Self-Driving Car Engineer Nanodegree

Deep Learning

Project: Build a Traffic Sign Recognition Classifier

Completed by Mukil Kesavan

Step 0: Load The Data

In [1]:
```
# Load pickled data
import pickle

training_file = "data/train.p"
testing_file = "data/test.p"

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

XX_train, y_train = train['features'], train['labels']
XX_test, y_test = test['features'], test['labels']
```

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 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 (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.shape.html) might be useful for calculating some of the summary results.

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

In [2]:
```
### Replace each question mark with the appropriate value.
### Use python, pandas or numpy methods rather than hard coding the results

# Number of training examples
n_train = train['features'].shape[0]

# Number of testing examples.
n_test = test['features'].shape[0]

# What's the shape of an traffic sign image?
image_shape = train['features'][0].shape

# How many unique classes/labels there are in the dataset.
n_classes = len(set(y_train))

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

```
Number of training examples = 39209
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

Exploratory visualization of the dataset

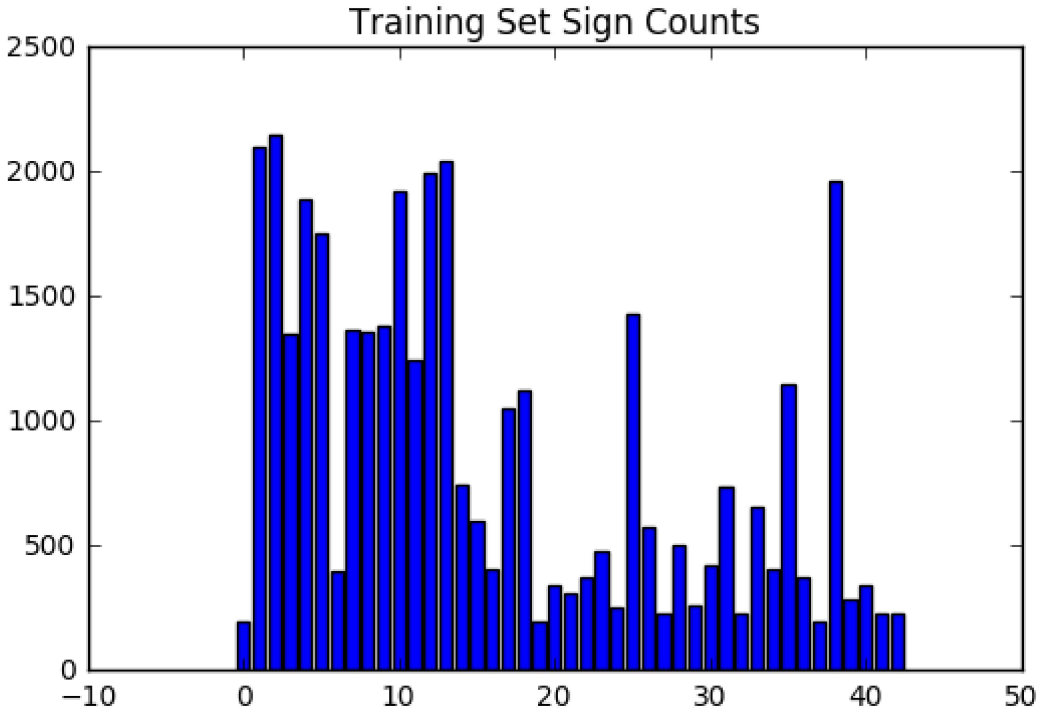
I have provided the following data visualizations to give some insights into the characteristics of the dataset:

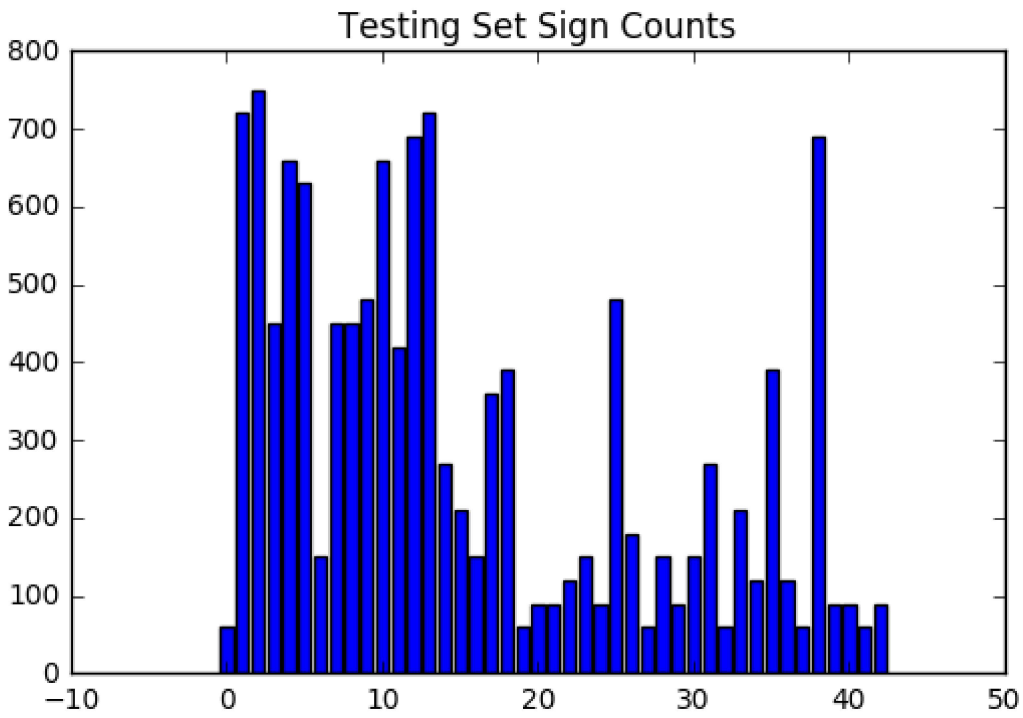
- Plot 5 random images from the Training set
- Show histogram of number of training set images of each sign type
- Show histogram of number of testing set images of each sign type

In [22]:
```
### Data exploration visualization code goes here.
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
import random
import numpy as np
# Visualizations will be shown in the notebook.
%matplotlib inline

def plotOneOfEachClass(images, labels):
    """ This function will plot a single image
    of each traffic sign type.
    """
    oneofeach = {}
    uniquelabels = np.unique(labels)
    for i in range(len(uniquelabels)):
        for j in range(len(labels)):
            if labels[j] == i:
                oneofeach[i] = j

    f = 1
    for v in oneofeach.values():
        fig = plt.figure(f, figsize = (2, 2))
        plt.axis("off")
        plt.imshow(images[v])
        f += 1

def plotImages(selImageIndices, images, labels, ptitle = None):
    """ This function will plot the images specified in the
    selImageIndices list parameter.
    """
    numImages = len(selImageIndices)
    fig = plt.figure()
    if ptitle is not None:
        fig.suptitle(ptitle, fontsize="x-large")
    ii = 1
    for si in selImageIndices:
        si = si % len(images)
        ax = fig.add_subplot(1, numImages, ii)
        ax.set_title(labels[si])
        plt.axis('off')
        plt.imshow(images[si].squeeze())
        ii += 1

def plotSignHistogram(labels, title):
```

```
    """ This function plots the histogram of number of
    samples of each traffic sign type.
    """
    indices, counts = np.unique(labels, return_counts = True)
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.bar(indices, counts, align = 'center')
    ax.set_title(title)
    plt.show()

randImageIndices = random.sample(range(len(XX_train)), 5)
plotImages(randImageIndices, XX_train, y_train, "Sample Images from Dataset")
plotSignHistogram(y_train, "Training Set Sign Counts")
plotSignHistogram(y_test, "Testing Set Sign Counts")
#plotOneOfEachClass(XX_train, y_train)
```

Sample Images from Dataset




Testing Set Sign Counts
800
700
600
500
400
300
200
100
0
−10
0
10
20
30
40
50

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 (http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset).

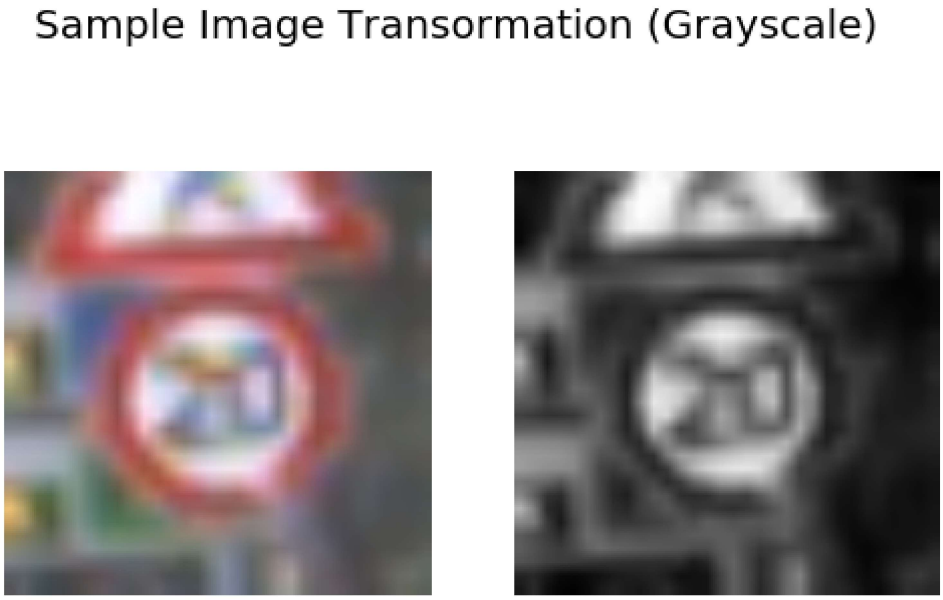
Here is the process I follow for classifying traffic signs:

1. **Preprocessing**: I first convert the training and testing set images from RGB to grayscale. I do this so that the neural network relies primarily on detecting shapes and contrast patterns for detecting traffic signs, instead of color. Colors tend to have different shades in different lighting conditions and I figured that this might be a source of noisy input to the neural network. Also, I did try normalization (mean shifting) of the images after grayscale transformation, but it did not improve my accuracy beyond any reasonable error margin. So, I've left that out of my pre-processing phase.
2. **Validation Set Splitting**: The pickled data provided with the project template has training and testing data. However, we still need a validation set to compute the loss function that will be used to adjust weights over multiple training epochs of the neural network. I split off 5% of images from the training set into the validation set for this purpose and shuffled them so that any inherent ordering of images does not affect learning.
3. **Neural Network Architecture**: I have used the LeNet-5 architecture, presented in class for the handwritten character recognition task, as the basis for classifying traffic signs with some modifications. In addition to changing the input layer and output layer shapes to fit the traffic sign dataset, I've also added dropout to convolutional and fully connected layers so as to prevent overfitting the training set. I used less aggressive droput (0.25) at the convolutional layers early on in the network to make sure that a good amount of input information is passed along the network and used more aggressive dropout (0.5) in the fully connected layers to prevent overfitting. This gave me a good 3-4% improvement in accuracy over the testing set.
4. **Training**: I used a batch size of 128 images, learning rate of 0.001 (with the more sophisticated adam optimizer) and 25 epochs to train the neural network. I experimented with different values and found that this combination gave me the best tradeoff between system memory usage and preventing overtraining (or overfitting to the training set well past any significant improvements to validation set accuracy).

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

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

In [23]:
```
#Convert Images to GrayScale

from skimage.color import rgb2gray
import numpy as np

def convertToGrayscale(img):
    img = rgb2gray(img)
    img.shape += (1, )
    return img

X_train = convertToGrayscale(XX_train)
X_test = convertToGrayscale(XX_test)

XP = [XX_train[0], X_train[0]]
yp = ["", ""]
plotImages([0, 1], XP, yp, "Sample Image Transormation (Grayscale)")
```



Split Data into Training, Validation and Testing Sets

In [5]:
```
### Split the data into training/validation/testing sets here.
### Feel free to use as many code cells as needed.

from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

X_train, X_validation, y_train, y_validation = train_test_split(
    X_train,
    y_train,
    test_size=0.05,
    random_state=832289)

X_train, y_train = shuffle(X_train, y_train)
X_validation, y_validation = shuffle(X_validation, y_validation)
```

Model Architecture

In [6]:
```
import tensorflow as tf
from tensorflow.contrib.layers import flatten
```

In [7]:
```
### Define your architecture here.
### Feel free to use as many code cells as needed.

keep_prob_conv = tf.placeholder(tf.float32)
keep_prob_fc = tf.placeholder(tf.float32)

def LeNet5(x):

    #Arguments for tf.truncated_normal
    mu = 0
    sigma = 0.1

    #Weights, Biases & Stride for Different Convolution Layers
    weights = {
        'l1': tf.Variable(tf.truncated_normal((5, 5, 1, 6), mu, sigma)),
        'l2': tf.Variable(tf.truncated_normal((5, 5, 6, 16), mu, sigma)),
        'l3': tf.Variable(tf.truncated_normal((400, 120), mu, sigma)),
        'l4': tf.Variable(tf.truncated_normal((120, 84), mu, sigma)),
        'l5': tf.Variable(tf.truncated_normal((84, 43), mu, sigma)),
    }

    biases = {
        'l1': tf.Variable(tf.zeros(6)),
        'l2': tf.Variable(tf.zeros(16)),
        'l3': tf.Variable(tf.zeros(120)),
        'l4': tf.Variable(tf.zeros(84)),
        'l5': tf.Variable(tf.zeros(43)),
    }

    cstride = 1
    cpadding = "VALID"

    #Pooling Layer Parameters
    pstride = 2
    pks = pstride
    ppadding = "VALID"

    #Layer 1: Convolve, add bias and activate - Input = 32x32x1, Output = 28x2
8x6
    out = tf.nn.conv2d(x, weights['l1'], strides = [1, cstride, cstride, 1], p
adding = cpadding)
    out = tf.nn.bias_add(out, biases['l1'])
    out = tf.nn.relu(out)
    out = tf.nn.dropout(out, keep_prob_conv)

    #Pooling Layer - Input = 28x28x6, Output = 14x14x6
    out = tf.nn.max_pool(out, ksize = [1, pks, pks, 1], strides = [1, pstride,
 pstride, 1], padding = ppadding)
```

```
    #Layer 2: Convolve, add bias and activate - Input = 14x14x6, Output = 10x1
0x16
    out = tf.nn.conv2d(out, weights['l2'], strides = [1, cstride, cstride, 1],
 padding = cpadding)
    out = tf.nn.bias_add(out, biases['l2'])
    out = tf.nn.relu(out)
    out = tf.nn.dropout(out, keep_prob_conv)

    #Pooling Layer - Input = 10x10x16, Output = 5x5x16
    out = tf.nn.max_pool(out, ksize = [1, pks, pks, 1], strides = [1, pstride,
 pstride, 1], padding = ppadding)

    #Flatten - Input = 5x5x16, Output = 400
    out = flatten(out)

    #Layer 3: Fully Connected. Input = 400, Output = 120
    out = tf.add(tf.matmul(out, weights['l3']), biases['l3'])
    out = tf.nn.relu(out)
    out = tf.nn.dropout(out, keep_prob_fc)

    #Layer 4: Fully Connected. Input = 120, Output = 84
    out = tf.add(tf.matmul(out, weights['l4']), biases['l4'])
    out = tf.nn.relu(out)
    out = tf.nn.dropout(out, keep_prob_fc)

    #Layer 5: Fully Connected. Input = 84, Output = 43
    logits = tf.add(tf.matmul(out, weights['l5']), biases['l5'])

    return logits
```

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 test set but low accuracy on the validation set implies overfitting.

In [8]:
```
### 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.

EPOCHS = 25
BATCH_SIZE = 128

x = tf.placeholder(tf.float32, (None, 32, 32, 1))
y = tf.placeholder(tf.int32, (None))
one_hot_y = tf.one_hot(y, 43)
```

In [9]:
```
LEARNING_RATE = 0.001

logits = LeNet5(x)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits, one_hot_y)
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate = LEARNING_RATE)
training_operation = optimizer.minimize(loss_operation)
```

In [10]:
```
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, kpc = 1.0, kpfc = 1.0):
    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:off
set+BATCH_SIZE]
        accuracy = sess.run(accuracy_operation, feed_dict={x: batch_x, y: batc
h_y, keep_prob_conv: kpc, keep_prob_fc: kpfc})
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples
```

In [11]:
```
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, ke
ep_prob_conv: 0.75, keep_prob_fc: 0.5})

        validation_accuracy = evaluate(X_validation, y_validation)
        print("EPOCH {} ...".format(i+1))
        print("Validation Accuracy = {:.3f}".format(validation_accuracy))
        print()

    print("Training Done!")
    saver.save(sess, './model/tsign')
    print("Model saved!")
```

```
Training...

EPOCH 1 ...
Validation Accuracy = 0.566

EPOCH 2 ...
Validation Accuracy = 0.777

EPOCH 3 ...
Validation Accuracy = 0.851

EPOCH 4 ...
Validation Accuracy = 0.881

EPOCH 5 ...
Validation Accuracy = 0.903

EPOCH 6 ...
Validation Accuracy = 0.917

EPOCH 7 ...
Validation Accuracy = 0.923

EPOCH 8 ...
Validation Accuracy = 0.936

EPOCH 9 ...
Validation Accuracy = 0.941

EPOCH 10 ...
Validation Accuracy = 0.952

EPOCH 11 ...
Validation Accuracy = 0.959

EPOCH 12 ...
Validation Accuracy = 0.957

EPOCH 13 ...
Validation Accuracy = 0.958

EPOCH 14 ...
Validation Accuracy = 0.961

EPOCH 15 ...
Validation Accuracy = 0.968

EPOCH 16 ...
Validation Accuracy = 0.970

EPOCH 17 ...
Validation Accuracy = 0.969

EPOCH 18 ...
Validation Accuracy = 0.972

EPOCH 19 ...
```

```
Validation Accuracy = 0.969

EPOCH 20 ...
Validation Accuracy = 0.977

EPOCH 21 ...
Validation Accuracy = 0.972

EPOCH 22 ...
Validation Accuracy = 0.975

EPOCH 23 ...
Validation Accuracy = 0.979

EPOCH 24 ...
Validation Accuracy = 0.978

EPOCH 25 ...
Validation Accuracy = 0.979

Training Done!
Model saved!
```

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

```
Test Accuracy = 0.929
```

Step 3: Test a Model on New Images

I have created 8 german traffic sign images by cutting out signs from images found freely on the web and youtube videos. I have made sure that at least some of the images have an angle to them (translation - e.g. i6, i8) and have non-optimal lighting (rainy and foggy conditions in i1 and i5). I've also manually classified these images and recorded them in new-signs/classes.csv. This is later used to compute the accuracy of the neural network for the new images. After that, I also output the top 3 predictions for each new image and their corresponding softmax probabilities.

Load and Output the Images

In [13]:
```
import csv
from scipy.misc import imread
from scipy.misc import imresize
import numpy as np

def loadScaledImage(imgpath):
    """This returns a numpy array representing
    the input image, scaled down to the 32x32
    input width and height expected by LeNet5.
    """
    img = imread(imgpath, mode='RGB')
    res = imresize(img, (32, 32))
    return res

def loadNewImageData(imgFilenames, imgClassesFile):
    """ Loads new images and their class labels.
    """
    y_new = []
    with open(imgClassesFile, mode='r') as cfile:
        reader = csv.reader(cfile)
        for line in reader:
            y_new.append(int(line[1].strip()))

    X_new = []
    for f in imgFilenames:
        img = loadScaledImage(f)
        X_new.append(img)
    return np.array(X_new), np.array(y_new)
```

In [14]:
```
### Load the images and plot them here.
### Feel free to use as many code cells as needed.

NEW_SIGN_FILES = [
    "new-signs/i1.png",
    "new-signs/i2.png",
    "new-signs/i3.png",
    "new-signs/i4.png",
    "new-signs/i5.png",
    "new-signs/i6.png",
    "new-signs/i7.png",
    "new-signs/i8.png"
]

NEW_SIGN_CLASSES_FILE = "new-signs/classes.csv"

XX_new, y_new = loadNewImageData(NEW_SIGN_FILES, NEW_SIGN_CLASSES_FILE)

plotImages(range(len(y_new)), XX_new, y_new)
```

In [15]:
```
#Pre-process Data

X_new = convertToGrayscale(np.array(XX_new))
```

Predict the Sign Type for Each Image

In [16]:
```
### Run the predictions here and use the model to output the prediction for ea
ch image.
### Make sure to pre-process the images with the same pre-processing pipeline
 used earlier.
### Feel free to use as many code cells as needed.

softmax_probs = tf.nn.softmax(logits)
top3 = tf.nn.top_k(softmax_probs, k = 3)

with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('./model'))
    top_preds = sess.run(top3, feed_dict={x: X_new, y: y_new, keep_prob_conv:
1.0, keep_prob_fc: 1.0})

preds = top_preds.indices[:, 0]
plotImages(range(len(preds)), XX_new, preds, "Sign Predictions")
```

Analyze Performance

In [17]:
```
### Calculate the accuracy for these 5 new images.
### For example, if the model predicted 1 out of 5 signs correctly, it's 20% a
ccurate on these new images.

correct_preds = 0
for i, j in zip(preds, y_new):
    correct_preds += int(i == j)
accuracy = float(correct_preds) / len(y_new)
print("Accuracy on New Images = {:.3f}".format(accuracy))
```

```
Accuracy on New Images = 0.875
```

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 (https://www.tensorflow.org/versions/r0.12/api_docs/python/nn.html#top_k) could prove helpful here.

In [19]:
```
### Print out the top five softmax probabilities for the predictions on the Ge
rman traffic sign images found on the web.
### Feel free to use as many code cells as needed.
from tabulate import tabulate

softmax_probs = list(zip(NEW_SIGN_FILES, top_preds.indices.tolist(),
top_preds.values.tolist()))

print(tabulate(softmax_probs, headers = ['Image', 'Predicted Classes', 'Softma
x Probabilities']))
```

```
Image             Predicted Classes    Softmax Probabilities
---------------  -------------------  ------------------------------------
----------------------------
new-signs/i1.png  [7, 5, 2]            [0.5513624548912048, 0.251869142055511
5, 0.09275676310062408]
new-signs/i2.png  [9, 12, 40]          [0.9989293217658997, 0.000600152590777
725, 0.00013770505029242486]
new-signs/i3.png  [17, 14, 12]         [0.9999732971191406, 2.601480628072749
8e-05, 4.348013931121386e-07]
new-signs/i4.png  [28, 29, 30]         [0.7324079871177673, 0.117915794253349
3, 0.05685333162546158]
new-signs/i5.png  [18, 26, 27]         [0.9278616905212402, 0.052106477320194
244, 0.01931988261640072]
new-signs/i6.png  [40, 16, 10]         [0.9268609881401062, 0.049990244209766
39, 0.009726230055093765]
new-signs/i7.png  [33, 1, 35]          [0.9991304278373718, 0.000196061184396
9673, 0.00017554273654241115]
new-signs/i8.png  [1, 2, 5]            [0.8214427828788757, 0.171836003661155
7, 0.006336591206490993 5]
```

> **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. You can then export the notebook by using the menu above and navigating to \n", "**File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

Project Writeup

I have exported this notebook along with outputs as an html file (report.html) in the my repository on github. I have also included a project write up, writeup.md in the repository. Here I explain the results we've seen in the notebook, factors that affect them and options for future improvement. Thanks for taking the time to review!