, height) representing the original width and height the image.
- `'coords'` is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image. **THESE COORDINATES ASSUME THE ORIGINAL IMAGE. THE PICKLED DATA CONTAINS RESIZED VERSIONS (32 by 32) OF THESE IMAGES**

Complete the basic data summary below. Use python, numpy and/or pandas methods to calculate the data summary rather than hard coding the results. For example, the pandas shape method might be useful for calculating some of the summary results.

1.4.1 Provide a Basic Summary of the Data Set Using Python, Numpy and/or Pandas

```
In [3]: ### Replace each question mark with the appropriate value.
        ### Use python, pandas or numpy methods rather than hard coding the results

        # TODO: Number of training examples
        n_train = len(X_train)

        # TODO: Number of validation examples
        n_valid = len(X_valid)
```

```
# TODO: Number of testing examples.
n_test = len(X_test)

# TODO: What's the shape of an traffic sign image?
image_shape = X_train[1].shape

# TODO: How many unique classes/labels there are in the dataset.
n_classes = max(y_train)+1

assert max(y_train)==max(y_valid), 'The number of classes are different between training
assert max(y_train)==max(y_test), 'The number of classes are different between training

print("Number of training examples =", n_train)
print("Number of validation examples =", n_valid)
print("Number of testing examples =", n_test)
print("Image data shape =", image_shape)
print("Number of classes =", n_classes)
```

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

1.4.2 Include an exploratory visualization of the dataset

Visualize the German Traffic Signs Dataset using the pickled file(s). This is open ended, suggestions include: plotting traffic sign images, plotting the count of each sign, etc.

The Matplotlib examples and gallery pages are a great resource for doing visualizations in Python.

NOTE: It's recommended you start with something simple first. If you wish to do more, come back to it after you've completed the rest of the sections. It can be interesting to look at the distribution of classes in the training, validation and test set. Is the distribution the same? Are there more examples of some classes than others?

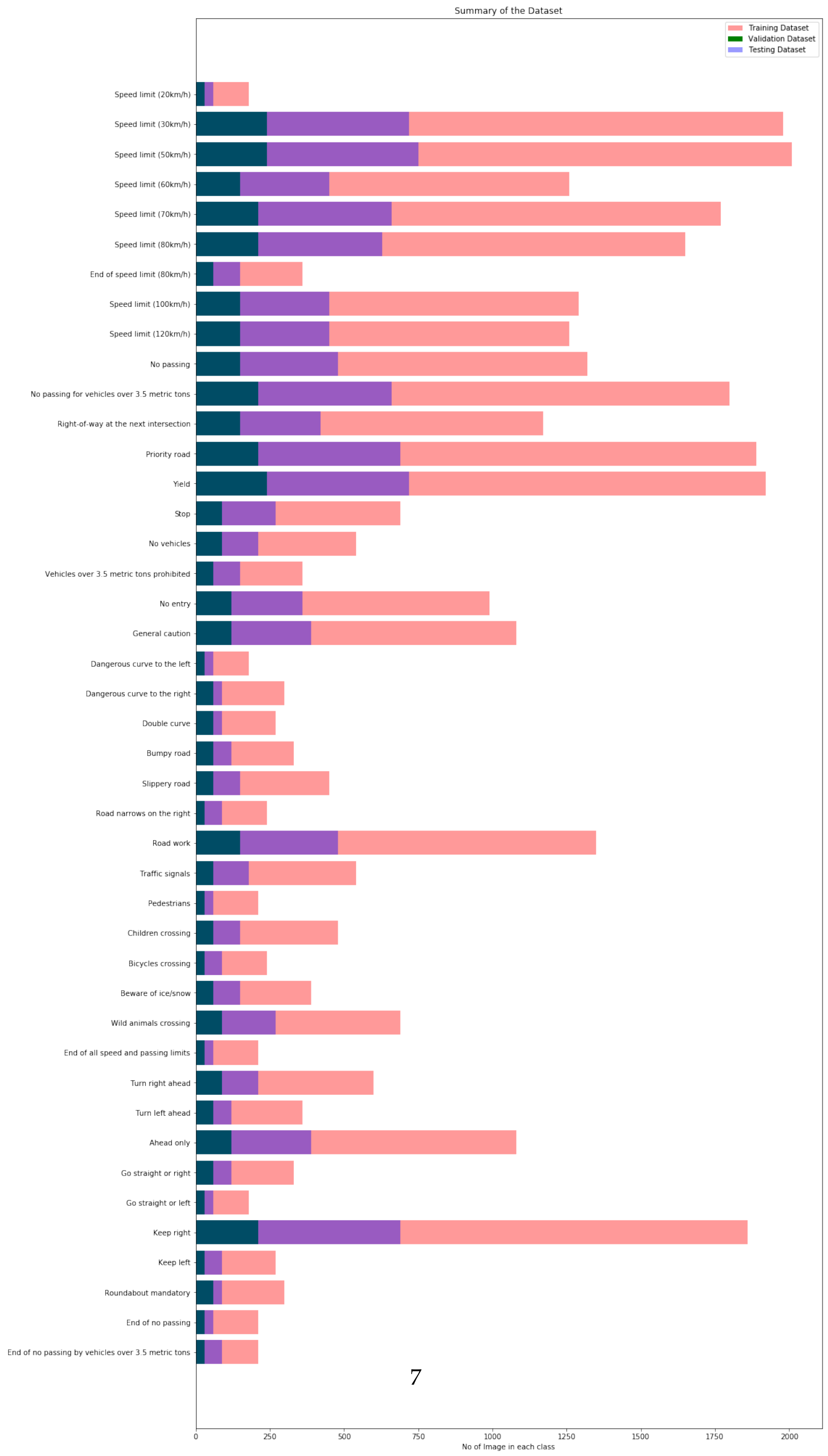
```
In [4]: ### Data exploration visualization code goes here.
        ### Feel free to use as many code cells as needed.
        import random
        import matplotlib.pyplot as plt
        import numpy as np
        from texttable import Texttable

        # Visualizations will be shown in the notebook.
        %matplotlib inline

        sign_names_file = "./signnames.csv"
```

```
f = open('./signnames.csv','r')
lines = f.readlines()
dictSignNames = {}
for line in lines:
    text = line.split(',')
    if not text [0] == "ClassId":
        dictSignNames.update({int(text[0]):text[1].strip()})

def GetSignIndices(y_data,dictSignNames):
    dictSignIndices = {}
    # Defining dictionary to contain sign names and their indices
    for i in dictSignNames:
        dictSignIndices.update({dictSignNames[i]:[]})

    # Storing the indices in its respective sign in the dictionary
    for index in range(len(y_data)):
        classid = y_data[index]
        signname = dictSignNames[classid]
        dictSignIndices[signname].append(index)

    return dictSignIndices


# Summary of data

dictSignIndices_train = GetSignIndices(y_train,dictSignNames)
dictSignIndices_valid = GetSignIndices(y_valid,dictSignNames)
dictSignIndices_test = GetSignIndices(y_test,dictSignNames)

#print('Summary of Training Dataset:')
##print('Sign Name\t No of Image\t Percentage in total data')
#t = Texttable()
#t.add_row(['Sign Name','No of Images','Percentage in total data'])
#
#for sign in dictSignIndices_train:
#
#    no_images = len(dictSignIndices_train[sign])
#    rel_no_images = no_images/n_train*100
#    t.add_row([sign,no_images,rel_no_images])
#
#t.add_row(['Total',n_train,'100'])
#
#print(t.draw())
#
## Summary of validation data
#
#
#print('\n\nSummary of Validation Dataset:')
```

```
##print('Sign Name\t No of Image\t Percentage in total data')
#t1 = Texttable()
#t1.add_row(['Sign Name','No of Images','Percentage in total data'])
#
#for sign in dictSignIndices_valid:
#
#    no_images = len(dictSignIndices_valid[sign])
#    rel_no_images = no_images/n_valid*100
#    t1.add_row([sign,no_images,rel_no_images])
#
#t1.add_row(['Total',n_valid,'100'])
#
#print(t1.draw())
#
## Summary of test data
#dictSignIndices_test = GetSignIndices(y_test,dictSignNames)
#
#print('\n\nSummary of Test Dataset:')
##print('Sign Name\t No of Image\t Percentage in total data')
#t2 = Texttable()
#t2.add_row(['Sign Name','No of Images','Percentage in total data'])
#
#for sign in dictSignIndices_test:
#
#    no_images = len(dictSignIndices_test[sign])
#    rel_no_images = no_images/n_test*100
#    t2.add_row([sign,no_images,rel_no_images])
#
#t2.add_row(['Total',n_test,'100'])
#
#print(t2.draw())
```

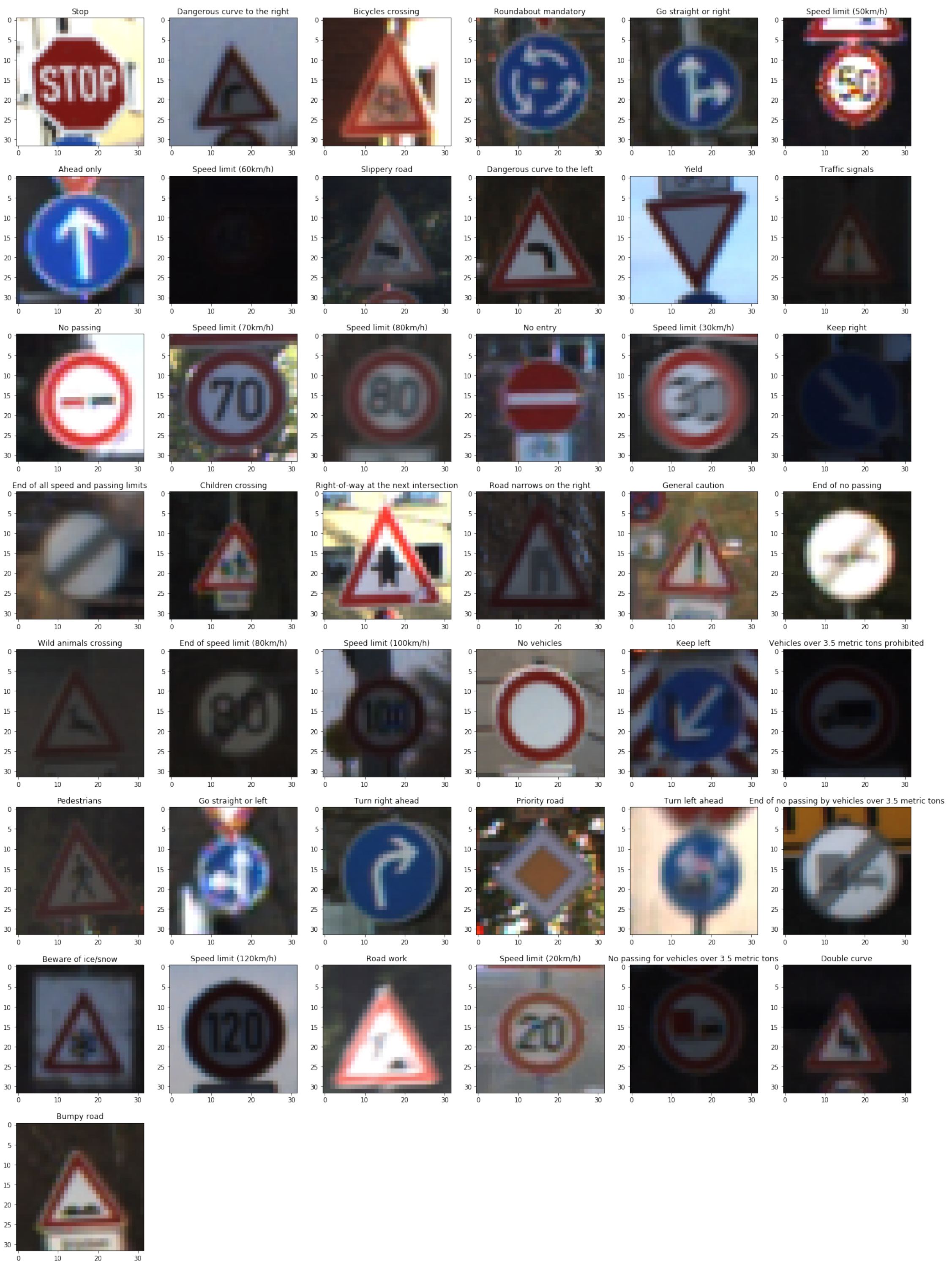
In [5]:
```
#plt.rcdefaults()

fig = plt.figure(figsize=(20,45))
ax = fig.add_axes([0.4,0.4,0.6,0.6])

SignNames =[dictSignNames[f] for f in range(len(dictSignNames))]
y_pos = np.arange(len(SignNames))
no_of_images_valid = [len(dictSignIndices_valid[f]) for f in SignNames]
no_of_images_train = [len(dictSignIndices_train[f]) for f in SignNames]
no_of_images_test = [len(dictSignIndices_test[f]) for f in SignNames]

bar_train = ax.barh(y_pos, no_of_images_train,alpha=0.4,color='r',label='Training Datase
bar_valid = ax.barh(y_pos, no_of_images_valid,color='g',label='Validation Dataset')
bar_test = ax.barh(y_pos, no_of_images_test,color='b',alpha= 0.4,label='Testing Dataset'
ax.set_yticks(y_pos)
ax.set_yticklabels(SignNames)
```

```python
ax.invert_yaxis()  # labels read top-to-bottom
ax.set_xlabel('No of Image in each class')
ax.set_title('Summary of the Dataset')
plt.legend()

plt.show()
```


Summary of the Dataset
Training Dataset
Validation Dataset
Testing Dataset
Speed limit (20km/h)
Speed limit (30km/h)
Speed limit (50km/h)
Speed limit (60km/h)
Speed limit (70km/h)
Speed limit (80km/h)
End of speed limit (80km/h)
Speed limit (100km/h)
Speed limit (120km/h)
No passing
No passing for vehicles over 3.5 metric tons
Right-of-way at the next intersection
Priority road
Yield
Stop
No vehicles
Vehicles over 3.5 metric tons prohibited
No entry
General caution
Dangerous curve to the left
Dangerous curve to the right
Double curve
Bumpy road
Slippery road
Road narrows on the right
Road work
Traffic signals
Pedestrians
Children crossing
Bicycles crossing
Beware of ice/snow
Wild animals crossing
End of all speed and passing limits
Turn right ahead
Turn left ahead
Ahead only
Go straight or right
Go straight or left
Keep right
Keep left
Roundabout mandatory
End of no passing
End of no passing by vehicles over 3.5 metric tons
0
250
500
750
1000
1250
1500
1750
2000
No of Image in each class

```
In [5]: print('Sample images of the Traffic Signs:')
        j = 1
        plt.figure(figsize=(25,40))
        for sign in dictSignIndices_train:
            plt.subplot(9,6,j)
            index = random.choice(dictSignIndices_train[sign])
            image = X_train[index].squeeze()
            plt.imshow(image)
            plt.title(sign)
            j +=1
```

```
Sample images of the Traffic Signs:
```


Stop
Dangerous curve to the right
Bicycles crossing
Roundabout mandatory
Go straight or right
Speed limit (50km/h)
Ahead only
Speed limit (60km/h)
Slippery road
Dangerous curve to the left
Yield
Traffic signals
No passing
Speed limit (70km/h)
Speed limit (80km/h)
No entry
Speed limit (30km/h)
Keep right
End of all speed and passing limits
Children crossing
Right-of-way at the next intersection
Road narrows on the right
General caution
End of no passing
Wild animals crossing
End of speed limit (80km/h)
Speed limit (100km/h)
No vehicles
Keep left
Vehicles over 3.5 metric tons prohibited
Pedestrians
Go straight or left
Turn right ahead
Priority road
Turn left ahead
End of no passing by vehicles over 3.5 metric tons
Beware of ice/snow
Speed limit (120km/h)
Road work
Speed limit (20km/h)
No passing for vehicles over 3.5 metric tons
Double curve
Bumpy road

1.5 Step 2: Design and Test a Model Architecture

Design and implement a deep learning model that learns to recognize traffic signs. Train and test your model on the German Traffic Sign Dataset.

The LeNet-5 implementation shown in the classroom at the end of the CNN lesson is a solid starting point. You'll have to change the number of classes and possibly the preprocessing, but aside from that it's plug and play!

With the LeNet-5 solution from the lecture, you should expect a validation set accuracy of about 0.89. To meet specifications, the validation set accuracy will need to be at least 0.93. It is possible to get an even higher accuracy, but 0.93 is the minimum for a successful project submission.

There are various aspects to consider when thinking about this problem:

- Neural network architecture (is the network over or underfitting?)
- Play around preprocessing techniques (normalization, rgb to grayscale, etc)
- Number of examples per label (some have more than others).
- Generate fake data.

Here is an example of a published baseline model on this problem. It's not required to be familiar with the approach used in the paper but, it's good practice to try to read papers like these.

1.5.1 Pre-process the Data Set (normalization, grayscale, etc.)

Minimally, the image data should be normalized so that the data has mean zero and equal variance. For image data, `(pixel - 128)/ 128` is a quick way to approximately normalize the data and can be used in this project.

Other pre-processing steps are optional. You can try different techniques to see if it improves performance.

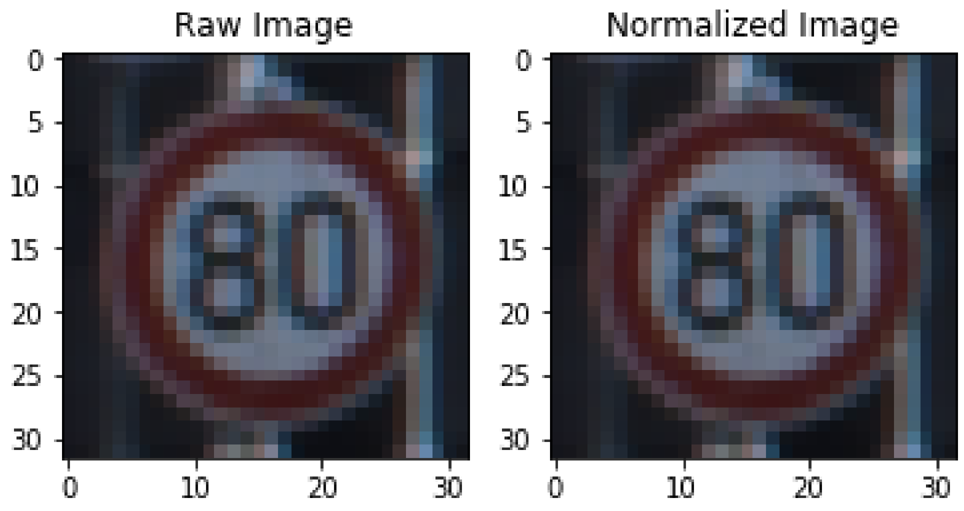
Use the code cell (or multiple code cells, if necessary) to implement the first step of your project.

```
In [6]: ### Preprocess the data here. It is required to normalize the data. Other preprocessing
        ### converting to grayscale, etc.
        ### Feel free to use as many code cells as needed.

        index = random.randint(0, len(X_train))
        image_raw = X_train[index].squeeze()

        def normalize_min_max(x_data,x_min = 0,x_max = 255,a = 0,b =1):
            return a + (x_data-x_min)*(b-a)/(x_max-x_min)

        X_train = normalize_min_max(X_train)
        X_valid = normalize_min_max(X_valid)
        X_test  = normalize_min_max(X_test)

        print('Data is normalized')
```

```
Data is normalized
```

```
In [7]: # image after normalization
        image_normalized = X_train[index].squeeze()

        raw_image = plt.subplot(1,2,1)
        raw_image.set_title('Raw Image')
        raw_image.imshow(image_raw, cmap="gray")

        normalized_image = plt.subplot(1,2,2)
        normalized_image.set_title('Normalized Image')
        normalized_image.imshow(image_normalized, cmap="gray")
        print(dictSignNames.get(y_train[index]))
```

```
Speed limit (80km/h)
```



1.5.2 Model Architecture

```
In [8]: ### Define your architecture here.
        ### Feel free to use as many code cells as needed.
        # Shuffling the data
        from sklearn.utils import shuffle
        import tensorflow as tf
        from tensorflow.contrib.layers import flatten

        X_train, y_train = shuffle(X_train, y_train)

        EPOCHS = 40
        BATCH_SIZE = 100
        keep_prob = 0.4
```

```
#LeNet function

def LeNet(x):
    # Arguments used for tf.truncated_normal, randomly defines variables for the weights
    mu = 0
    sigma = 0.1

    weights = {'wc1':tf.Variable(tf.truncated_normal([5,5,3,6],mu,sigma,dtype=tf.float32
               'wc2':tf.Variable(tf.truncated_normal([5,5,6,16],mu,sigma,dtype=tf.float3
               'wd1':tf.Variable(tf.truncated_normal([400,120],mu,sigma,dtype=tf.float32
               'wd2':tf.Variable(tf.truncated_normal([120,84],mu,sigma,dtype=tf.float32)
               'wd3':tf.Variable(tf.truncated_normal([84,n_classes],mu,sigma,dtype=tf.fl
              }

    biases = {'bc1': tf.Variable(tf.zeros([6],dtype=tf.float32)),\
              'bc2': tf.Variable(tf.zeros([16],dtype=tf.float32)),\
              'bd1': tf.Variable(tf.zeros([120],dtype=tf.float32)),\
              'bd2': tf.Variable(tf.zeros([84],dtype=tf.float32)),\
              'bd3': tf.Variable(tf.zeros([n_classes],dtype=tf.float32)),\
             }

    # TODO: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.
    conv1 = tf.nn.conv2d(x,weights['wc1'],strides=[1,1,1,1],padding='VALID')
    conv1 = tf.nn.bias_add(conv1,biases['bc1'])
    print("Tensor name of convolution 1",conv1)

    # TODO: Activation.
    conv1 = tf.nn.relu(conv1)
    print("Tensor name of Relu 1",conv1)

    #conv1_dims = [conv1.shape.dims[1].value,conv1.shape.dims[2].value,conv1.shape.dims[
    #assert conv1_dims == [28,28,6],'The ouput of first convolution layer is not right'

    # TODO: Pooling. Input = 28x28x6. Output = 14x14x6.
    pool1 = tf.nn.max_pool(conv1,ksize=[1,2,2,1],strides=[1,2,2,1],padding='VALID')
    print("Tensor name of max pooling 1:",pool1)

    #pool1_dims = [pool1.dims[1].value,pool1.dims[2].value,pool1.dims[3].value]
    #assert pool1_dims == [14,14,6],'The ouput of first pooling layer is not right'

    # TODO: Layer 2: Convolutional. Output = 10x10x16.
    conv2 = tf.nn.conv2d(pool1,weights['wc2'],strides=[1,1,1,1],padding='VALID')
    conv2 = tf.nn.bias_add(conv2,biases['bc2'])
    print("Tensor name of convolution 2",conv2)

    # TODO: Activation.
    conv2 = tf.nn.relu(conv2)
    print("Tensor name of relu 2",conv2)
    #conv2_dims = [conv2.shape.dims[1].value,conv2.shape.dims[2].value,conv2.shape.dims[
    #assert conv1_dims == [10,10,16],'The ouput of first convolution layer is not right'

    # TODO: Pooling. Input = 10x10x16. Output = 5x5x16.
    pool2 = tf.nn.max_pool(conv2,ksize=[1,2,2,1],strides=[1,2,2,1],padding = 'VALID')
    print("Tensor name of max pooling 2",pool2)

    #pool1_dims = [pool2.dims[1].value,pool2.dims[2].value,pool2.dims[3].value]
    #assert pool1_dims == [5,5,16],'The ouput of first pooling layer is not right'

    # TODO: Flatten. Input = 5x5x16. Output = 400.
    flat1 = flatten(pool2)

    # TODO: Layer 3: Fully Connected. Input = 400. Output = 120.
    fc1 = tf.add(tf.matmul(flat1,weights['wd1']),biases['bd1'])

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

    # TODO: Layer 4: Fully Connected. Input = 120. Output = 84.
    fc2 = tf.add(tf.matmul(fc1,weights['wd2']),biases['bd2'])

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

    # TODO: Layer 5: Fully Connected. Input = 84. Output = n_classes.
    logits = tf.add(tf.matmul(fc2,weights['wd3']),biases['bd3'])
    #logits = tf.nn.dropout(logits,keep_prob)
    return logits
```

1.5.3 Train, Validate and Test the Model

A validation set can be used to assess how well the model is performing. A low accuracy on the training and validation sets imply underfitting. A high accuracy on the training set but low accuracy on the validation set implies overfitting.

```
In [9]: ### Train your model here.
        ### Calculate and report the accuracy on the training and validation set.
        ### Once a final model architecture is selected,
        ### the accuracy on the test set should be calculated and reported as well.
        ### Feel free to use as many code cells as needed.
        # Normalizing the data with zero mean

        # Placeholder for inputs and labels

        x = tf.placeholder(tf.float32, (None, 32, 32, 3))
        y = tf.placeholder(tf.int32, (None))
```

```
one_hot_y = tf.one_hot(y, n_classes)

# Forward and Backprobagation

rate = 0.0005

logits = LeNet(x)
logits_dropout = tf.nn.dropout(logits,keep_prob)
print("Tensor name of logits",logits)
cross_entropy_dropout = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot_y, logits
loss_operation_dropout = tf.reduce_mean(cross_entropy_dropout)
optimizer = tf.train.AdamOptimizer(learning_rate = rate)
training_operation = optimizer.minimize(loss_operation_dropout)



#Evaluating the Model

cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot_y, logits=logits)
loss_operation = tf.reduce_mean(cross_entropy)
prediction_softmax = tf.nn.softmax(logits)
prediction_argmax = tf.argmax(prediction_softmax,1)
top_5_softmax = tf.nn.top_k(prediction_softmax, k=5)
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
    total_loss = 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_
        accuracy = sess.run(accuracy_operation, feed_dict={x: batch_x, y: batch_y})
        loss = sess.run(loss_operation, feed_dict={x: batch_x, y: batch_y})
        total_loss += (loss*len(batch_x))
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples, total_loss/ num_examples

# Training the model
validation_accuracy = []
validation_loss = []
training_accuracy = []
training_loss = []

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})


            valid_acc,valid_los = evaluate(X_valid, y_valid)
            train_acc,training_los = evaluate(X_train,y_train)

            validation_accuracy.append(valid_acc)
            validation_loss.append(valid_los)
            training_accuracy.append(train_acc)
            training_loss.append(training_los)
            print("EPOCH {} ...".format(i+1))
            print("Validation Accuracy = {:.3f}".format(valid_acc))
            print("Training Accuracy = {:.3f}".format(train_acc))
            print()

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

```
Tensor name of convolution 1 Tensor("BiasAdd:0", shape=(?, 28, 28, 6), dtype=float32)
Tensor name of Relu 1 Tensor("Relu:0", shape=(?, 28, 28, 6), dtype=float32)
Tensor name of max pooling 1: Tensor("MaxPool:0", shape=(?, 14, 14, 6), dtype=float32)
Tensor name of convolution 2 Tensor("BiasAdd_1:0", shape=(?, 10, 10, 16), dtype=float32)
Tensor name of relu 2 Tensor("Relu_1:0", shape=(?, 10, 10, 16), dtype=float32)
Tensor name of max pooling 2 Tensor("MaxPool_1:0", shape=(?, 5, 5, 16), dtype=float32)
Tensor name of logits Tensor("Add_2:0", shape=(?, 43), dtype=float32)
Training...

EPOCH 1 ...
Validation Accuracy = 0.542
Training Accuracy = 0.618

EPOCH 2 ...
Validation Accuracy = 0.737
Training Accuracy = 0.794

EPOCH 3 ...
Validation Accuracy = 0.817
Training Accuracy = 0.888
```

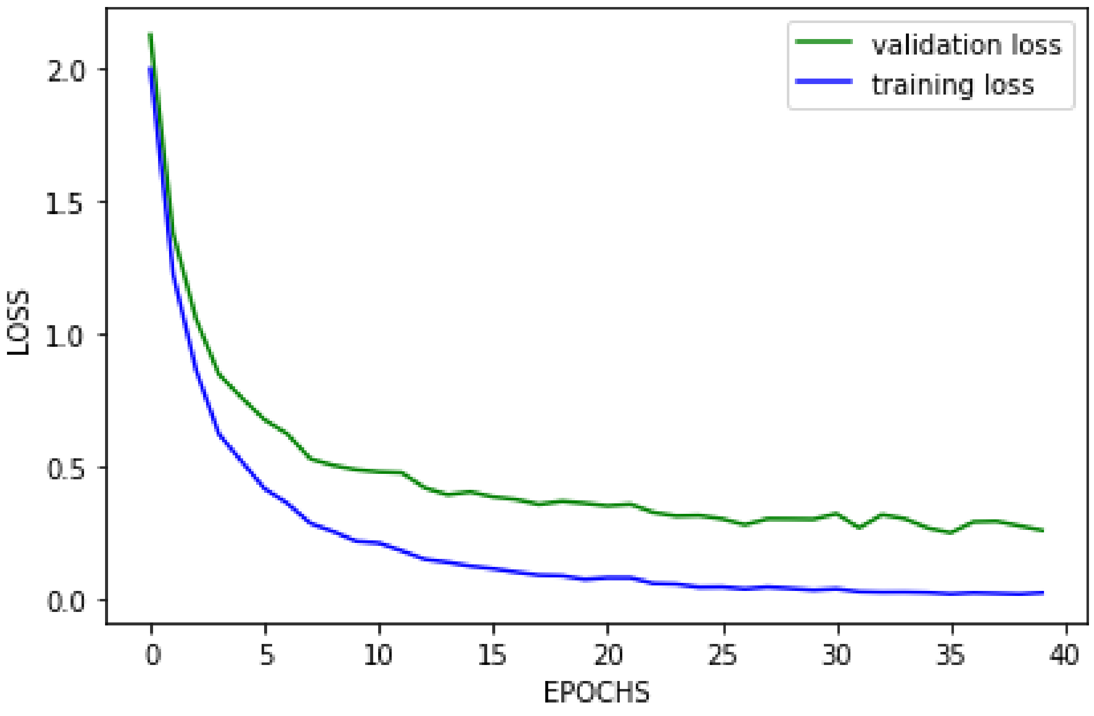
```
EPOCH 4 ...
Validation Accuracy = 0.844
Training Accuracy = 0.922

EPOCH 5 ...
Validation Accuracy = 0.865
Training Accuracy = 0.942

EPOCH 6 ...
Validation Accuracy = 0.863
Training Accuracy = 0.950

EPOCH 7 ...
Validation Accuracy = 0.873
Training Accuracy = 0.964

EPOCH 8 ...
Validation Accuracy = 0.892
Training Accuracy = 0.968

EPOCH 9 ...
Validation Accuracy = 0.889
Training Accuracy = 0.972

EPOCH 10 ...
Validation Accuracy = 0.897
Training Accuracy = 0.976

EPOCH 11 ...
Validation Accuracy = 0.889
Training Accuracy = 0.973

EPOCH 12 ...
Validation Accuracy = 0.880
Training Accuracy = 0.981

EPOCH 13 ...
Validation Accuracy = 0.899
Training Accuracy = 0.983

EPOCH 14 ...
Validation Accuracy = 0.912
Training Accuracy = 0.990

EPOCH 15 ...
Validation Accuracy = 0.897
Training Accuracy = 0.988
```

```
EPOCH 16 ...
Validation Accuracy = 0.910
Training Accuracy = 0.990

EPOCH 17 ...
Validation Accuracy = 0.911
Training Accuracy = 0.990

EPOCH 18 ...
Validation Accuracy = 0.917
Training Accuracy = 0.993

EPOCH 19 ...
Validation Accuracy = 0.910
Training Accuracy = 0.992

EPOCH 20 ...
Validation Accuracy = 0.911
Training Accuracy = 0.993

EPOCH 21 ...
Validation Accuracy = 0.912
Training Accuracy = 0.990

EPOCH 22 ...
Validation Accuracy = 0.913
Training Accuracy = 0.991

EPOCH 23 ...
Validation Accuracy = 0.911
Training Accuracy = 0.995

EPOCH 24 ...
Validation Accuracy = 0.925
Training Accuracy = 0.995

EPOCH 25 ...
Validation Accuracy = 0.927
Training Accuracy = 0.997

EPOCH 26 ...
Validation Accuracy = 0.929
Training Accuracy = 0.997

EPOCH 27 ...
Validation Accuracy = 0.929
Training Accuracy = 0.998
```

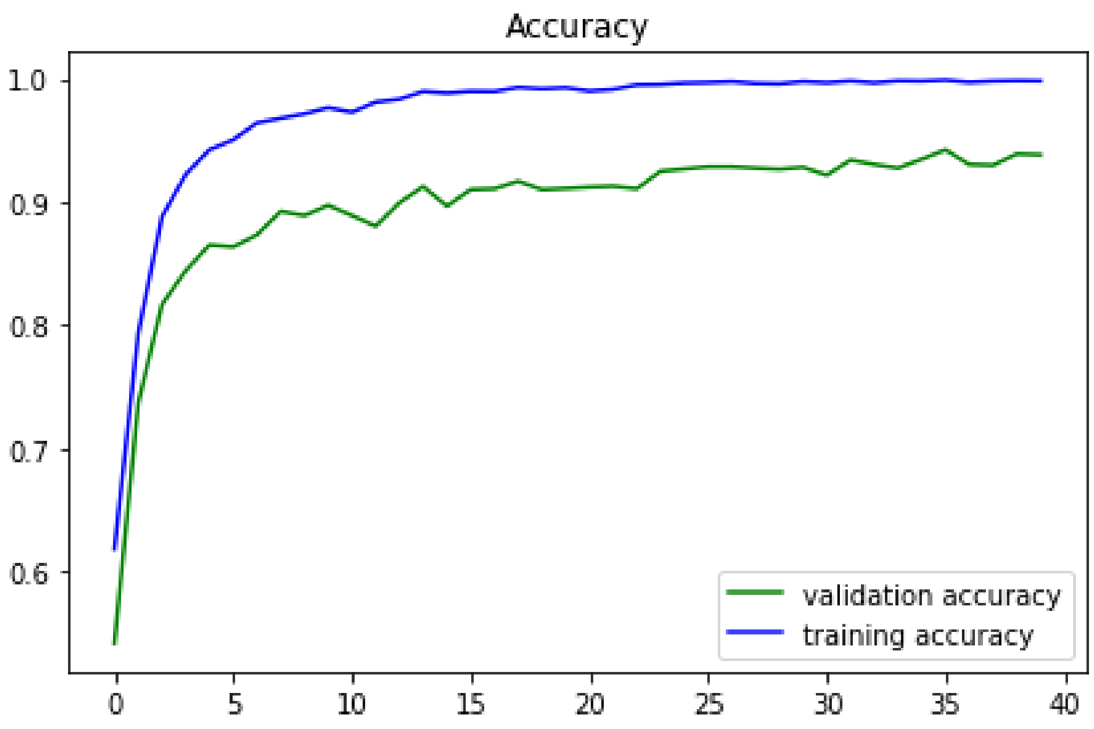
```
EPOCH 28 ...
Validation Accuracy = 0.927
Training Accuracy = 0.996

EPOCH 29 ...
Validation Accuracy = 0.927
Training Accuracy = 0.996

EPOCH 30 ...
Validation Accuracy = 0.928
Training Accuracy = 0.998

EPOCH 31 ...
Validation Accuracy = 0.922
Training Accuracy = 0.997

EPOCH 32 ...
Validation Accuracy = 0.934
Training Accuracy = 0.998

EPOCH 33 ...
Validation Accuracy = 0.930
Training Accuracy = 0.997

EPOCH 34 ...
Validation Accuracy = 0.927
Training Accuracy = 0.998

EPOCH 35 ...
Validation Accuracy = 0.935
Training Accuracy = 0.998

EPOCH 36 ...
Validation Accuracy = 0.942
Training Accuracy = 0.999

EPOCH 37 ...
Validation Accuracy = 0.930
Training Accuracy = 0.997

EPOCH 38 ...
Validation Accuracy = 0.930
Training Accuracy = 0.998

EPOCH 39 ...
Validation Accuracy = 0.939
Training Accuracy = 0.998
```

```
EPOCH 40 ...
Validation Accuracy = 0.938
Training Accuracy = 0.998

Model saved
```

```
In [10]: loss_plot = plt
         loss_plot.xlabel('EPOCHS')
         loss_plot.ylabel('LOSS')
         loss_plot.plot(range(EPOCHS),validation_loss, 'g',label = 'validation loss')
         loss_plot.plot(range(EPOCHS), training_loss, 'b',label = 'training loss')
         loss_plot.legend(loc=1)
         plt.tight_layout()
         plt.show()
```



```
In [11]: # Accuracy Plot
         accuracy_plot = plt
         accuracy_plot.title('Accuracy')
         accuracy_plot.plot(range(EPOCHS),validation_accuracy,'g',label='validation accuracy')
         accuracy_plot.plot(range(EPOCHS),training_accuracy,'b',label='training accuracy')
         accuracy_plot.legend(loc = 4)
         plt.tight_layout()
         plt.show()
```



1.6 Testing the model on Test Dataset

```
In [12]: with tf.Session() as sess:
             saver.restore(sess, tf.train.latest_checkpoint('.'))
             test_acc,test_loss = evaluate(X_test,y_test)
             print("Test Accuracy = {:.3f}".format(test_acc))
```

```
INFO:tensorflow:Restoring parameters from .\TrafficSignClassifier
Test Accuracy = 0.915
```

1.7 Step 3: Test a Model on New Images

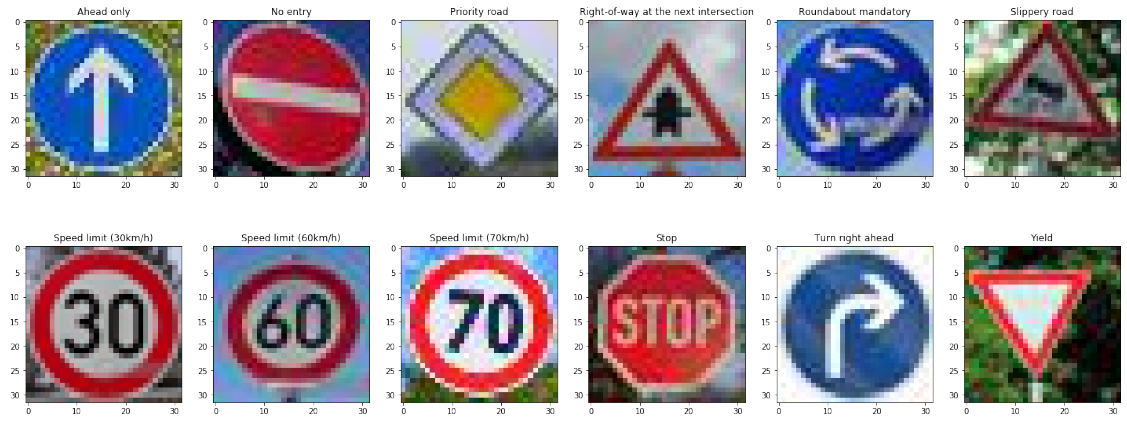
To give yourself more insight into how your model is working, download at least five pictures of German traffic signs from the web and use your model to predict the traffic sign type.

You may find `signnames.csv` useful as it contains mappings from the class id (integer) to the actual sign name.

1.7.1 Load and Output the Images

In [22]:
```
### Load the images and plot them here.
### Feel free to use as many code cells as needed.
from os import listdir
from os.path import isfile, join
import matplotlib.image as mpimg

def CreateDataset(FolderPath,dictSign):
    dictClassId = {}

    for classid in dictSign:
        dictClassId.update({dictSign[classid]:classid})

    onlyfiles = [f for f in listdir(FolderPath) if isfile(join(FolderPath, f))]
    X_data = []
    y_data = []

    for filename in onlyfiles:
        filepath = FolderPath+filename
        img = mpimg.imread(filepath)
        X_data.append(img)

        if filename.find('kmh') !=-1:
            filename = filename.replace('kmh','km/h')
            filename = filename.split('.')[0]
        else:
            filename = filename.split('.')[0]

        ClassId = dictClassId[filename]
        y_data.append(ClassId)

    assert len(X_data)==len(y_data), 'The x and y data does not match '

    return np.asarray(X_data),np.asarray(y_data)

X_internet,y_internet = CreateDataset('./TestImagesInternet/',dictSignNames)

# normalize the data using min max normalization
X_internet = normalize_min_max(X_internet)

j = 1
plt.figure(figsize=(25,15))
for img in X_internet:
    plt.subplot(3,6,j)
    plt.imshow(img)
    sign = dictSignNames[y_internet[j-1]]
    plt.title(sign)
```

```
        j +=1
```

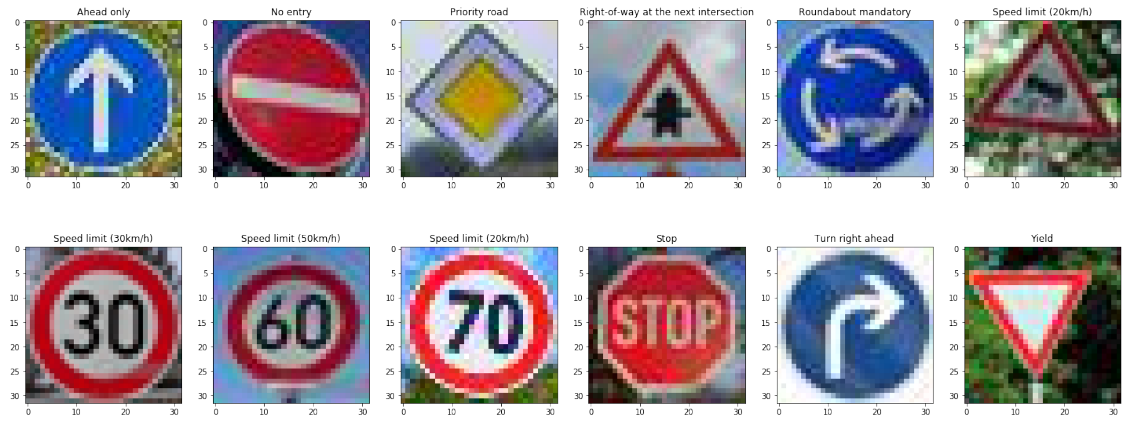


1.7.2 Predict the Sign Type for Each Image

```
In [14]: ### Run the predictions here and use the model to output the prediction for each image.
         ### Make sure to pre-process the images with the same pre-processing pipeline used earl
         ### Feel free to use as many code cells as needed.

         with tf.Session() as sess:
             saver.restore(sess, tf.train.latest_checkpoint('.'))
             prediction_internet = sess.run(prediction_argmax,feed_dict={x:X_internet})
             values_argmax,indices_argmax = sess.run(top_5_softmax,feed_dict={x:X_internet})
             test_acc,test_loss = evaluate(X_internet,y_internet)
             print("Test Accuracy = {:.3f}".format(test_acc))

         values_argmax = values_argmax.tolist()
         indices_argmax = indices_argmax.tolist()
```

```
INFO:tensorflow:Restoring parameters from .\TrafficSignClassifier
Test Accuracy = 0.750
```

```
In [15]: #prediction_internet = np.asarray(prediction_internet)
         #classification = np.argmax(prediction_internet,axis = 1)

         j = 1
         plt.figure(figsize=(25,15))
         for img in X_internet:
             plt.subplot(3,6,j)
             plt.imshow(img)
             sign = dictSignNames[prediction_internet[j-1]]
             plt.title(sign)
             j +=1
```



1.7.3 Analyze Performance

```
In [35]: ### Calculate the accuracy for these 5 new images.
         ### For example, if the model predicted 1 out of 5 signs correctly, it's 20% accurate o
         t3 = Texttable()
         t3.add_row(['Image No','Label','Highest','2nd highest','3rd highest','4th highest','5th

         for i in range(len(y_internet)):
             t3.add_row([i+1]+[dictSignNames[y_internet[i]],dictSignNames[indices_argmax[i][0]],
                          dictSignNames[indices_argmax[i][2]],dictSignNames[indices_argmax[i][3]]
                          dictSignNames[indices_argmax[i][4]]])
             t3.add_row(['']+['']+values_argmax[i])

         t3.set_precision(5)

         print('Probablity of the prediction')
         print(t3.draw())
```

```
Probablity of the prediction
+----------+-----------+-----------+----------+----------+----------+----------+
| Image No | Label     | Highest   | 2nd      | 3rd      | 4th      | 5th      |
|          |           |           | highest  | highest  | highest  | highest  |
+----------+-----------+-----------+----------+----------+----------+----------+
| 1        | Ahead     | Ahead     | Roundabo | Turn     | Go       | Turn     |
|          | only      | only      | ut manda | right    | straight | left     |
|          |           |           | tory     | ahead    | or right | ahead    |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 1         | 0.000    | 0.000    | 0.000    | 0.000    |
+----------+-----------+-----------+----------+----------+----------+----------+
| 2        | No entry  | No entry  | Yield    | Bicycles | No       | Slippery |
|          |           |           |          | crossing | passing  | road     |
|          |           |           |          |          | for      |          |
|          |           |           |          |          | vehicles |          |
|          |           |           |          |          | over 3.5 |          |
|          |           |           |          |          | metric   |          |
|          |           |           |          |          | tons     |          |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 0.589     | 0.250    | 0.150    | 0.008    | 0.001    |
+----------+-----------+-----------+----------+----------+----------+----------+
| 3        | Priority  | Priority  | End of   | Traffic  | End of   | No entry |
|          | road      | road      | all      | signals  | no       |          |
|          |           |           | speed    |          | passing  |          |
|          |           |           | and      |          |          |          |
|          |           |           | passing  |          |          |          |
|          |           |           | limits   |          |          |          |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 1         | 0.000    | 0.000    | 0.000    | 0.000    |
+----------+-----------+-----------+----------+----------+----------+----------+
| 4        | Right-of- | Right-of- | Bicycles | Speed    | Beware   | Speed    |
|          | way at    | way at    | crossing | limit    | of       | limit    |
|          | the next  | the next  |          | (80km/h) | ice/snow | (30km/h) |
|          | intersect | intersect |          |          |          |          |
|          | ion       | ion       |          |          |          |          |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 1.000     | 0.000    | 0.000    | 0.000    | 0.000    |
+----------+-----------+-----------+----------+----------+----------+----------+
| 5        | Roundabou | Roundabou | General  | Right-   | Children | Keep     |
|          | t         | t         | caution  | of-way   | crossing | right    |
|          | mandatory | mandatory |          | at the   |          |          |
|          |           |           |          | next int |          |          |
|          |           |           |          | ersectio |          |          |
|          |           |           |          | n        |          |          |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 0.999     | 0.001    | 0.000    | 0.000    | 0.000    |
+----------+-----------+-----------+----------+----------+----------+----------+
| 6        | Slippery  | Speed     | No entry | Dangerou | End of   | Speed    |
|          | road      | limit     |          | s curve  | speed    | limit    |
|          |           | (20km/h)  |          | to the   | limit    | (30km/h) |
|          |           |           |          | right    | (80km/h) |          |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 0.958     | 0.034    | 0.004    | 0.002    | 0.000    |
+----------+-----------+-----------+----------+----------+----------+----------+
| 7        | Speed     | Speed     | Speed    | Go       | Keep     | End of   |
|          | limit     | limit     | limit    | straight | right    | all      |
|          | (30km/h)  | (30km/h)  | (50km/h) | or right |          | speed    |
|          |           |           |          |          |          | and      |
|          |           |           |          |          |          | passing  |
|          |           |           |          |          |          | limits   |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 0.937     | 0.062    | 0.000    | 0.000    | 0.000    |
+----------+-----------+-----------+----------+----------+----------+----------+
| 8        | Speed     | Speed     | Speed    | Speed    | Speed    | Go       |
|          | limit     | limit     | limit    | limit    | limit    | straight |
|          | (60km/h)  | (50km/h)  | (60km/h) | (80km/h) | (30km/h) | or right |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 0.718     | 0.282    | 0.000    | 0.000    | 0.000    |
+----------+-----------+-----------+----------+----------+----------+----------+
| 9        | Speed     | Speed     | Bicycles | Ahead    | Speed    | Go       |
|          | limit     | limit     | crossing | only     | limit    | straight |
|          | (70km/h)  | (20km/h)  |          |          | (30km/h) | or right |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 0.589     | 0.373    | 0.033    | 0.005    | 0.000    |
+----------+-----------+-----------+----------+----------+----------+----------+
| 10       | Stop      | Stop      | Speed    | Speed    | Priority | Speed    |
|          |           |           | limit    | limit    | road     | limit    |
|          |           |           | (30km/h) | (80km/h) |          | (50km/h) |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 0.985     | 0.014    | 0.001    | 0.000    | 0.000    |
+----------+-----------+-----------+----------+----------+----------+----------+
| 11       | Turn      | Turn      | Right-   | Beware   | Road     | Double   |
|          | right     | right     | of-way   | of       | narrows  | curve    |
|          | ahead     | ahead     | at the   | ice/snow | on the   |          |
|          |           |           | next int |          | right    |          |
|          |           |           | ersectio |          |          |          |
|          |           |           | n        |          |          |          |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 0.623     | 0.377    | 0.000    | 0.000    | 0.000    |
+----------+-----------+-----------+----------+----------+----------+----------+
| 12       | Yield     | Yield     | Speed    | No       | No       | Speed    |
|          |           |           | limit    | vehicles | passing  | limit    |
|          |           |           | (30km/h) |          |          | (60km/h) |
+----------+-----------+-----------+----------+----------+----------+----------+
|          |           | 1.000     | 0.000    | 0.000    | 0.000    | 0.000    |
+----------+-----------+-----------+----------+----------+----------+----------+
```

1.7.4 Output Top 5 Softmax Probabilities For Each Image Found on the Web

For each of the new images, print out the model's softmax probabilities to show the **certainty** of the model's predictions (limit the output to the top 5 probabilities for each image). `tf.nn.top_k` could prove helpful here.

The example below demonstrates how tf.nn.top_k can be used to find the top k predictions for each image.

`tf.nn.top_k` will return the values and indices (class ids) of the top k predictions. So if k=3, for each sign, it'll return the 3 largest probabilities (out of a possible 43) and the correspoding class ids.

Take this numpy array as an example. The values in the array represent predictions. The array contains softmax probabilities for five candidate images with six possible classes. `tf.nn.top_k` is used to choose the three classes with the highest probability:

```
# (5, 6) array
a = np.array([[ 0.24879643,  0.07032244,  0.12641572,  0.34763842,  0.07893497,
         0.12789202],
       [ 0.28086119,  0.27569815,  0.08594638,  0.0178669 ,  0.18063401,
         0.15899337],
       [ 0.26076848,  0.23664738,  0.08020603,  0.07001922,  0.1134371 ,
         0.23892179],
       [ 0.11943333,  0.29198961,  0.02605103,  0.26234032,  0.1351348 ,
         0.16505091],
       [ 0.09561176,  0.34396535,  0.0643941 ,  0.16240774,  0.24206137,
         0.09155967]])
```

Running it through `sess.run(tf.nn.top_k(tf.constant(a), k=3))` produces:

```
TopKV2(values=array([[ 0.34763842,  0.24879643,  0.12789202],
       [ 0.28086119,  0.27569815,  0.18063401],
       [ 0.26076848,  0.23892179,  0.23664738],
       [ 0.29198961,  0.26234032,  0.16505091],
       [ 0.34396535,  0.24206137,  0.16240774]]), indices=array([[3, 0, 5],
       [0, 1, 4],
       [0, 5, 1],
       [1, 3, 5],
       [1, 4, 3]], dtype=int32))
```

Looking just at the first row we get `[ 0.34763842,  0.24879643,  0.12789202]`, you can confirm these are the 3 largest probabilities in `a`. You'll also notice `[3, 0, 5]` are the corresponding indices.

```
In [17]: ### Print out the top five softmax probabilities for the predictions on the German traf
         ### Feel free to use as many code cells as needed.
```

1.7.5 Project Writeup

Once you have completed the code implementation, document your results in a project writeup using this template as a guide. The writeup can be in a markdown or pdf file.

> **Note**: Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to ","**File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

1.8 Step 4 (Optional): Visualize the Neural Network's State with Test Images

This Section is not required to complete but acts as an additional excersise for understaning the output of a neural network's weights. While neural networks can be a great learning device they are often referred to as a black box. We can understand what the weights of a neural network look like better by plotting their feature maps. After successfully training your neural network you can see what it's feature maps look like by plotting the output of the network's weight layers in response to a test stimuli image. From these plotted feature maps, it's possible to see what characteristics of an image the network finds interesting. For a sign, maybe the inner network feature maps react with high activation to the sign's boundary outline or to the contrast in the sign's painted symbol.

Provided for you below is the function code that allows you to get the visualization output of any tensorflow weight layer you want. The inputs to the function should be a stimuli image, one used during training or a new one you provided, and then the tensorflow variable name that represents the layer's state during the training process, for instance if you wanted to see what the LeNet lab's feature maps looked like for it's second convolutional layer you could enter conv2 as the tf_activation variable.

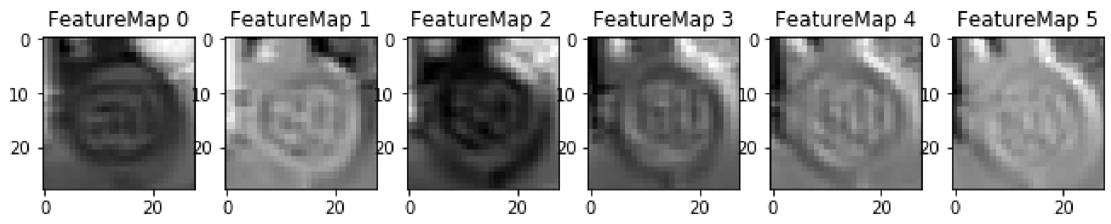
For an example of what feature map outputs look like, check out NVIDIA's results in their paper End-to-End Deep Learning for Self-Driving Cars in the section Visualization of internal CNN State. NVIDIA was able to show that their network's inner weights had high activations to road boundary lines by comparing feature maps from an image with a clear path to one without. Try experimenting with a similar test to show that your trained network's weights are looking for interesting features, whether it's looking at differences in feature maps from images with or without a sign, or even what feature maps look like in a trained network vs a completely untrained one on the same sign image.

Your output should look something like this (above)

```
In [18]: ### Visualize your network's feature maps here.
         ### Feel free to use as many code cells as needed.

         # image_input: the test image being fed into the network to produce the feature maps
         # tf_activation: should be a tf variable name used during your training procedure that
         # activation_min/max: can be used to view the activation contrast in more detail, by de
         # plt_num: used to plot out multiple different weight feature map sets on the same bloc

         def outputFeatureMap(image_input, tf_activation, activation_min=-1, activation_max=-1 ,
             # Here make sure to preprocess your image_input in a way your network expects
             # with size, normalization, ect if needed
             # image_input =
             # Note: x should be the same name as your network's tensorflow data placeholder var
             # If you get an error tf_activation is not defined it may be having trouble accessi
             activation = tf_activation.eval(session=sess,feed_dict={x : image_input})
             featuremaps = activation.shape[3]
             plt.figure(plt_num, figsize=(15,15))
             for featuremap in range(featuremaps):
                 plt.subplot(6,8, featuremap+1) # sets the number of feature maps to show on eac
                 plt.title('FeatureMap ' + str(featuremap)) # displays the feature map number
                 if activation_min != -1 & activation_max != -1:
                     plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmin =ac
                 elif activation_max != -1:
                     plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmax=act
                 elif activation_min !=-1:
                     plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmin=act
            else:
                plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", cmap="gr
```

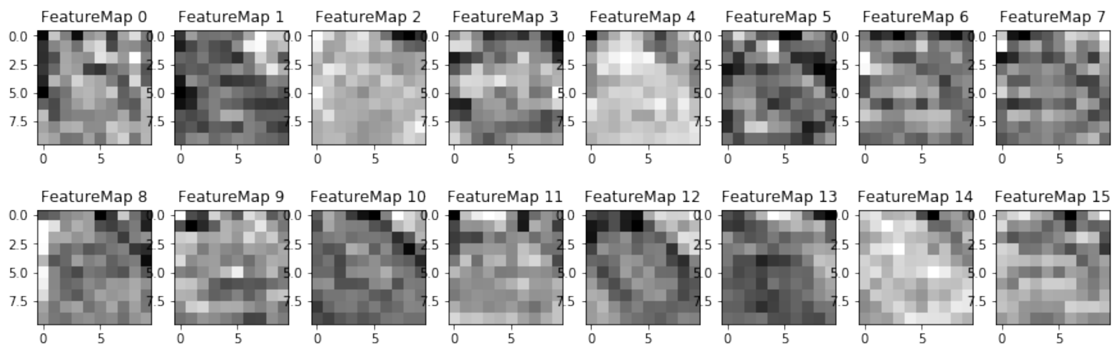
```
In [38]: image = X_train[index].reshape(1,32,32,3)
         with tf.Session() as sess:
             new_saver = tf.train.import_meta_graph('TrafficSignClassifier.meta')
             new_saver.restore(sess, tf.train.latest_checkpoint('.'))
             Tensor_names =[tensor.name for tensor in tf.get_default_graph().as_graph_def().node
             Tensor_operations = [tensor.op for tensor in tf.get_default_graph().as_graph_def().
             Conv2D = tf.get_default_graph().get_tensor_by_name('BiasAdd:0')
             sess.run(tf.global_variables_initializer())
             outputFeatureMap(image,Conv2D)
```

```
INFO:tensorflow:Restoring parameters from .\TrafficSignClassifier
```



```
In [39]: with tf.Session() as sess:
             new_saver = tf.train.import_meta_graph('TrafficSignClassifier.meta')
             new_saver.restore(sess, tf.train.latest_checkpoint('.'))
             Tensor_names =[tensor.name for tensor in tf.get_default_graph().as_graph_def().node
             Tensor_operations = [tensor.op for tensor in tf.get_default_graph().as_graph_def().
             Conv2D = tf.get_default_graph().get_tensor_by_name('BiasAdd_1:0')
             sess.run(tf.global_variables_initializer())
             outputFeatureMap(image,Conv2D)
```

```
INFO:tensorflow:Restoring parameters from .\TrafficSignClassifier
```


FeatureMap 0
FeatureMap 1
FeatureMap 2
FeatureMap 3
FeatureMap 4
FeatureMap 5
FeatureMap 6
FeatureMap 7
FeatureMap 8
FeatureMap 9
FeatureMap 10
FeatureMap 11
FeatureMap 12
FeatureMap 13
FeatureMap 14
FeatureMap 15