Self-Driving Car Engineer Nanodegree

Deep Learning

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 \n", "**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 (https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project/blob/master/writeup_template.md) that can be used to guide the writing process. Completing the code template and writeup template will cover all of the rubric points (https://review.udacity.com/#!/rubrics/481/view) for this project.

The rubric (https://review.udacity.com/#!/rubrics/481/view) 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.

Step 0: Load The Data

In [662]:
```
import numpy as np
import cv2
import pickle
from sklearn.utils import shuffle

import random
import tensorflow as tf
from tensorflow.contrib.layers import flatten
import glob
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline




# TODO: Fill this in based on where you saved the training and testin
g data

training_file = 'dataset1/train.p'
validation_file= 'dataset1/valid.p'
testing_file = 'dataset1/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_validation, y_validation = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']

print(type(X_train), type(y_train))
assert(len(X_train) == len(y_train))
assert(len(X_validation) == len(y_validation))
assert(len(X_test) == len(y_test))

print()
print("Image Shape: {}".format(X_train[0].shape))
print()
print("Training Set:   {} samples".format(len(X_train)))
print("Validation Set: {} samples".format(len(X_validation)))
print("Test Set:       {} samples".format(len(X_test)))
n_classes = len(np.unique(y_train))
print(n_classes)
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'>

Image Shape: (32, 32, 3)

Training Set:   34799 samples
Validation Set: 4410 samples
Test Set:       12630 samples
43
```

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 (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 [663]:
```python
### Replace each question mark with the appropriate value.
### Use python, pandas or numpy methods rather than hard coding the r
esults

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

# TODO: Number of validation examples
n_validation = len(X_validation)

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

# TODO: What's the shape of an traffic sign image?
image_shape = X_train[0].shape

# TODO: 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("Image data shape =", image_shape)
print("Number of classes =", n_classes)
```

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

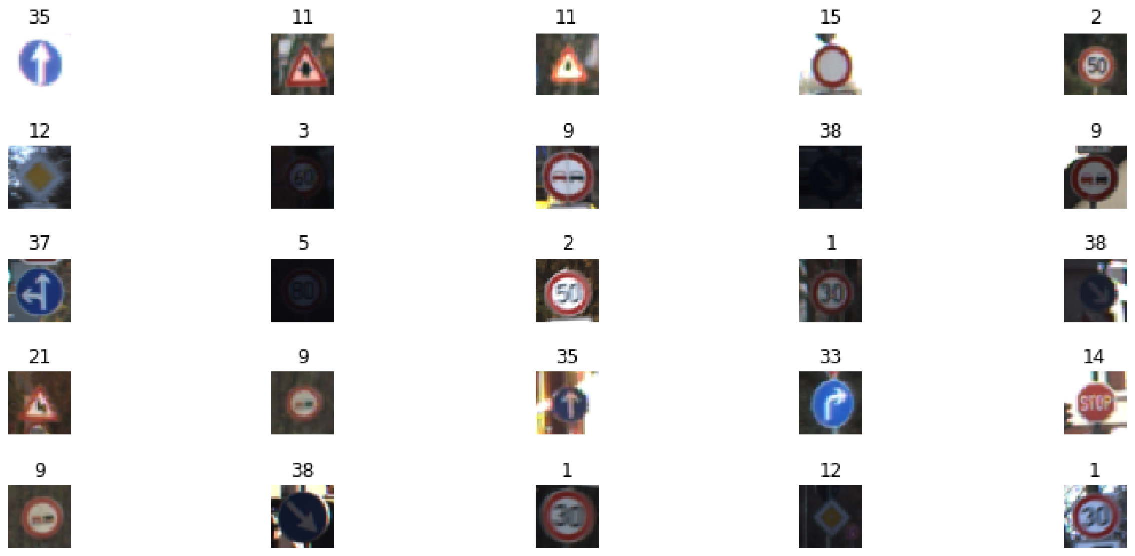
Include an exploratory visualization of the dataset

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 (http://matplotlib.org/) examples (http://matplotlib.org/examples/index.html) and gallery (http://matplotlib.org/gallery.html) 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?

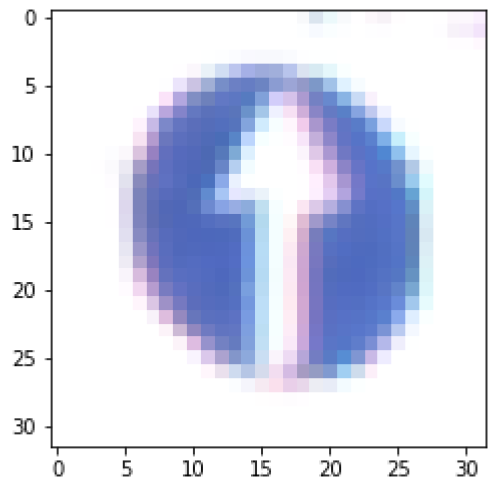
In [664]:
```python
random_images = []
random_indices = []

fig, axs = plt.subplots(5,5, figsize=(15, 6))
fig.subplots_adjust(hspace = .8, wspace=.001)
axs = axs.ravel()
for i in range(25):
    index = random.randint(0, len(X_train))
    random_images.append(X_train[index])
    #print(random_images.shape)
    random_indices.append(index)
    axs[i].axis('off')
    axs[i].imshow(random_images[i])
    axs[i].set_title(y_train[index])
```

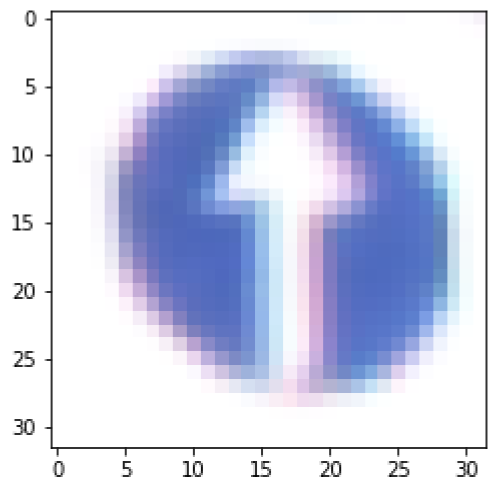


In [665]:

```
# affine transform

img = random_images[0]
def affine_transform(img):
    rows,cols = img.shape[:2]
    tx,ty = np.random.randint(0,4,2)

    pts1 = np.float32([[tx,ty],[rows-tx,ty],[ty,cols-ty]])
    pts2 = np.float32([[0,0],[rows,0],[0,cols]])

    M = cv2.getAffineTransform(pts1,pts2)

    dst = cv2.warpAffine(img,M,(cols,rows))

    res = cv2.resize(dst,(cols, rows), interpolation = cv2.INTER_CUBI
C)

    return res


plt.figure()
plt.imshow(img)
plt.show

plt.figure()
plt.imshow(affine_transform(img))
plt.show
```

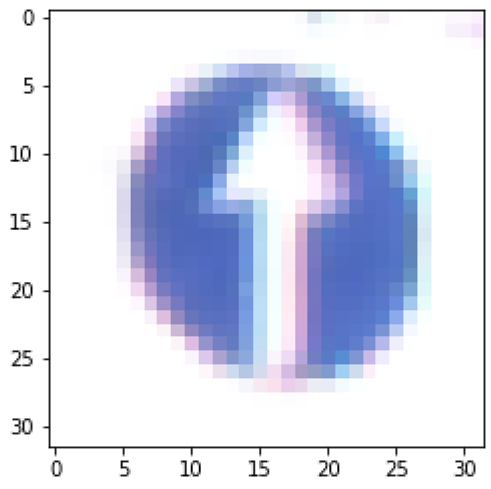
```
Out[665]: <function matplotlib.pyplot.show(*args, **kw)>
```

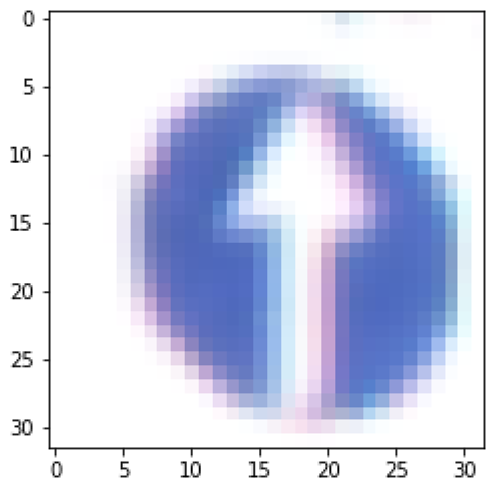




In [666]:
```
#perspective transform

def perspective_transform(img):
    rows,cols = img.shape[:2]

    # transform limits
    begin = np.random.randint(0,4)
    end = np.random.randint(-4, 0)

    pts1 = np.float32([ [begin, begin], [rows+end, begin], [begin, co
ls+end], [rows+end, cols+end] ])
    pts2 = np.float32([[0,0],[rows,0],[0,cols],[rows,cols]])

    M = cv2.getPerspectiveTransform(pts1,pts2)
    dst = cv2.warpPerspective(img,M,(rows,cols))

    return dst

plt.figure()
plt.imshow(img, cmap='gray')
plt.show

plt.figure()
plt.imshow(perspective_transform(img), cmap='gray')
plt.show
```

```
Out[666]: <function matplotlib.pyplot.show(*args, **kw)>
```





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).

The LeNet-5 implementation shown in the classroom (https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/6df7ae49-c61c-4bb2-a23e-6527e69209ec/lessons/601ae704-1035-4287-8b11-e2c2716217ad/concepts/d4aca031-508f-4e0b-b493-e7b706120f81) 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 (http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf). 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.

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.

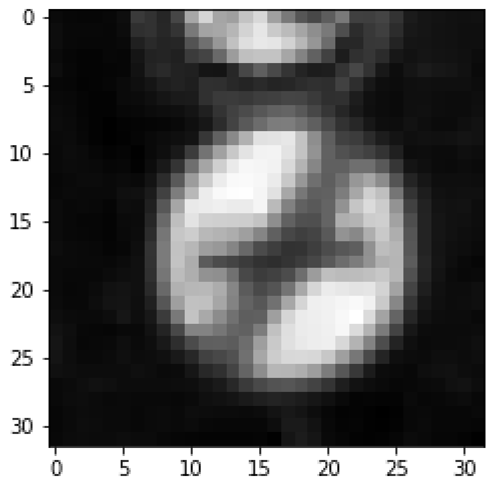
Pre processing data

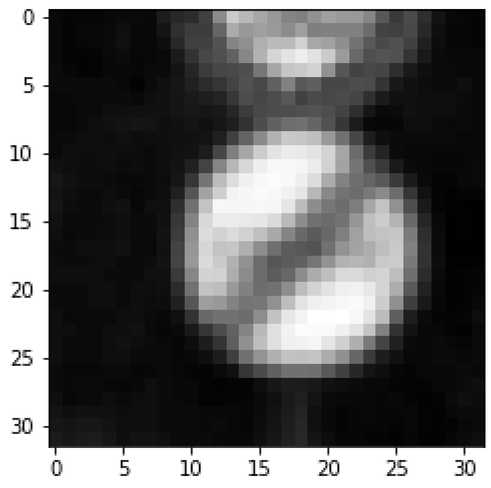
Convert to Grayscale

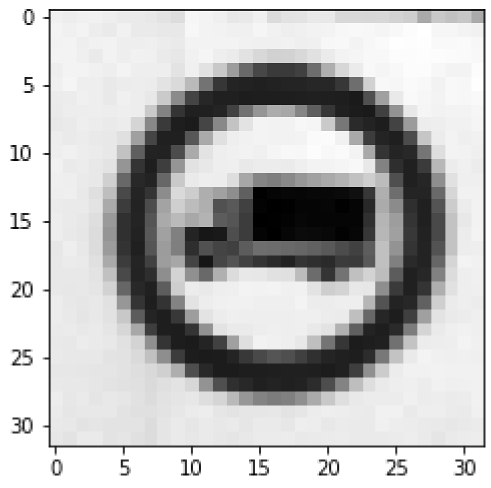
In [667]:
```
X_train_original = X_train
X_train_gray = np.sum(X_train/3, axis=3, keepdims=True)

X_validation_original = X_validation
X_validation_gray = np.sum(X_validation/3, axis=3, keepdims=True)

X_test_original = X_test
X_test_gray = np.sum(X_test/3, axis=3, keepdims=True)

print('X_train RGB shape:', X_train_original.shape)
print('X_train Grayscale shape:', X_train_gray.shape)
print()

print('X_validation RGB shape:', X_validation_original.shape)
print('X_validation Grayscale shape:', X_validation_gray.shape)
print()

print('X_test RGB shape:', X_test_original.shape)
print('X_test Grayscale shape:', X_test_gray.shape)
```

```
X_train RGB shape: (34799, 32, 32, 3)
X_train Grayscale shape: (34799, 32, 32, 1)

X_validation RGB shape: (4410, 32, 32, 3)
X_validation Grayscale shape: (4410, 32, 32, 1)

X_test RGB shape: (12630, 32, 32, 3)
X_test Grayscale shape: (12630, 32, 32, 1)
```

```
In [668]: plt.figure()
          plt.imshow(X_train_gray[0].squeeze(), cmap='gray')

          plt.figure()
          plt.imshow(X_validation_gray[0].squeeze(), cmap='gray')

          plt.figure()
          plt.imshow(X_test_gray[0].squeeze(), cmap='gray')

Out[668]: <matplotlib.image.AxesImage at 0x7fbf189609b0>
```







Normalization

In [669]:
```
print(np.mean(X_train_gray))
print(np.mean(X_validation_gray))
print(np.mean(X_test_gray))
```

```
82.677589037
83.5564273756
82.1484603612
```

In [670]:
```
## Normalize the train, validation and test datasets to (-1,1)## Norm
a

X_train_normalized = (X_train_gray - 128)/128
y_train_normalized = y_train

X_validation_normalized = (X_validation_gray - 128)/128
y_validation_normalized = y_validation

X_test_normalized = (X_test_gray - 128)/128
y_test_normalized = y_test

print(np.mean(X_train_normalized))
print(np.mean(X_validation_normalized))
print(np.mean(X_test_normalized))
```

```
-0.354081335648
-0.347215411128
-0.358215153428
```

```
In [671]: fig, axs = plt.subplots(1,2, figsize=(10, 3))
          axs = axs.ravel()

          axs[0].axis('off')
          axs[0].set_title('normalized')
          axs[0].imshow(X_train_normalized[0].squeeze(), cmap='gray')

          axs[1].axis('off')
          axs[1].set_title('original')
          axs[1].imshow(X_train[0])
```

```
Out[671]: <matplotlib.image.AxesImage at 0x7fbf1875c6a0>
```

Augmenting training data

```
In [672]: # Backup all the datasets, just in case

          X_train_original = X_train
          X_validation_original = X_validation
          X_test_original = X_test
```

In [673]:
```
# Creating new numpy arrays to augment data

X_train_at = np.empty(X_train_normalized.shape)
y_train_at = np.empty(y_train.shape)

X_train_pt = np.empty(X_train_normalized.shape)
y_train_pt = np.empty(y_train.shape)

print("Normalized X Shape: ", X_train_normalized.shape)
print("Normalized y Shape: ", y_train.shape)
print("AT X Shape: ", X_train_at.shape)
print("AT y Shape: ", y_train_at.shape)
print("PT X Shape: ", X_train_pt.shape)
print("PT y Shape: ", y_train_pt.shape)
```

```
Normalized X Shape:  (34799, 32, 32, 1)
Normalized y Shape:  (34799,)
AT X Shape:  (34799, 32, 32, 1)
AT y Shape:  (34799,)
PT X Shape:  (34799, 32, 32, 1)
PT y Shape:  (34799,)
```

In [674]:
```
# Tranforming all images randomly using affine_transform using normal
ized data set

for i in range(len(X_train_at)):
    X_train_at[i] = affine_transform(X_train_normalized[i]).reshape(3
2, 32, 1)
    y_train_at[i] = y_train[i]
```

In [675]:
```
# Tranforming all images randomly using perspective_transform using n
ormalized data set

for i in range(len(X_train_pt)):
    X_train_pt[i] = perspective_transform(X_train_normalized[i]).resh
ape(32, 32, 1)
    y_train_pt[i] = y_train[i]
```

In [676]:
```
# Concatenating all data into final data set

X_train_total = np.concatenate((X_train_normalized, X_train_at, X_tra
in_pt))
y_train_total = np.concatenate((y_train, y_train, y_train))
```

In [677]:
```
print(X_train_total.shape)
print(y_train_total.shape)
```

```
(104397, 32, 32, 1)
(104397,)
```

```
In [678]: X_train = X_train_total
          y_train = y_train_total

          X_validation = X_validation_normalized
          X_test = X_test_normalized
```

Model Architecture

Setup TensorFlow

The `EPOCH` and `BATCH_SIZE` values affect the training speed and model accuracy.

```
In [679]: EPOCHS = 100
          BATCH_SIZE = 128
```

SOLUTION: Implement LeNet-5

Implement the LeNet-5 (http://yann.lecun.com/exdb/lenet/) neural network architecture.

This is the only cell you need to edit.

Input

The LeNet architecture accepts a 32x32xC image as input, where C is the number of color channels. Since MNIST images are grayscale, C is 1 in this case.

Architecture

Layer 1: Convolutional. The output shape should be 28x28x6.

Activation. Your choice of activation function.

Pooling. The output shape should be 14x14x6.

Layer 2: Convolutional. The output shape should be 10x10x16.

Activation. Your choice of activation function.

Pooling. The output shape should be 5x5x16.

Flatten. Flatten the output shape of the final pooling layer such that it's 1D instead of 3D. The easiest way to do is by using `tf.contrib.layers.flatten`, which is already imported for you.

Layer 3: Fully Connected. This should have 120 outputs.

Activation. Your choice of activation function.

Layer 4: Fully Connected. This should have 84 outputs.

Activation. Your choice of activation function.

Layer 5: Fully Connected (Logits). This should have 10 outputs.

Output

Return the result of the 2nd fully connected layer.

In [680]:
```
### Define your architecture here.
### Feel free to use as many code cells as needed.


def LeNet(x):
    # Arguments used for tf.truncated_normal, randomly defines variab
les for the weights and biases for each layer
    mu = 0
    sigma = 0.1

    # SOLUTION: Layer 1: Convolutional. Input = 32x32x1. Output = 28x
28x6.
    #conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 6), me
an = mu, stddev = sigma))

    # SOLUTION: Layer 1: Convolutional. Input = 32x32x1. Output = 28x
28x32.
    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 32), me
an = mu, stddev = sigma))
    conv1_b = tf.Variable(tf.zeros(32), name="conv1_b")
    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.
    # SOLUTION: Pooling. Input = 28x28x6. Output = 14x14x32.
    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), m
ean = mu, stddev = sigma))

    # SOLUTION: Layer 2: Convolutional. Input = 14x14x32 Output = 10x
10x64.
    conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 32, 64), m
ean = mu, stddev = sigma))
    conv2_b = tf.Variable(tf.zeros(64), name = "conv2_b")
    conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padd
ing='VALID') + conv2_b

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

    # SOLUTION: Pooling. Input = 10x10x16. Output = 5x5x16.

    # SOLUTION: Pooling. Input = 10x10x64. Output = 5x5x64.
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2,
2, 1], padding='VALID')

    # SOLUTION: Flatten. Input = 5x5x16. Output = 400.

    # SOLUTION: Flatten. Input = 5x5x64. Output = 1600.
    fc0   = flatten(conv2)
    fc0 = tf.nn.dropout(fc0, keep_prob)


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

    # SOLUTION: Layer 3: Fully Connected. Input = 1600. Output = 120.
    fc1_W = tf.Variable(tf.truncated_normal(shape=(1600, 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)
    fc1 = tf.nn.dropout(fc1, keep_prob)

    # 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)
    fc2    = tf.nn.dropout(fc2, keep_prob)

    # SOLUTION: Layer 5: Fully Connected. Input = 84. Output = 10.
    fc3_W  = tf.Variable(tf.truncated_normal(shape=(84, 43), mean = m
u, stddev = sigma))
    fc3_b  = tf.Variable(tf.zeros(43))
    logits = tf.matmul(fc2, fc3_W) + fc3_b

    return logits
```

Features and Labels

Train LeNet to classify MNIST (http://yann.lecun.com/exdb/mnist/) data.

x is a placeholder for a batch of input images. y is a placeholder for a batch of output labels.

You do not need to modify this section.

In [681]:
```
keep_prob = tf.placeholder(tf.float32)
x = tf.placeholder(tf.float32, (None, 32, 32, 1))
y = tf.placeholder(tf.int32, (None))
one_hot_y = tf.one_hot(y, 43)
print(x.shape)
print(y.shape)
```

```
(?, 32, 32, 1)
<unknown>
```

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.

Training Pipeline

Create a training pipeline that uses the model to classify MNIST data.

You do not need to modify this section.

In [682]:
```
### 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.

rate = 0.00025
keep_probability = 0.8
logits = LeNet(x)
softmax = tf.nn.softmax(logits)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_ho
t_y, logits=logits)
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate = rate)
training_operation = optimizer.minimize(loss_operation)
prediction_operation = tf.argmax(logits, 1)
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot
_y, 1))
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.fl
oat32))
top5_operation = tf.nn.top_k(softmax, 5)

saver = tf.train.Saver()
```

Model Evaluation

Evaluate how well the loss and accuracy of the model for a given dataset.

You do not need to modify this section.

In [683]:
```
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[o
ffset:offset+BATCH_SIZE]
        accuracy = sess.run(accuracy_operation, feed_dict={x: batch_x
, y: batch_y, keep_prob: 1.0})
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples
```

Train the Model

Run the training data through the training pipeline to train the model.

Before each epoch, shuffle the training set.

After each epoch, measure the loss and accuracy of the validation set.

Save the model after training.

You do not need to modify this section.

In [684]:
```
EPOCHS = 30
BATCH_SIZE = 64

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:en
d]
            sess.run(training_operation, feed_dict={x: batch_x, y: ba
tch_y, keep_prob: keep_probability})

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

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

```
Training...

EPOCH 1 ...
Validation Accuracy = 0.848

EPOCH 2 ...
Validation Accuracy = 0.917

EPOCH 3 ...
Validation Accuracy = 0.940

EPOCH 4 ...
Validation Accuracy = 0.949

EPOCH 5 ...
Validation Accuracy = 0.957

EPOCH 6 ...
Validation Accuracy = 0.961

EPOCH 7 ...
Validation Accuracy = 0.962

EPOCH 8 ...
Validation Accuracy = 0.966

EPOCH 9 ...
Validation Accuracy = 0.969

EPOCH 10 ...
Validation Accuracy = 0.973

EPOCH 11 ...
Validation Accuracy = 0.968

EPOCH 12 ...
Validation Accuracy = 0.958

EPOCH 13 ...
Validation Accuracy = 0.970

EPOCH 14 ...
Validation Accuracy = 0.971

EPOCH 15 ...
Validation Accuracy = 0.972

EPOCH 16 ...
Validation Accuracy = 0.967

EPOCH 17 ...
Validation Accuracy = 0.969

EPOCH 18 ...
Validation Accuracy = 0.963

EPOCH 19 ...
```

```
Validation Accuracy = 0.975

EPOCH 20 ...
Validation Accuracy = 0.969

EPOCH 21 ...
Validation Accuracy = 0.965

EPOCH 22 ...
Validation Accuracy = 0.968

EPOCH 23 ...
Validation Accuracy = 0.975

EPOCH 24 ...
Validation Accuracy = 0.971

EPOCH 25 ...
Validation Accuracy = 0.977

EPOCH 26 ...
Validation Accuracy = 0.971

EPOCH 27 ...
Validation Accuracy = 0.969

EPOCH 28 ...
Validation Accuracy = 0.971

EPOCH 29 ...
Validation Accuracy = 0.970

EPOCH 30 ...
Validation Accuracy = 0.979

Model saved
```

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.

Load and Output the Images

In [685]:
```
import csv
print(n_classes)
```
```
43
```

In [686]:
```
traffic_sign_names = [None] * n_classes

with open('signnames.csv', 'r') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    for row in reader:
        try:
            traffic_sign_names[int(row[0])] = row[1]
        except ValueError:
            pass
```

In [687]:
```
print(len(traffic_sign_names))
print(traffic_sign_names)
```
```
43
['Speed limit (20km/h)', 'Speed limit (30km/h)', 'Speed limit (50km/
h)', 'Speed limit (60km/h)', 'Speed limit (70km/h)', 'Speed limit (80
km/h)', 'End of speed limit (80km/h)', 'Speed limit (100km/h)', 'Spee
d limit (120km/h)', 'No passing', 'No passing for vehicles over 3.5 m
etric tons', 'Right-of-way at the next intersection', 'Priority roa
d', 'Yield', 'Stop', 'No vehicles', 'Vehicles over 3.5 metric tons pr
ohibited', 'No entry', 'General caution', 'Dangerous curve to the lef
t', 'Dangerous curve to the right', 'Double curve', 'Bumpy road', 'Sl
ippery road', 'Road narrows on the right', 'Road work', 'Traffic sign
als', 'Pedestrians', 'Children crossing', 'Bicycles crossing', 'Bewar
e of ice/snow', 'Wild animals crossing', 'End of all speed and passin
g limits', 'Turn right ahead', 'Turn left ahead', 'Ahead only', 'Go s
traight or right', 'Go straight or left', 'Keep right', 'Keep left',
'Roundabout mandatory', 'End of no passing', 'End of no passing by ve
hicles over 3.5 metric tons']
```

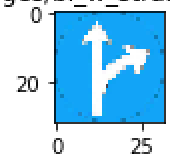
In [688]:
```
### Load the images and plot them here.
### Feel free to use as many code cells as needed.



new_test_images = glob.glob('new_test_images/*1.jpg')
print(new_test_images)
print(len(new_test_images))
```
```
['new_test_images/bl_w_straight_or_right1.jpg', 'new_test_images/stra
ight_or_or_right1.jpg', 'new_test_images/bl_w_straight_or_bright1.jp
g', 'new_test_images/do_not_enter1.jpg', 'new_test_images/no_left_tur
n1.jpg', 'new_test_images/straight_or_right1.jpg', 'new_test_images/n
o_u_turn1.jpg', 'new_test_images/straight_or_left1.jpg', 'new_test_im
ages/right_turn_only1.jpg', 'new_test_images/ss1.jpg', 'new_test_imag
es/stop1.jpg', 'new_test_images/straight_only1.jpg']
12
```

In [689]:
```
images_to_process = []

for image in new_test_images:
    img = mpimg.imread(image)
    print("Original image size: ", img.shape)
    resized_image = cv2.resize(img, (32, 32))
    print("Resized image size: ", resized_image.shape)
    images_to_process.append(resized_image)
```

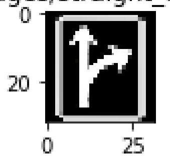
```
Original image size:  (185, 186, 3)
Resized image size:  (32, 32, 3)
Original image size:  (208, 208, 3)
Resized image size:  (32, 32, 3)
Original image size:  (167, 150, 3)
Resized image size:  (32, 32, 3)
Original image size:  (69, 68, 3)
Resized image size:  (32, 32, 3)
Original image size:  (71, 73, 3)
Resized image size:  (32, 32, 3)
Original image size:  (208, 208, 3)
Resized image size:  (32, 32, 3)
Original image size:  (77, 78, 3)
Resized image size:  (32, 32, 3)
Original image size:  (316, 316, 3)
Resized image size:  (32, 32, 3)
Original image size:  (93, 95, 3)
Resized image size:  (32, 32, 3)
Original image size:  (252, 255, 3)
Resized image size:  (32, 32, 3)
Original image size:  (84, 87, 3)
Resized image size:  (32, 32, 3)
Original image size:  (91, 91, 3)
Resized image size:  (32, 32, 3)
```

```
In [690]: for image, index in zip(images_to_process, range(len(images_to_proces
          s))):
              plt.figure(figsize=(1,1))
              plt.title(new_test_images[index])
              plt.imshow(image)
```

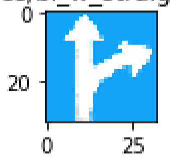
new_test_images/bl_w_straight_or_right1.jpg



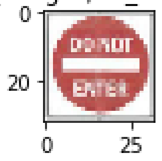
new_test_images/straight_or_or_right1.jpg



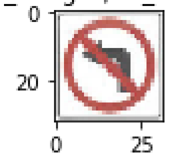
new_test_images/bl_w_straight_or_bright1.jpg



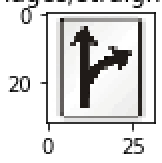
new_test_images/do_not_enter1.jpg



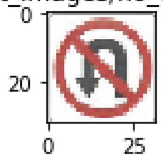
new_test_images/no_left_turn1.jpg



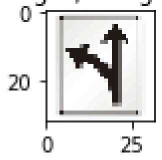
new_test_images/straight_or_right1.jpg



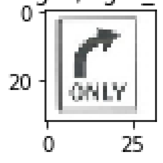
new_test_images/no_u_turn1.jpg



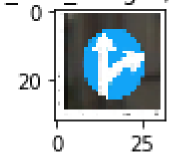
new_test_images/straight_or_left1.jpg



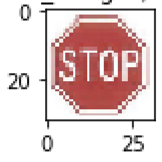
new_test_images/right_turn_only1.jpg



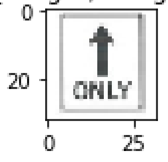
new_test_images/ss1.jpg



new_test_images/stop1.jpg



new_test_images/straight_only1.jpg



In [691]:
```
# Convert new images to gray scale and normalize
labels_for_new_test_images = np.array([36, 36, 36, 17, 26, 36, 26, 37
, 26, 36, 14, 26])

new_images_to_process = np.asarray(images_to_process)
images_to_process_gray = np.sum(new_images_to_process/3, axis=3, keep
dims=True)
images_to_process_norm = (images_to_process_gray - 128)/128


print(images_to_process_gray.shape)
```

```
(12, 32, 32, 1)
```

Predict the Sign Type for Each Image

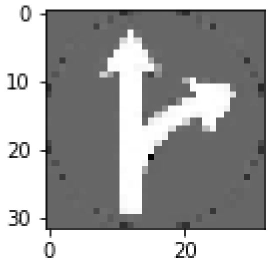
In [692]:
```
images = images_to_process_norm
```

In [693]:
```
print(images[0].shape)
a = np.array([image])
print(a.shape)
```

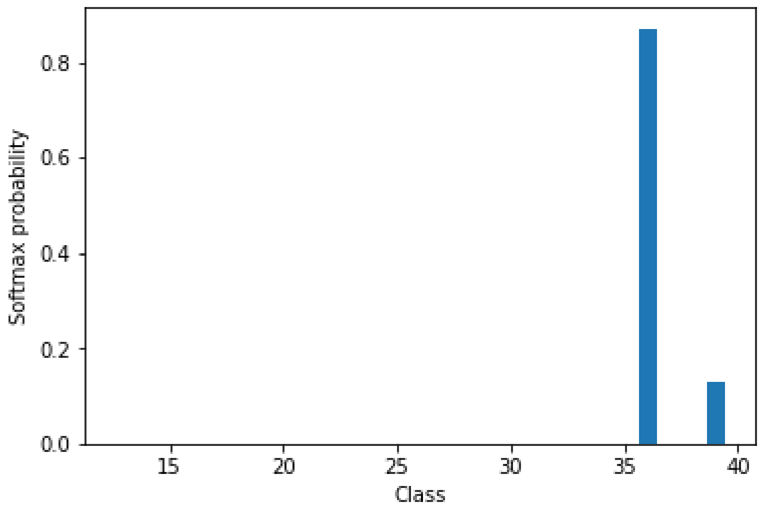
```
(32, 32, 1)
(1, 32, 32, 3)
```

```
In [694]: correct_predictions = 0
          print("Loading model...")
          with tf.Session() as sess:
              saver.restore(sess, tf.train.latest_checkpoint('.'))
              print("Model loaded.")

              for (image, cls) in zip(images, labels_for_new_test_images):
                  plt.figure(figsize=(2,2))
                  plt.imshow(image.squeeze(), cmap='gray')
                  plt.show()
                  '''image = process_input_image(image)
                  plt.figure(figsize=(2,2))
                  plt.imshow(image.squeeze(), cmap='gray', vmin=-1, vmax=1)
                  plt.show()'''
                  #predictedName = predict_name(image)
                  image = np.expand_dims(image, axis=0)
                  #image = np.expand_dims(image, axis=3)
                  print('image shape:', image.shape)
                  a = np.array([image])
                  print(a.shape)
                  prediction = sess.run(prediction_operation, feed_dict={x: ima
          ge, keep_prob: 1.0})

                  print('prediction',prediction)
                  print('traffic_sign_name', traffic_sign_names[prediction[0]])
                  print('label', traffic_sign_names[cls])
                  if traffic_sign_names[prediction[0]] == traffic_sign_names[cl
          s]:
                      print('Predicted: {0} (CORRECT)'.format(traffic_sign_name
          s[prediction[0]]))
                      correct_predictions += 1
                  else:
                      print('Predicted: {0} (INCORRECT, expected: {1})'.format(
          traffic_sign_names[prediction[0]], traffic_sign_names[cls]))

                  top5 = sess.run(top5_operation, feed_dict={x: image, keep_pro
          b: 1.0})
                  plt.bar(top5.indices[0], top5.values[0])
                  plt.xlabel("Class")
                  plt.ylabel('Softmax probability')
                  plt.show()
                  for (v,i) in zip(top5.values[0], top5.indices[0]):
                      print("{0:.4f} {1}".format(v, class_name[i]))

          print('{0} out of {1} web images are predicted correctly'.format(corr
          ect_predictions, len(images)))
```

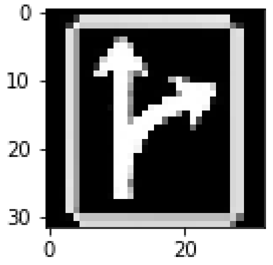
```
Loading model...
INFO:tensorflow:Restoring parameters from ./lenet
Model loaded.
```



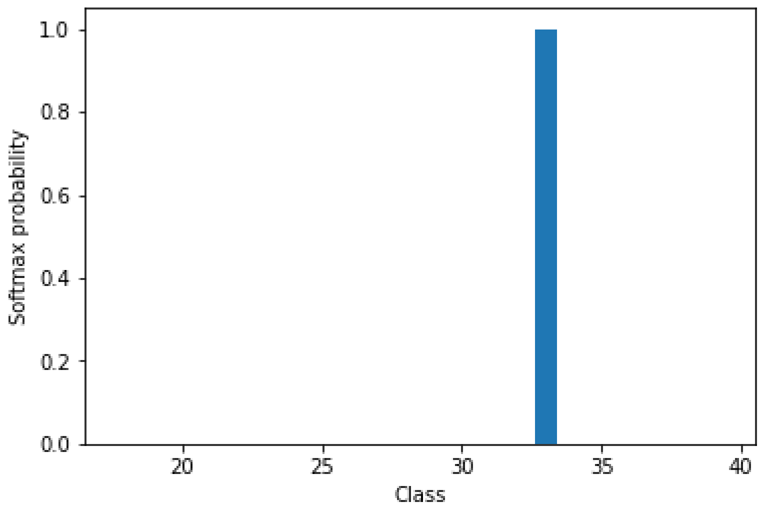
```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [36]
traffic_sign_name Go straight or right
label Go straight or right
Predicted: Go straight or right (CORRECT)
```



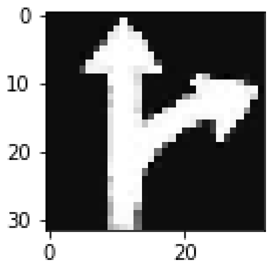
```
0.8708 Go straight or right
0.1275 Keep left
0.0008 Yield
0.0006 Turn right ahead
0.0003 Road work
```



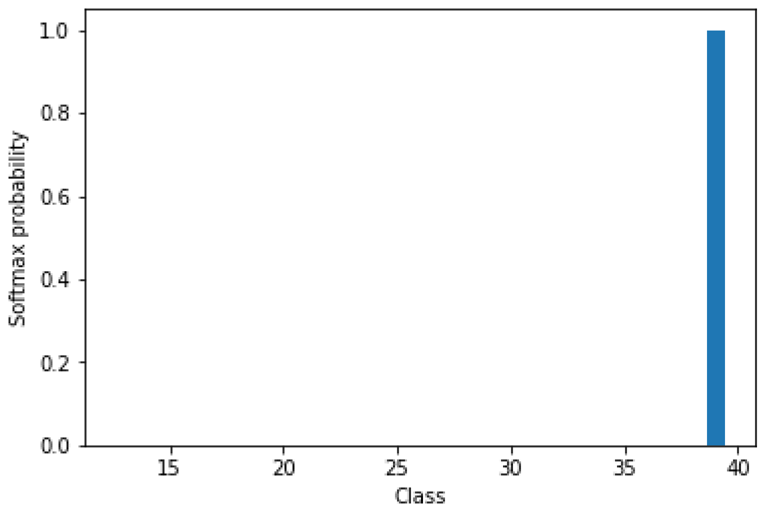
```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [33]
traffic_sign_name Turn right ahead
label Go straight or right
Predicted: Turn right ahead (INCORRECT, expected: Go straight or righ
t)
```



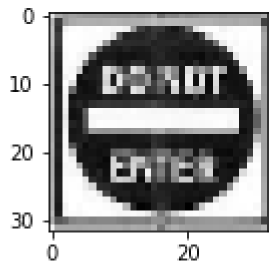
```
1.0000 Turn right ahead
0.0000 Keep left
0.0000 Road work
0.0000 General caution
0.0000 Pedestrians
```



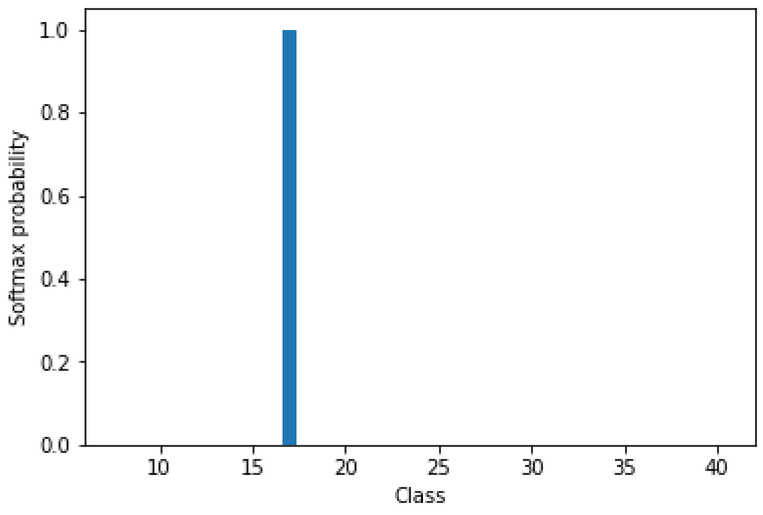
```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [39]
traffic_sign_name Keep left
label Go straight or right
Predicted: Keep left (INCORRECT, expected: Go straight or right)
```



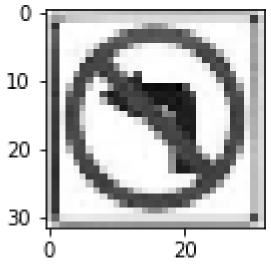
```
1.0000 Keep left
0.0000 Yield
0.0000 Road work
0.0000 Turn right ahead
0.0000 Go straight or right
```



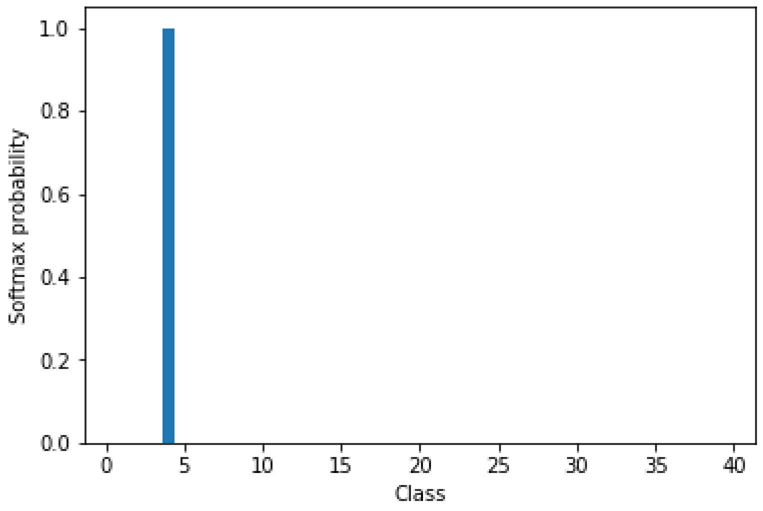
```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [17]
traffic_sign_name No entry
label No entry
Predicted: No entry (CORRECT)
```



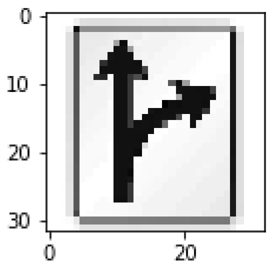
```
1.0000 No entry
0.0000 Speed limit (120km/h)
0.0000 Roundabout mandatory
0.0000 No passing
0.0000 Go straight or left
```



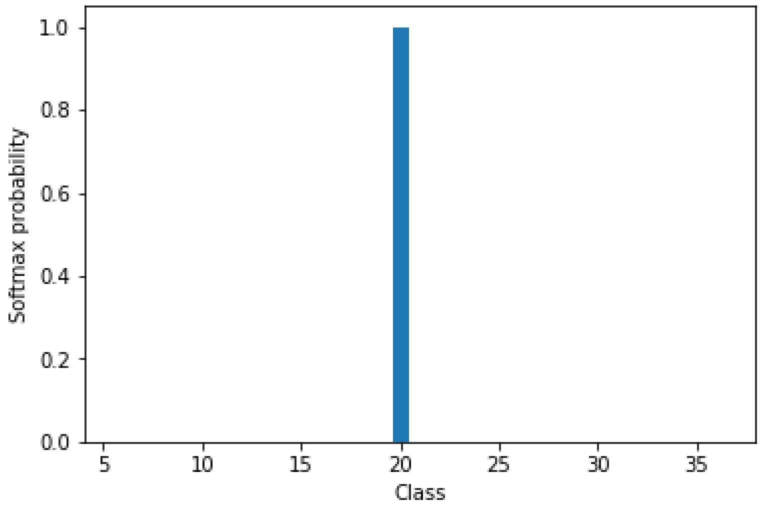
```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [4]
traffic_sign_name Speed limit (70km/h)
label Traffic signals
Predicted: Speed limit (70km/h) (INCORRECT, expected: Traffic signal
s)
```



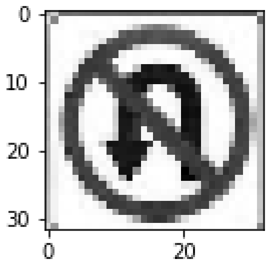
```
1.0000 Speed limit (70km/h)
0.0000 Speed limit (100km/h)
0.0000 Speed limit (30km/h)
0.0000 Speed limit (50km/h)
0.0000 Keep left
```



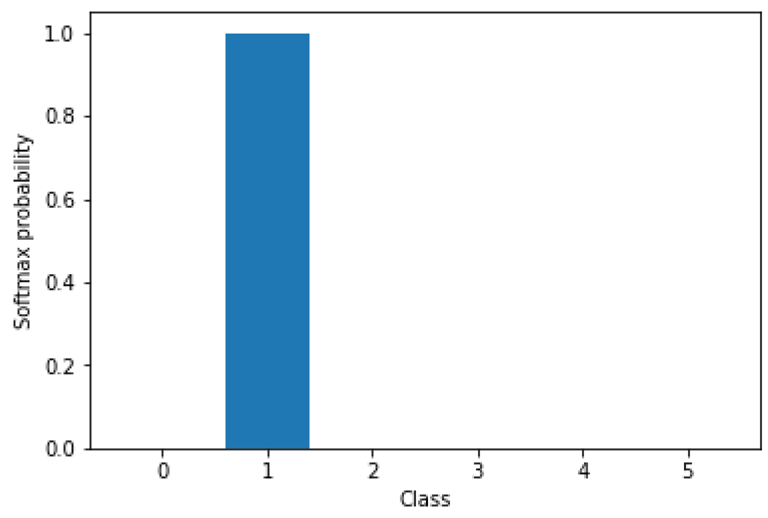
```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [20]
traffic_sign_name Dangerous curve to the right
label Go straight or right
Predicted: Dangerous curve to the right (INCORRECT, expected: Go stra
ight or right)
```



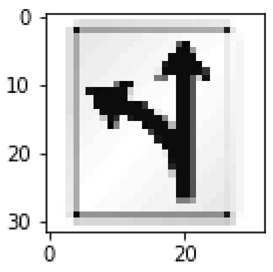
```
1.0000 Dangerous curve to the right
0.0000 Go straight or right
0.0000 End of speed limit (80km/h)
0.0000 Children crossing
0.0000 End of all speed and passing limits
```



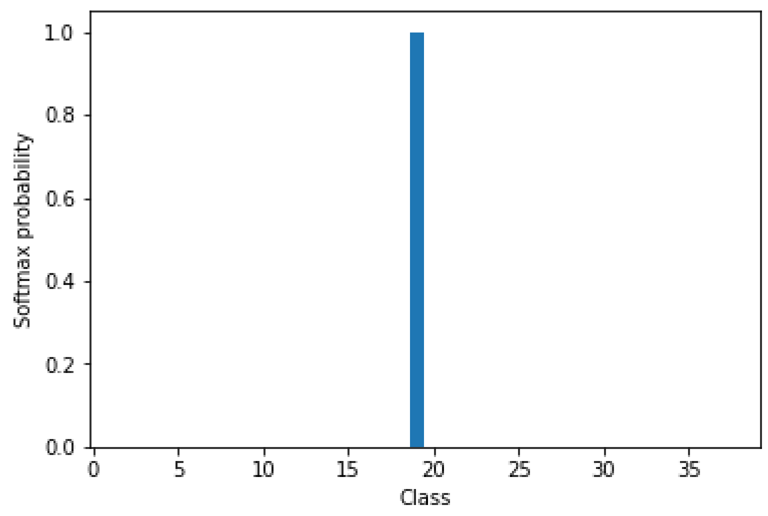
```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [1]
traffic_sign_name Speed limit (30km/h)
label Traffic signals
Predicted: Speed limit (30km/h) (INCORRECT, expected: Traffic signal
s)
```



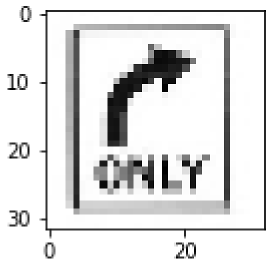
```
0.9998 Speed limit (30km/h)
0.0002 Speed limit (20km/h)
0.0000 Speed limit (50km/h)
0.0000 Speed limit (70km/h)
0.0000 Speed limit (80km/h)
```



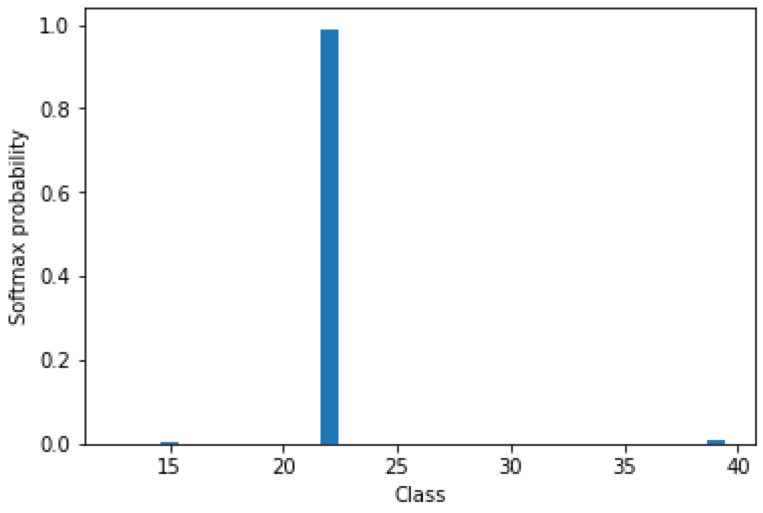
```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [19]
traffic_sign_name Dangerous curve to the left
label Go straight or left
Predicted: Dangerous curve to the left (INCORRECT, expected: Go strai
ght or left)
```



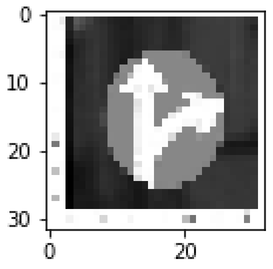
```
1.0000 Dangerous curve to the left
0.0000 Double curve
0.0000 Go straight or left
0.0000 Speed limit (50km/h)
0.0000 Traffic signals
```



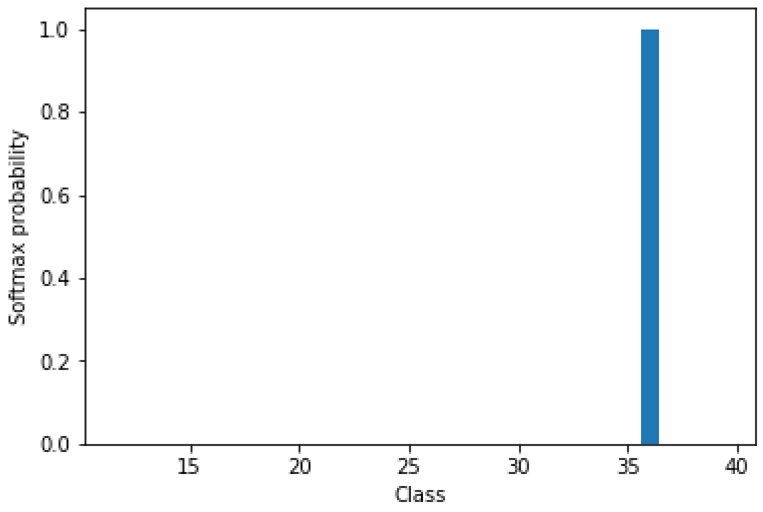
```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [22]
traffic_sign_name Bumpy road
label Traffic signals
Predicted: Bumpy road (INCORRECT, expected: Traffic signals)
```



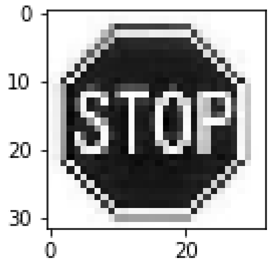
```
0.9898 Bumpy road
0.0076 Keep left
0.0025 No vehicles
0.0000 Yield
0.0000 Children crossing
```



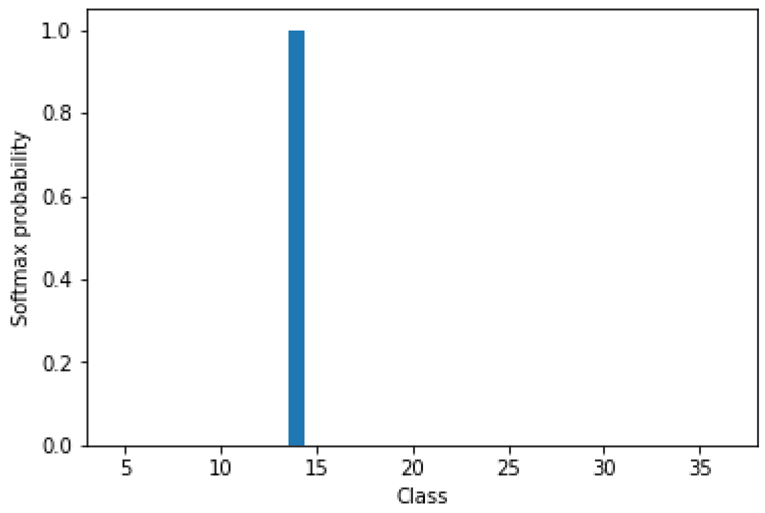
```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [36]
traffic_sign_name Go straight or right
label Go straight or right
Predicted: Go straight or right (CORRECT)
```



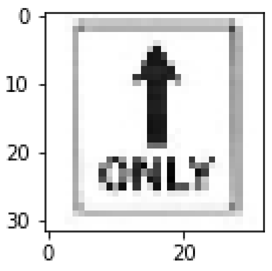
```
0.9999 Go straight or right
0.0000 Keep left
0.0000 Road work
0.0000 No vehicles
0.0000 Priority road
```



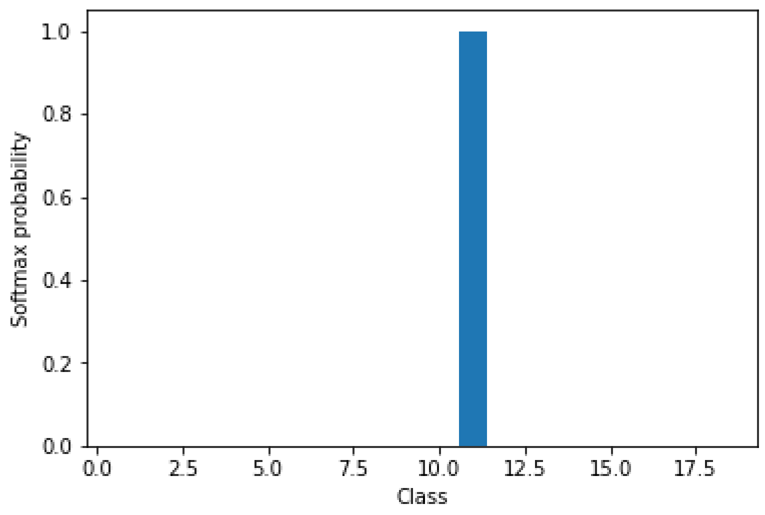
```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [14]
traffic_sign_name Stop
label Stop
Predicted: Stop (CORRECT)
```



```
1.0000 Stop
0.0000 Go straight or right
0.0000 Speed limit (80km/h)
0.0000 Turn right ahead
0.0000 No entry
```



```
image shape: (1, 32, 32, 1)
(1, 1, 32, 32, 1)
prediction [11]
traffic_sign_name Right-of-way at the next intersection
label Traffic signals
Predicted: Right-of-way at the next intersection (INCORRECT, expecte
d: Traffic signals)
```



```
0.9997 Right-of-way at the next intersection
0.0001 General caution
0.0001 Speed limit (60km/h)
0.0001 Speed limit (30km/h)
0.0000 Yield
4 out of 12 web images are predicted correctly
```

Analyze Performance

In [695]:
```
### Calculate the accuracy for these 5 new images.
### For example, if the model predicted 1 out of 5 signs correctly, i
t's 20% accurate on these new images.
```

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.

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,  0.078934
97,
         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 [696]:
```
### Print out the top five softmax probabilities for the predictions
 on the German traffic sign images found on the web.
### Feel free to use as many code cells as needed.
```

```
In [697]: with tf.Session() as sess:
              saver.restore(sess, tf.train.latest_checkpoint('.'))

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

```
INFO:tensorflow:Restoring parameters from ./lenet
Test Accuracy = 0.965
```

```
In [698]: with tf.Session() as sess:
              saver.restore(sess, "./lenet")

              train_accuracy = evaluate(X_train, y_train)
              print("Training Accuracy = {:.3f}".format(train_accuracy))

              valid_accuracy = evaluate(X_validation, y_validation)
              print("Validation Accuracy = {:.3f}".format(valid_accuracy))

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

```
INFO:tensorflow:Restoring parameters from ./lenet
Training Accuracy = 1.000
Validation Accuracy = 0.979
Test Accuracy = 0.965
```

Project Writeup

Once you have completed the code implementation, document your results in a project writeup using this template (https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project/blob/master/writeup_template.md) 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 \n", "**File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

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 (https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/6df7ae49-c61c-4bb2-a23e-6527e69209ec/lessons/601ae704-1035-4287-8b11-e2c2716217ad/concepts/d4aca031-508f-4e0b-b493-e7b706120f81) 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 (https://devblogs.nvidia.com/parallelforall/deep-learning-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.

Combined Image

Your output should look something like this (above)

In [699]:
```
### 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 t
he feature maps
# tf_activation: should be a tf variable name used during your traini
ng procedure that represents the calculated state of a specific weigh
t layer
# activation_min/max: can be used to view the activation contrast in
 more detail, by default matplot sets min and max to the actual min a
nd max values of the output
# plt_num: used to plot out multiple different weight feature map set
s on the same block, just extend the plt number for each new feature
 map entry

def outputFeatureMap(image_input, tf_activation, activation_min=-1, a
ctivation_max=-1 ,plt_num=1):
    # Here make sure to preprocess your image_input in a way your net
work expects
    # with size, normalization, ect if needed
    # image_input =
    # Note: x should be the same name as your network's tensorflow da
ta placeholder variable
    # If you get an error tf_activation is not defined it may be havi
ng trouble accessing the variable from inside a function
    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 m
aps to show on each row and column
        plt.title('FeatureMap ' + str(featuremap)) # displays the fea
ture map number
        if activation_min != -1 & activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation=
"nearest", vmin =activation_min, vmax=activation_max, cmap="gray")
        elif activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation=
"nearest", vmax=activation_max, cmap="gray")
        elif activation_min !=-1:
            plt.imshow(activation[0,:,:, featuremap], interpolation=
"nearest", vmin=activation_min, cmap="gray")
        else:
            plt.imshow(activation[0,:,:, featuremap], interpolation=
"nearest", cmap="gray")
```