Self-Driving Car Engineer Nanodegree

Project: Build a Traffic Sign Recognition Classifier

Step 1: Load The Data

In [1]:

```
# Load pickled data
import pickle
import pandas as pd

# Data's location
training_file = "../data/train.p"
validation_file = "../data/valid.p"
testing_file = "../data/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)

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

# Sign id<->name mapping
sign_names = pd.read_csv('signnames.csv').to_dict(orient='index')
sign_names = { key : val['SignName'] for key, val in sign_names.items() }
print ("Data read")
```

```
Data read
```

Step 2: Dataset Summary & Exploration

Step 2.1: Retrieve Summary From DataSet

In [2]:

```
### Replace each question mark with the appropriate value.
### Use python, pandas or numpy methods rather than hard coding the results
import numpy as np

# Number of training examples
n_train = len(X_train)

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

# Number of validation examples.
n_valid = len(X_valid)

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

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

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

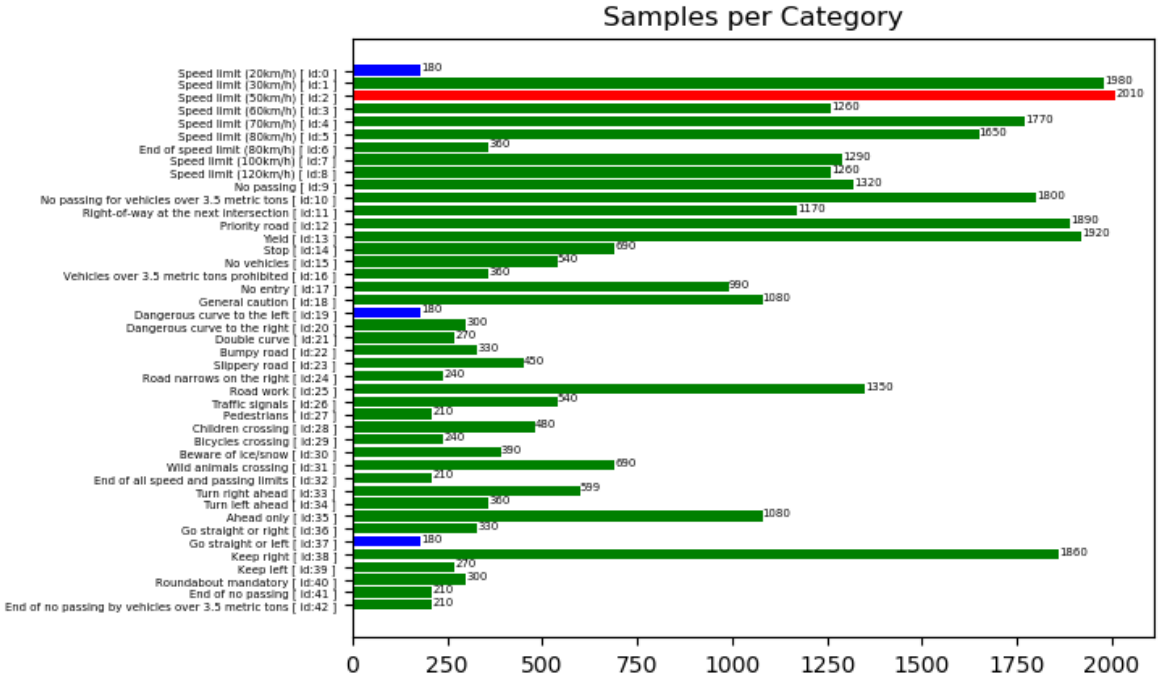
```
Number of training examples   =  34799
Number of testing examples    =  12630
Number of validation examples =  4410
Image data shape              =  (32, 32, 3)
Number of classes             =  43
```

Step 2.1: Visualizing the dataset

The categories with minimum/maximum number of samples are marked with yellow/red color correspondingly. (Seems the minimum appears 3 times within the data)

In [4]:

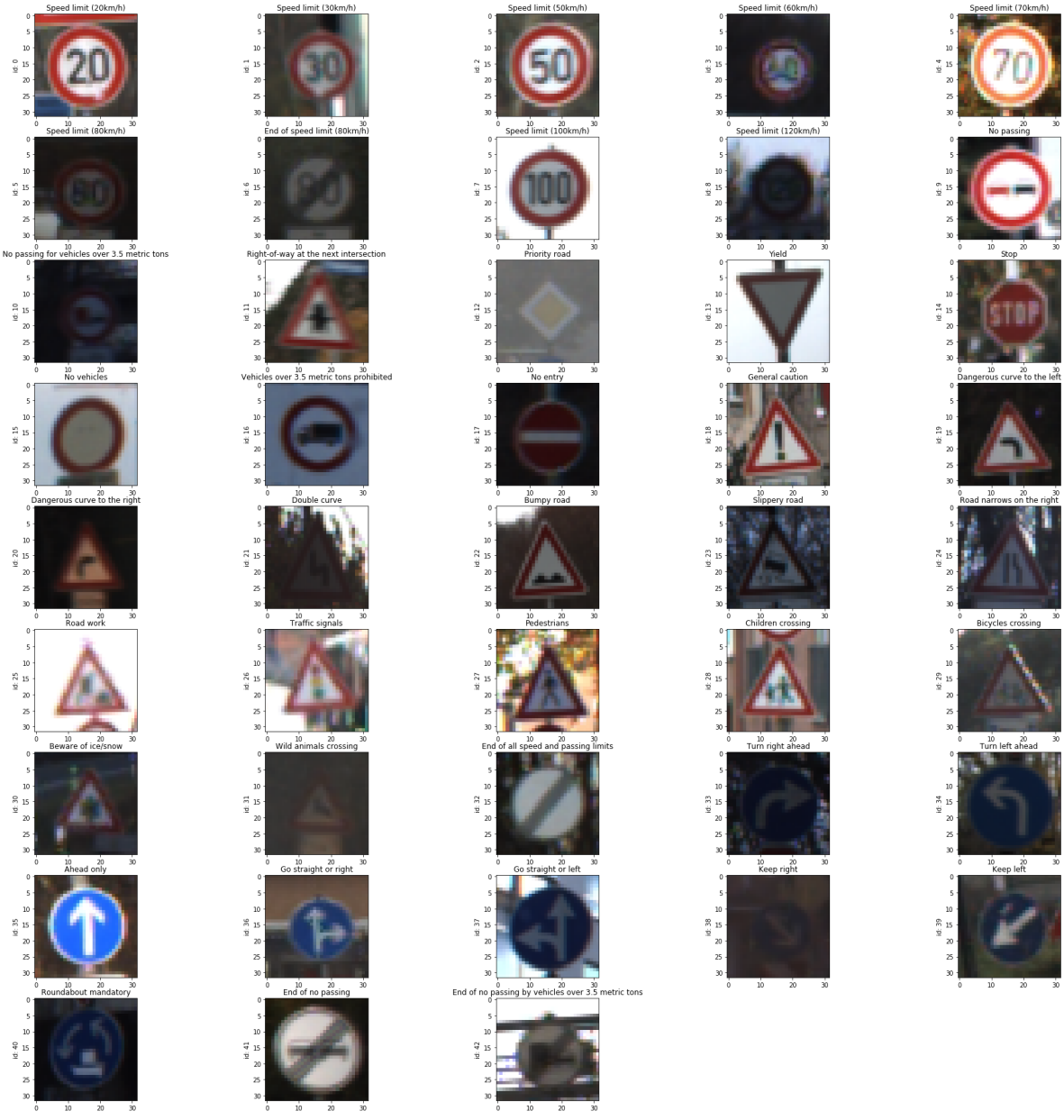
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

plt.rcdefaults()
fig, ax = plt.subplots()

samples_per_category = [len(np.where(y_train==cat_id)[0]) for cat_id in sign_names.keys()]
category_names = tuple([val + " [ id:{id} ]".format(id=key) for key,val in sign_names.items

# retrieve min & max from data
min_cnt = min(samples_per_category)
max_cnt = max(samples_per_category)

y_pos = np.arange(len(category_names))

rects = ax.barh(y_pos, samples_per_category, align='center',
                color=['green' if val != min_cnt and val != max_cnt \
                       else 'blue' if val == min_cnt \
                       else 'red' for val in samples_per_category])

# setting labels for each bar
for i in range(0,len(rects)):
    ax.text(int(rects[i].get_width()),
            int(rects[i].get_y()+rects[i].get_height()/2.0),
            samples_per_category[i],
            fontproperties=fm.FontProperties(size=5))

ax.set_yticks(y_pos)
ax.set_yticklabels(category_names,fontproperties=fm.FontProperties(size=5))
ax.invert_yaxis()
ax.set_title('Samples per Category')

plt.show()
```



Step 2.3: Showing Random Images

In [5]:

```
import random
import numpy as np
import matplotlib.pyplot as plt
import math

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

h_or_w = image_shape[0]
fig = plt.figure(figsize=(h_or_w,h_or_w))
for i in range(0, n_classes):
    samples = np.where(y_train==i)[0]
    index = random.randint(0, len(samples) - 1)
    image = X_train[samples[index]]

    ax = fig.add_subplot(math.ceil(n_classes/5), 5, i+1)
    ax.set_title(sign_names[i])
    ax.set_ylabel("id: {id}".format(id=i))
    plt.imshow(image)
plt.show()
```



Step 3: Design and Test a Model Architecture

Design and implement a deep learning model that learns to recognize traffic signs. The LeNet-5 CNN architecture is used here with minor modifications: dropout parameter added to the first fully connected layer.

Step 3.1: Pre-process the Data Set (normalization, grayscale, etc.)

Step 3.1.1: Shuffle Data

In [6]:

```
from sklearn.utils import shuffle

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

Step 3.1.2: Prepare Images

In [7]:

```
import cv2

def prepare_image(image_set):
    ###   1:normalize image
    ###   2: convert RGB image to gray scale

    # initialize empty image set for prepared images
    new_shape = image_shape[0:2] + (1,)

    prep_image_set = np.empty(shape=(len(image_set),) + new_shape, dtype=int)

    for ind in range(0, len(image_set)):
        # normalize
        norm_img = cv2.normalize(image_set[ind], np.zeros(image_shape[0:2]), 0, 255, cv2.NC

        # grayscale
        gray_img = cv2.cvtColor(norm_img, cv2.COLOR_RGB2GRAY)

        # set new image to the corresponding position
        prep_image_set[ind] = np.reshape(gray_img, new_shape)

    return prep_image_set

def equalize_number_of_samples(image_set, image_labels):
    ###Make number of samples in each category equal.

    num = max([len(np.where(image_labels==cat_id)[0]) for cat_id in sign_names.keys()])

    equalized_image_set = np.empty(shape=(num * n_classes,) + image_set.shape[1:], dtype=ir
    equalized_image_labels = np.empty(shape=(num * n_classes,), dtype=int)
    j = 0

    for cat_id in sign_names.keys():
        cat_inds = np.where(y_train==cat_id)[0]
        cat_inds_len = len(cat_inds)

        for i in range(0, num):
            equalized_image_set[j] = image_set[cat_inds[i % cat_inds_len]]
            equalized_image_labels[j] = image_labels[cat_inds[i % cat_inds_len]]
            j += 1

    # at this stage data is definitely not randomly shuffled, so shuffle it
    return shuffle(equalized_image_set, equalized_image_labels)

X_train_prep = prepare_image(X_train)
X_test_prep = prepare_image(X_test)
X_valid_prep = prepare_image(X_valid)

X_train_prep, y_train_prep = equalize_number_of_samples(X_train_prep, y_train)
# we do not need to transform labes for validation and test sets
y_test_prep = y_test
y_valid_prep = y_valid

image_shape_prep = X_train_prep[0].shape
```

Step 3.2: Define The Model

In [8]:

```
#### here the magic happens!

# LeNet-5 architecture is used.

import tensorflow as tf
from tensorflow.contrib.layers import flatten

def LeNet(x, channels, classes, keep_prob, mu=0, sigma=0.01):
    # Arguments used for tf.truncated_normal, randomly defines variables
    # for the weights and biases for each layer

    # Layer 1: Convolutional. Input = 32x32xchannels. Output = 28x28x6.
    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, channels, 6), mean = mu, stddev
    conv1_b = tf.Variable(tf.zeros(6))
    conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv1_b

    # Layer 1: Activation.
    conv1 = tf.nn.relu(conv1)

    # Layer 1: Pooling. Input = 28x28x6. Output = 14x14x6.
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID'

    # Layer 2: Convolutional. Output = 10x10x16.
    conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean = mu, stddev = sigm
    conv2_b = tf.Variable(tf.zeros(16))
    conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VALID') + conv2_b

    # Layer 2: Activation.
    conv2 = tf.nn.relu(conv2)

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

    # Layer 2: Flatten. Input = 5x5x16. Output = 400.
    fc0   = flatten(conv2)
    fc0   = tf.nn.dropout(fc0, keep_prob=keep_prob)

    # 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

    # Layer 3: Activation.
    fc1    = tf.nn.relu(fc1)

    # 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

    # Layer 4: Activation.
    fc2    = tf.nn.relu(fc2)

    # Layer 5: Fully Connected. Input = 84. Output = 10.
    fc3_W  = tf.Variable(tf.truncated_normal(shape=(84, classes), mean = mu, stddev = sigma
    fc3_b  = tf.Variable(tf.zeros(classes))
    logits = tf.matmul(fc2, fc3_W) + fc3_b

    return logits
```

Step 3.3: 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.

Step 3.3.1: Features & Labels

In [9]:

```
import tensorflow as tf

# x is a placeholder for a batch of input images
x = tf.placeholder(tf.float32, (None,) + image_shape_prep)

# y is a placeholder for a batch of output labels
y = tf.placeholder(tf.int32, (None))
one_hot_y = tf.one_hot(y, n_classes)
```

Step 3.3.2: Training Pipeline

In [10]:

```
# definition of the hyperparameters of the training process
BATCH_SIZE = 128        #### Batch Size; may need to be adapted to the memory available
#EPOCHS = 50             #### Epochs used
EPOCHS = 39             #### Epochs used -- 39 showed highest rate
RATE = 0.0009           #### Learning Rate
KEEP_PROB = 0.7
STDDEV = 0.01
```

In [11]:

```
keep_prob = tf.placeholder(tf.float32)
logits = LeNet(x, image_shape_prep[-1], n_classes, keep_prob, sigma=STDDEV)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot_y, logits=logits)
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate = RATE)
training_operation = optimizer.minimize(loss_operation)
```

Step 3.3.3: Evaludation Of The Model

In [12]:

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

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

Step 3.3.4: Training Of The Model

In [13]:

```
model_file='./model.sav'
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    num_examples = len(X_train_prep)

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

        train_accuracy = evaluate(X_train_prep, y_train_prep)
        validation_accuracy = evaluate(X_valid_prep, y_valid_prep)
        print("EPOCH {} ...".format(i+1))
        print("Train Accuracy = {:.3f}".format(train_accuracy))
        print("Validation Accuracy = {:.3f}".format(validation_accuracy))
        print()

    print("Training finished!")
    saver.save(sess, model_file)
    print("Model saved")
```

```
Validation Accuracy = 0.949

EPOCH 34 ...
Train Accuracy = 0.998
Validation Accuracy = 0.951

EPOCH 35 ...
Train Accuracy = 0.999
Validation Accuracy = 0.958

EPOCH 36 ...
Train Accuracy = 0.998
Validation Accuracy = 0.942

EPOCH 37 ...
Train Accuracy = 0.999
Validation Accuracy = 0.951

EPOCH 38 ...
Train Accuracy = 1.000
```

Step 3.3.5: Evaluate Trained Model Using Test Samples

In [14]:

```
with tf.Session() as sess:
    saver.restore(sess, model_file)

    test_accuracy = evaluate(X_test_prep, y_test_prep)
    print("Test Accuracy = {:.3f}".format(test_accuracy))
```

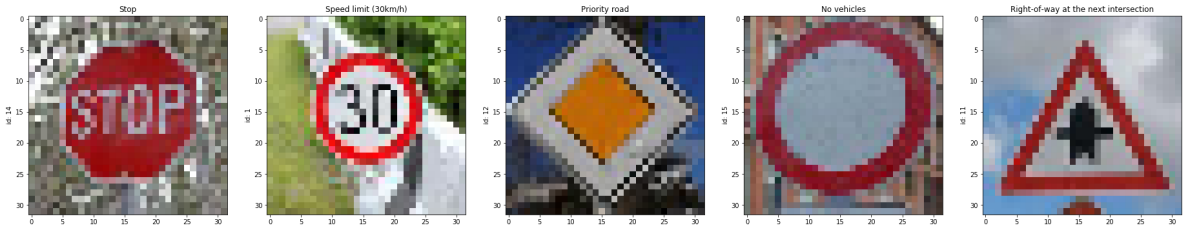
```
INFO:tensorflow:Restoring parameters from ./model.sav
Test Accuracy = 0.937
```

Step 4: Test a Model on New Images

Step 4.1: Load and Output the Images

In [15]:

```
import os
import cv2
import matplotlib.image as mpimg

#### internet found data store in ./test-signs
other_test_data_dir="./test-signs"
img_paths = os.listdir(other_test_data_dir)
images = list()
labels = list()

# read images and resize
for img_path in img_paths:
    # read image from file
    if (img_path != '.ipynb_checkpoints'):
        img = mpimg.imread(os.path.join(other_test_data_dir, img_path))
        img = cv2.resize(img, image_shape[0:2], interpolation=cv2.INTER_CUBIC)
        images.append(img)

        # prefix of each image name is a number of its category
        labels.append(int(img_path[0:img_path.find('-')]))
images = np.array(images)
labels = np.array(labels)


# output the resized images
h_or_w = image_shape[0]
fig = plt.figure(figsize=(h_or_w,h_or_w))
for i in range(0, len(images)):
    ax = fig.add_subplot(1, len(images), i+1)
    ax.set_title(sign_names[labels[i]])
    ax.set_ylabel("id: {id}".format(id=labels[i]))
    plt.imshow(images[i])
plt.show()
```



Step 4.2: Predict the Sign Type for Each Image

In [16]:

```
### 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 earlier.
### Feel free to use as many code cells as needed.
# preprocess images first
images_prep = prepare_image(images)
labels_prep = labels

# then make a prediction
with tf.Session() as sess:
    saver.restore(sess, model_file)
    sign_ids = sess.run(tf.argmax(logits, 1), feed_dict={x: images_prep, y: labels_prep, ke

# output the results in the table
print('-' * 93)
print("| {p:^43} | {a:^43} |".format(p='PREDICTED', a='ACTUAL'))
print('-' * 93)
for i in range(len(sign_ids)):
    print('| {p:^2} {strp:^40} | {a:^2} {stra:^40} |'.format(
        p=sign_ids[i], strp=sign_names[sign_ids[i]], a=labels[i], stra=sign_names[labels[i]
print('-' * 93)
```

```
INFO:tensorflow:Restoring parameters from ./model.sav
---------------------------------------------------------------------------
-----------------
|                  PREDICTED                  |                  ACTUAL
|
---------------------------------------------------------------------------
-----------------
| 14                   Stop                   | 14                   Stop
|
| 11  Right-of-way at the next intersection   | 1            Speed limit (30
km/h)           |
| 12              Priority road               | 12              Priority roa
d               |
| 32   End of all speed and passing limits    | 15               No vehicles
|
| 11  Right-of-way at the next intersection   | 11  Right-of-way at the next
intersection   |
---------------------------------------------------------------------------
-----------------
```

Step 4.3: 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% accurate on th

### Calculate the accuracy for these 5 new images.
### For example, if the model predicted 1 out of 5 signs correctly, it's 20% accurate on th
# run evaluation on the new images

with tf.Session() as sess:
    saver.restore(sess, model_file)

    test_accuracy = evaluate(images_prep, labels_prep)
    print("Accuracy = {:.3f}".format(test_accuracy))
```

```
INFO:tensorflow:Restoring parameters from ./model.sav
Accuracy = 0.600
```

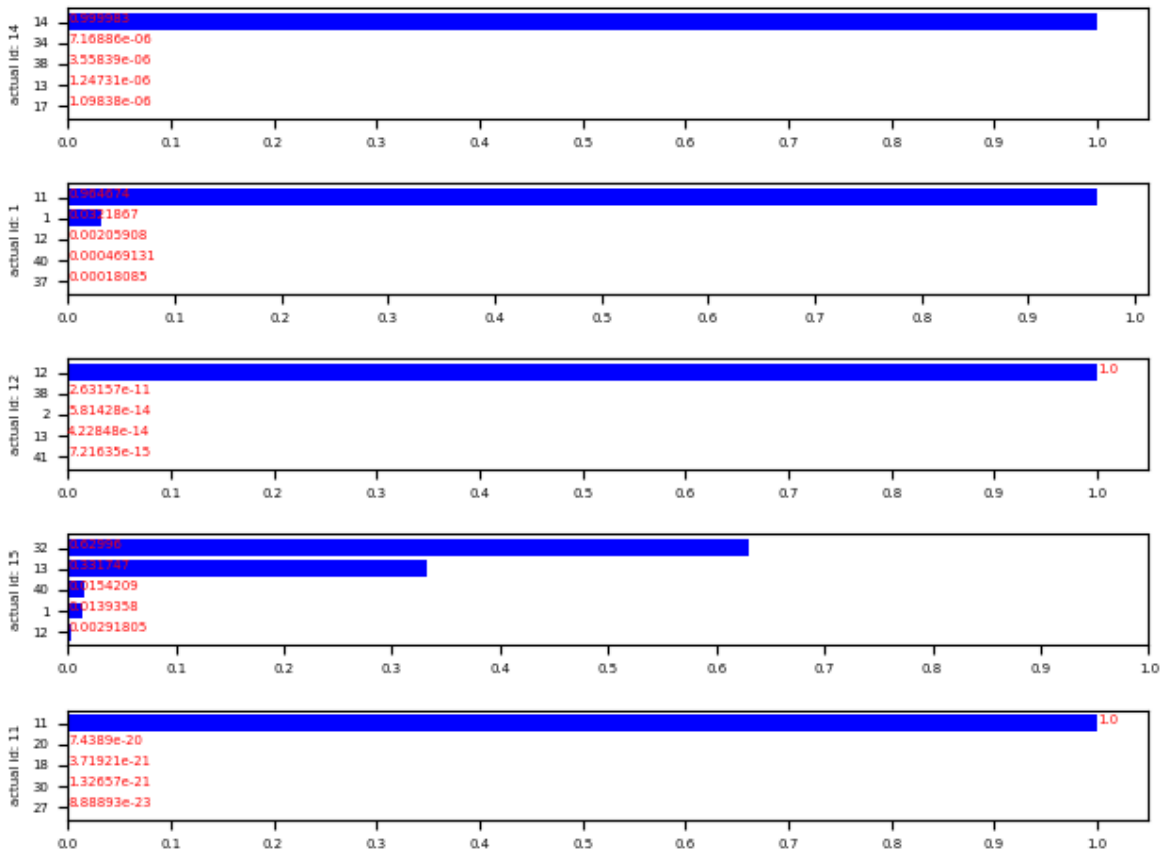
Step 4.4: Output Top 5 Softmax Probabilities For Each Image Found on the Web

In [18]:

```
### Print out the top five softmax probabilities for the predictions on the German traffic
### Feel free to use as many code cells as needed.

### Print out the top five softmax probabilities for the predictions on
### the German traffic sign images found on the web.

with tf.Session() as sess:
    saver.restore(sess, model_file)
    top_k = sess.run(tf.nn.top_k(tf.nn.softmax(logits), k=5),
                     feed_dict={x: images_prep, y: labels_prep, keep_prob: 1})
print(top_k)

plt.rcdefaults()

# show histogram of top 5 softmax probabilities for each image
h_or_w = image_shape[0]
fig = plt.figure()
for i in range(0, len(images)):
    ax = fig.add_subplot(len(images), 1, i+1)
    probabilities = top_k.values[i]
    y_pos = np.arange(len(probabilities))
    ax.set_ylabel("actual id: {id}".format(id=labels[i]), fontproperties=fm.FontProperties(
    rects = ax.barh(y_pos,
                probabilities,
                align='center',
                color='blue')
    # setting labels for each bar
    for j in range(0,len(rects)):
        ax.text(int(rects[j].get_width()),
                int(rects[j].get_y()+rects[j].get_height()/2.0),
                probabilities[j],
                fontproperties=fm.FontProperties(size=5), color='red')

    ax.set_yticks(y_pos)
    ax.set_yticklabels(top_k.indices[i], fontproperties=fm.FontProperties(size=5))

    xticks = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
    ax.set_xticks(xticks)
    ax.set_xticklabels(xticks, fontproperties=fm.FontProperties(size=5))
    ax.invert_yaxis()
plt.tight_layout()
plt.show()
```

```
INFO:tensorflow:Restoring parameters from ./model.sav
TopKV2(values=array([[  9.99982953e-01,   7.16885916e-06,   3.55839256e-0
6,
          1.24730627e-06,   1.09837822e-06],
       [  9.64673877e-01,   3.21867019e-02,   2.05908273e-03,
          4.69130697e-04,   1.80849602e-04],
       [  1.00000000e+00,   2.63156580e-11,   5.81428284e-14,
          4.22847555e-14,   7.21634632e-15],
       [  6.29960179e-01,   3.31746906e-01,   1.54209370e-02,
          1.39357755e-02,   2.91805272e-03],
       [  1.00000000e+00,   7.43889793e-20,   3.71921123e-21,
          1.32657081e-21,   8.88892886e-23]], dtype=float32), indices=arr
ay([[14, 34, 38, 13, 17],
       [11,  1, 12, 40, 37],
```

```
       [12, 38,  2, 13, 41],
       [32, 13, 40,  1, 12],
       [11, 20, 18, 30, 27]], dtype=int32))
```



In [ ]:

In [ ]: