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 [1]:

```
# Load pickled data
import pickle

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

training_file = "/home/carnd/CarND-Traffic-Sign-Classifier-Project/tra
ffic_signs_data/train.p"
validation_file= "/home/carnd/CarND-Traffic-Sign-Classifier-Project/tr
affic_signs_data/valid.p"
testing_file = "/home/carnd/CarND-Traffic-Sign-Classifier-Project/traf
fic_signs_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)

X_train, y_train = train['features'], train['labels']
X_valid, y_valid = valid['features'], valid['labels']
X_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 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 re
sults
import numpy as np
# TODO: Number of training examples
n_train = y_train.shape[0]

# TODO: Number of validation examples
n_validation = y_valid.shape[0]

# TODO: Number of testing examples.
n_test = y_test.shape[0]

# TODO: What's the shape of an traffic sign image?
image_shape = X_train.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 = (34799, 32, 32, 3)
Number of classes = 43
```

Include an exploratory visualization of the dataset

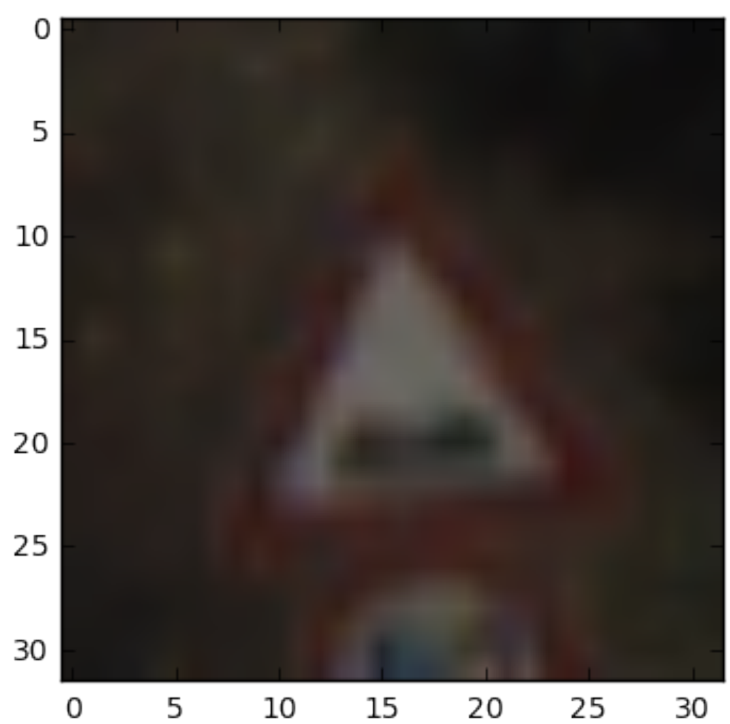
Visualize the German Traffic Signs Dataset using the pickled file(s). This is open ended, suggestions include: plotting traffic sign images, plotting the count of each sign, etc.

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.

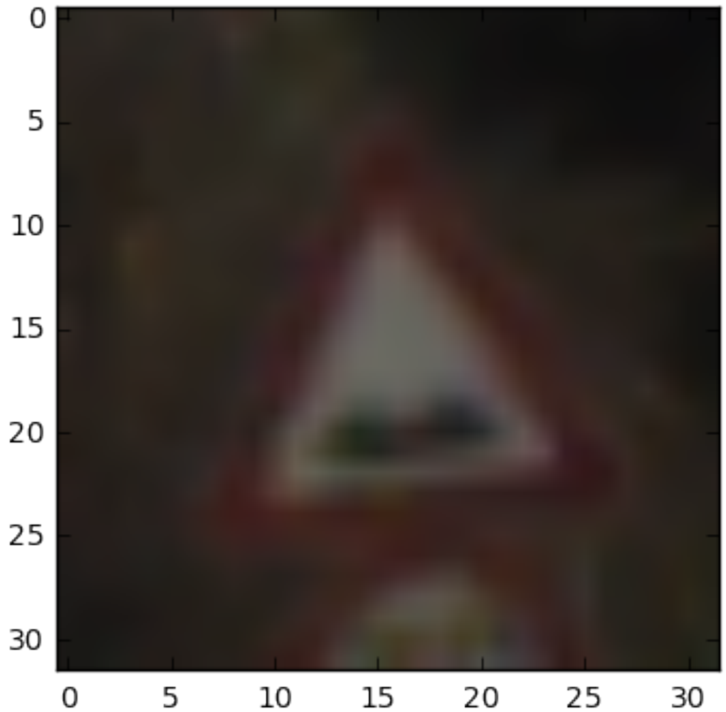
NOTE: It's recommended you start with something simple first. If you wish to do more, come back to it after you've completed the rest of the sections. It can be interesting to look at the distribution of classes in the training, validation and test set. Is the distribution the same? Are there more examples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

In [3]:
```
### Data exploration visualization code goes here.
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
# Visualizations will be shown in the notebook.
%matplotlib inline

img_indx = [9, 22] # put in any imag class we want to see.
def find_imgs(img_indx, num_of_imgs = 1):
    imgs_printed = {}
    for i in range(len(y_train)):
        for idx in img_indx:
            if idx == y_train[i]:
                if idx not in imgs_printed:
                    imgs_printed[idx] = 0
                imgs_printed[idx] += 1
                if imgs_printed[idx] <= 5:
                    print('img ', i)
                    plt.imshow(X_train[i])
                    plt.show()
#         print(imgs_printed['idx'])

find_imgs(img_indx)
```

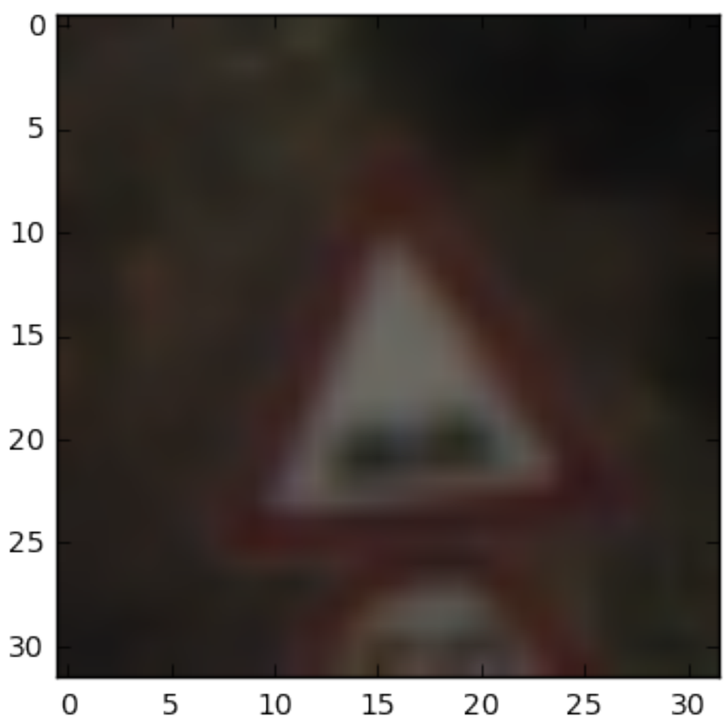
```
img  4500
```




```
img  4501
```

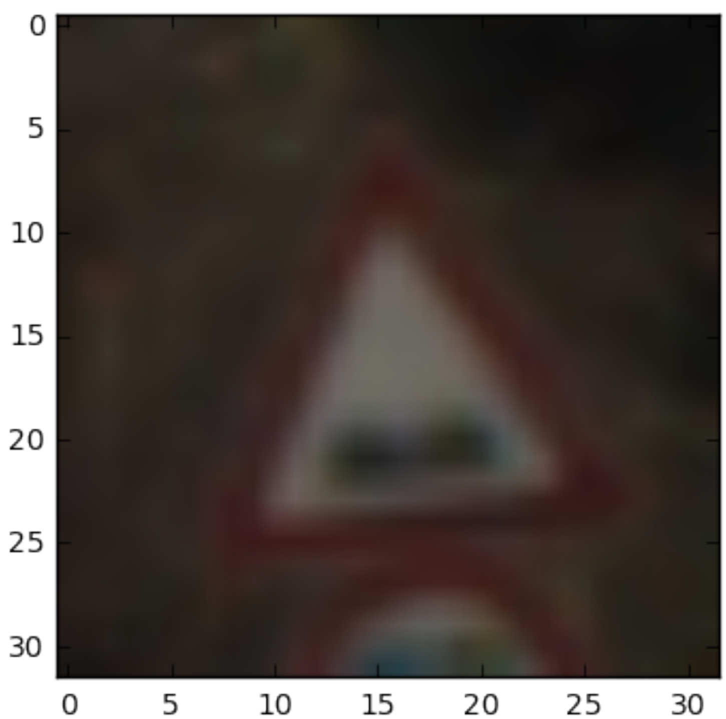


img 4502

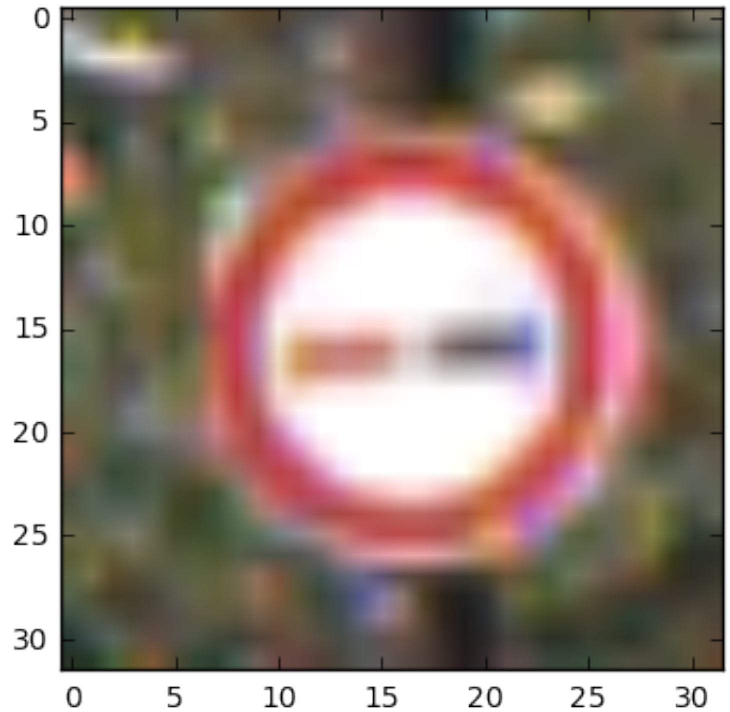


img 4503

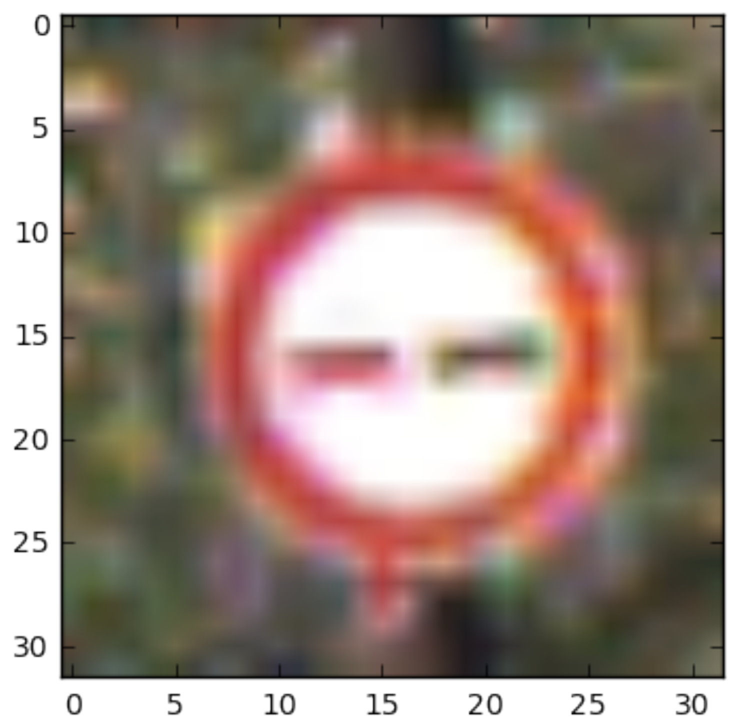
img 4504



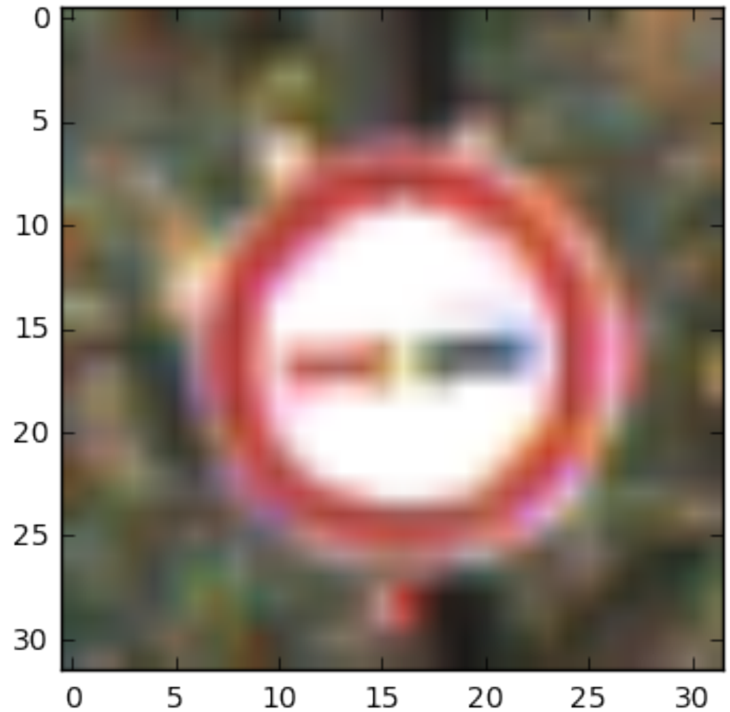
img 11040



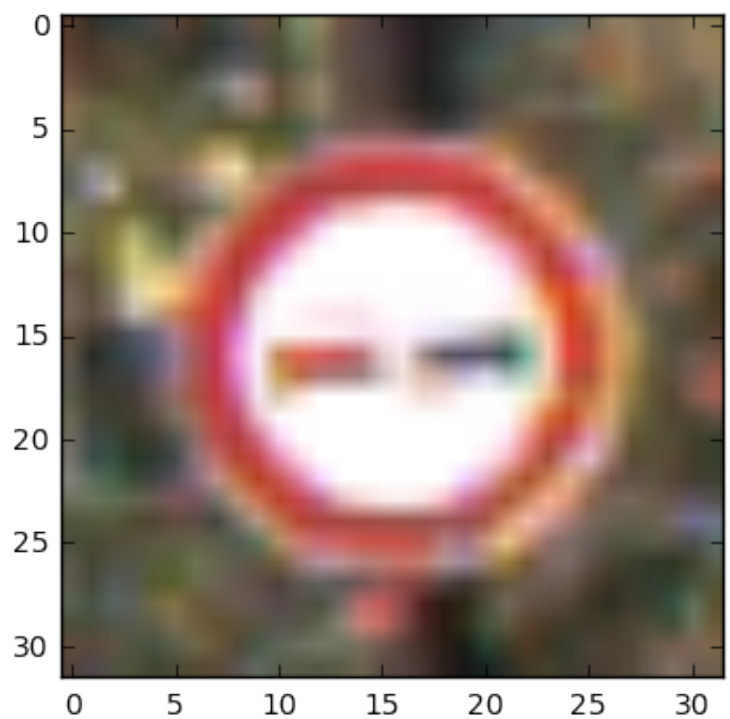
img 11041



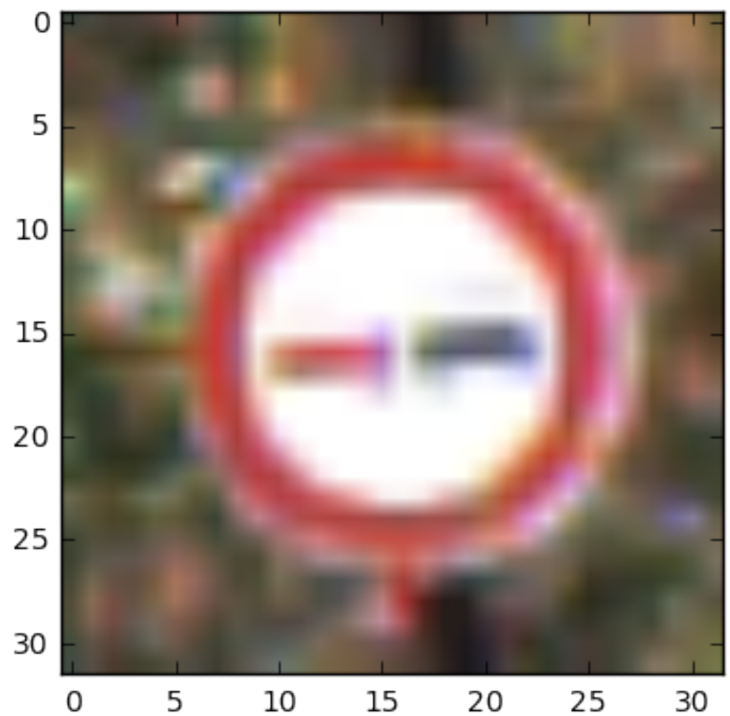
img 11042



img 11043



img 11044



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.

In [4]:
```
### Preprocess the data here. It is required to normalize the data. Ot
her preprocessing steps could include
### converting to grayscale, etc.
### Feel free to use as many code cells as needed.
def normalize_image(image_set):
    return (image_set-128)/128

print('before norm.')
print('xtrain', X_train[0][0][0])
print('xtest', X_test[0][0][0])
print('xvalid', X_valid[0][0][0])

X_train = normalize_image(X_train.astype(np.int16))
X_test = normalize_image(X_test.astype(np.int16))
X_valid = normalize_image(X_valid.astype(np.int16))

print('after norm.')
print('xtrain', X_train.shape)
print('xtest', X_test[0][0][0])
print('xvalid', X_valid[0][0][0])
```

```
before norm.
xtrain [28 25 24]
xtest [116 139 174]
xvalid [13 12 12]
after norm.
xtrain (34799, 32, 32, 3)
xtest [-0.09375    0.0859375  0.359375 ]
xvalid [-0.8984375 -0.90625   -0.90625  ]
```

Model Architecture

In [5]:
```
### Define your architecture here.
### Feel free to use as many code cells as needed.
import tensorflow as tf
from tensorflow.contrib.layers import flatten

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

    # SOLUTION: Layer 1: Convolutional.
    conv1_in_depth = 3
    conv1_out_depth = 50
    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 3, conv1_ou
```

```
t_depth), mean = mu, stddev = sigma))
    conv1_b = tf.Variable(tf.zeros(conv1_out_depth))
    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.
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2
, 1], padding='VALID')

    # SOLUTION: Layer 2: Convolutional.
    conv2_out_depth = 40
    conv2_fmp_size = 5
    conv2_W = tf.Variable(tf.truncated_normal(shape=(conv2_fmp_size, c
onv2_fmp_size, conv1_out_depth, conv2_out_depth), mean = mu, stddev = 
sigma))
    conv2_b = tf.Variable(tf.zeros(conv2_out_depth))
    conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], paddi
ng='VALID') + conv2_b

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

    # SOLUTION: Pooling.
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2
, 1], padding='VALID')

    # SOLUTION: Flatten.
    fc0   = flatten(conv2)

    # SOLUTION: Layer 3: Fully Connected.
    fc1_in = conv2_fmp_size**2*conv2_out_depth
    fc1_out = int(fc1_in/1.5)
    fc1_W = tf.Variable(tf.truncated_normal(shape=(fc1_in, fc1_out), m
ean = mu, stddev = sigma))
    fc1_b = tf.Variable(tf.zeros(fc1_out))
    fc1   = tf.matmul(fc0, fc1_W) + fc1_b

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

    # SOLUTION: Layer 4: Fully Connected.
    fc2_out = int(fc1_out/2)
    fc2_W  = tf.Variable(tf.truncated_normal(shape=(fc1_out, fc2_out),
mean = mu, stddev = sigma))
    fc2_b  = tf.Variable(tf.zeros(fc2_out))
    fc2    = tf.matmul(fc1, fc2_W) + fc2_b
```

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

    # SOLUTION: Layer 5: Fully Connected.
    fc3_W  = tf.Variable(tf.truncated_normal(shape=(fc2_out, n_classes
), mean = mu, stddev = sigma))
    fc3_b  = tf.Variable(tf.zeros(n_classes))
    logits = tf.matmul(fc2, fc3_W) + fc3_b

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

In [6]:
```
### Train your model here.
### Calculate and report the accuracy on the training and validation s
et.
### 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.
from sklearn.utils import shuffle
data_shape = X_train.shape
x = tf.placeholder(tf.float32, (None, data_shape[1], data_shape[2], da
ta_shape[3]))
y = tf.placeholder(tf.int32, (None))
one_hot_y = tf.one_hot(y, n_classes)

BATCH_SIZE = 128
EPOCHS = 20

# training pipeline
learning_rate = 0.001
logits = LeNet(x)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot
_y, logits=logits)
loss = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# evaluate

correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_
```

```
y, 1))
accuracy_op = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

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_dat
a[offset:offset+BATCH_SIZE]
            accuracy = sess.run(accuracy_op, feed_dict={x:batch_x, y:b
atch_y})
            total_accuracy += accuracy * len(batch_x)
    return total_accuracy/num_examples



# train the model
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):
            batch_x, batch_y = X_train[offset:offset+BATCH_SIZE], y_tr
ain[offset:offset+BATCH_SIZE]
            sess.run(training_op, feed_dict={x:batch_x, y:batch_y})

        validation_accuracy = evaluate(X_valid, y_valid)
        print("Epoch {} : ".format(i+1))
        print("Validation accuracy = {:3f}".format(validation_accuracy
))
        print()

# evaluate the model
    sess = tf.get_default_session()
    test_accuracy = evaluate(X_test, y_test)
    print("Test Accuracy = {:3f}".format(test_accuracy))
```

```
Training.

Epoch 1 :
Validation accuracy = 0.861905

Epoch 2 :
Validation accuracy = 0.899773

Epoch 3 :
```

```
Validation accuracy = 0.899546

Epoch 4 :
Validation accuracy = 0.920181

Epoch 5 :
Validation accuracy = 0.924490

Epoch 6 :
Validation accuracy = 0.914512

Epoch 7 :
Validation accuracy = 0.929025

Epoch 8 :
Validation accuracy = 0.895238

Epoch 9 :
Validation accuracy = 0.928571

Epoch 10 :
Validation accuracy = 0.933787

Epoch 11 :
Validation accuracy = 0.931293

Epoch 12 :
Validation accuracy = 0.929252

Epoch 13 :
Validation accuracy = 0.937868

Epoch 14 :
Validation accuracy = 0.954875

Epoch 15 :
Validation accuracy = 0.940590

Epoch 16 :
Validation accuracy = 0.945125

Epoch 17 :
Validation accuracy = 0.955329

Epoch 18 :
Validation accuracy = 0.948980

Epoch 19 :
Validation accuracy = 0.936054
```

```
Epoch 20 :
Validation accuracy = 0.943537

Test Accuracy = 0.941568
```

Step 3: Test a Model on New Images

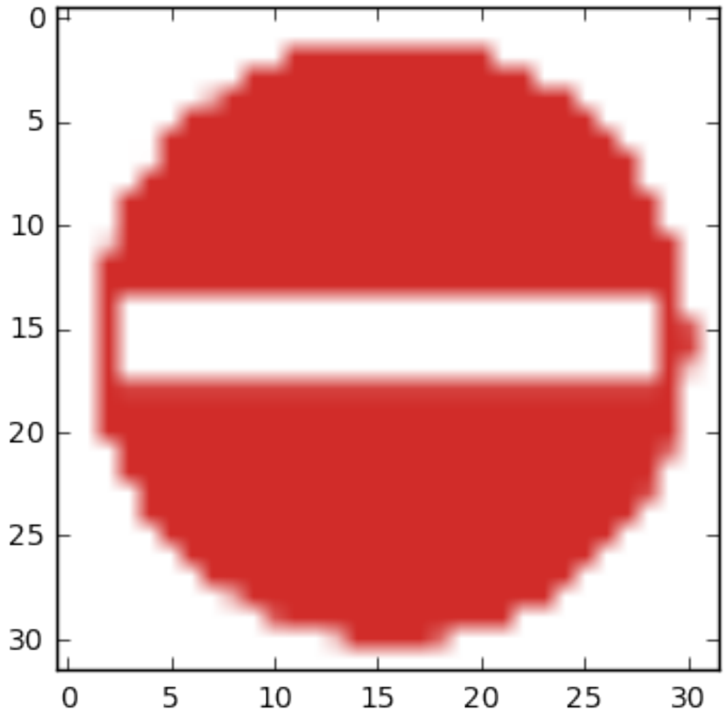
To give yourself more insight into how your model is working, download at least five pictures of German traffic signs from the web and use your model to predict the traffic sign type.

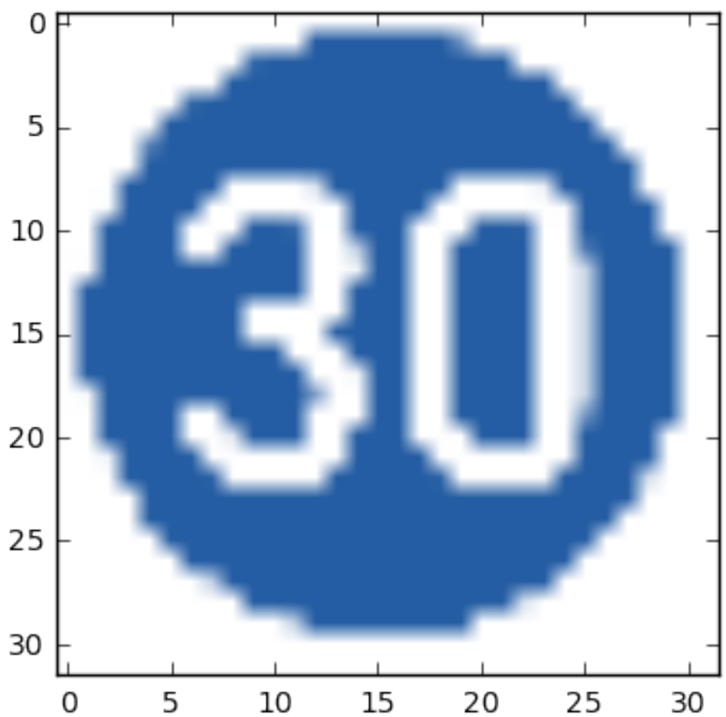
You may find `signnames.csv` useful as it contains mappings from the class id (integer) to the actual sign name.

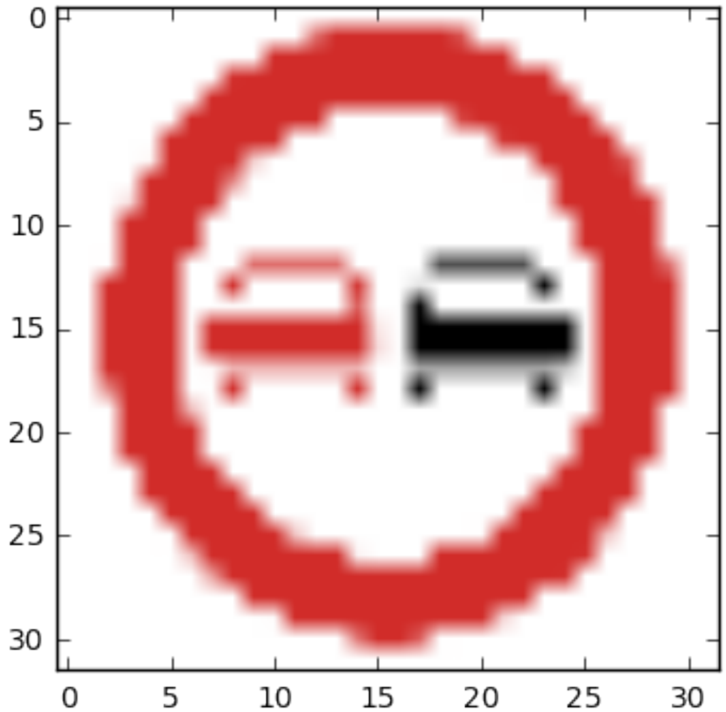
Load and Output the Images

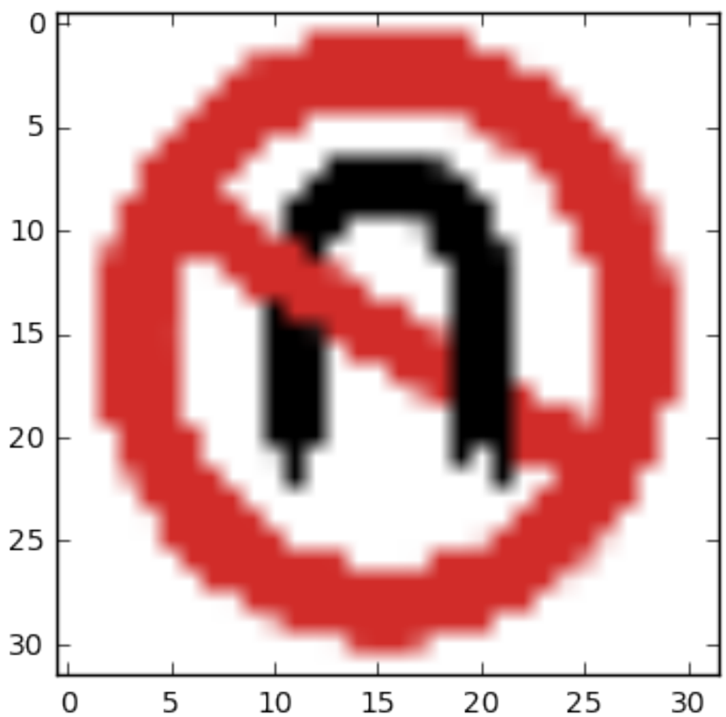
In [7]:

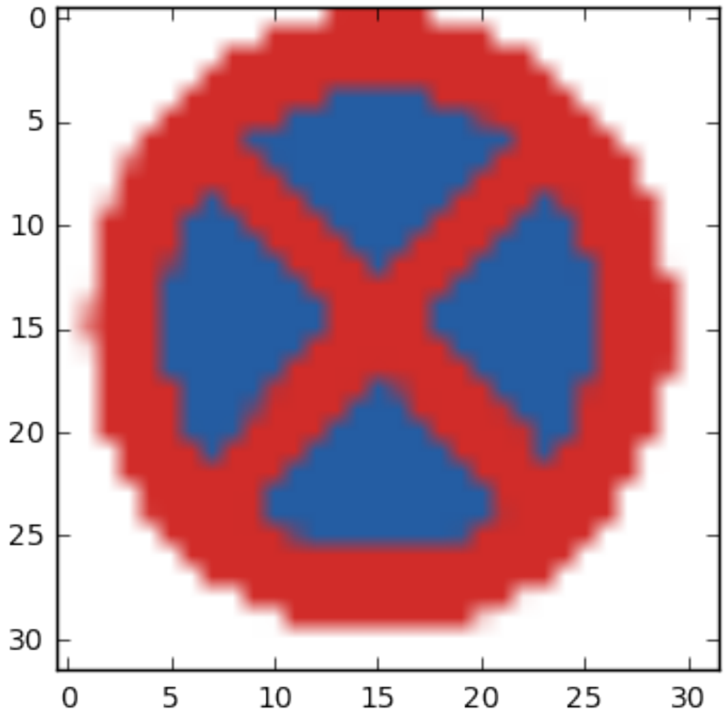
```
### Load the images and plot them here.
### Feel free to use as many code cells as needed.
from os import listdir
from os.path import isfile, join
from PIL import Image

mypath = "/home/carnd/CarND-Traffic-Sign-Classifier-Project/traffic_si
gns_data/traffic_signs_on_the_web"
img_files = [join(mypath,f) for f in listdir(mypath) if isfile(join(my
path, f))]
%matplotlib inline
web_imgs = []
for file in img_files:
    img = Image.open(file).convert('RGB')
    web_imgs.append(img.resize((data_shape[1], data_shape[2]), Image.N
EAREST))

for img in web_imgs:
    plt.imshow(img)
    plt.show()

web_imgs = np.asarray([np.asarray(web_imgs[i]) for i in range(len(web_
imgs))])
normalize_image(web_imgs)
```


0
5
10
15
20
25
30
0 5 10 15 20 25 30


30
0
5
10
15
20
25
30
0 5 10 15 20 25 30


0
5
10
15
20
25
30
0 5 10 15 20 25 30


0
5
10
15
20
25
30
0 5 10 15 20 25 30




```
Out[7]: array([[[[0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875],
                 ...,
                 [0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875]],

                [[0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875],
                 ...,
                 [0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875]],

                [[0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875],
                 ...,
                 [0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875]],

                ...,

                [[0.9921875, 0.9921875, 0.9921875],
                 [0.9921875, 0.9921875, 0.9921875],
```

```
         [0.9921875, 0.9921875, 0.9921875],
         ...,
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875]],

        [[0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         ...,
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875]],

        [[0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         ...,
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875]]],


       [[[0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         ...,
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875]],

        [[0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         ...,
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875]],

        [[0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         ...,
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875],
         [0.9921875, 0.9921875, 0.9921875]],

        ...,

        [[0.9921875, 0.9921875, 0.9921875],
```

```
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]]],


[[[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 ...,
```

```
 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]]],


[[[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 ...,
```

```
 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]]],


[[[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],

 [[0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  ...,
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875],
  [0.9921875, 0.9921875, 0.9921875]],
```

```
         ...,

         [[0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875],
          ...,
          [0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875]],

         [[0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875],
          ...,
          [0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875]],

         [[0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875],
          ...,
          [0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875],
          [0.9921875, 0.9921875, 0.9921875]]]])
```

Predict the Sign Type for Each Image

In [8]:

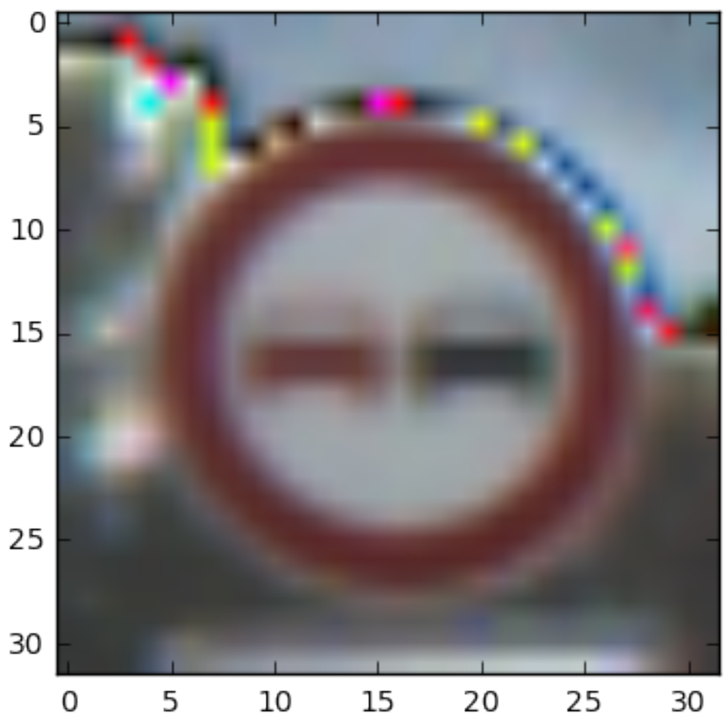
```
### Run the predictions here and use the model to output the predictio
n for each image.
### Make sure to pre-process the images with the same pre-processing p
ipeline used earlier.
### Feel free to use as many code cells as needed.
print(web_imgs.shape)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print(sess)
    output = tf.argmax(logits, 1).eval(feed_dict={x:web_imgs})
    print(output)
```

```
(5, 32, 32, 3)
<tensorflow.python.client.session.Session object at 0x7fb58e9b4978>
[22 22  9  9 22]
```

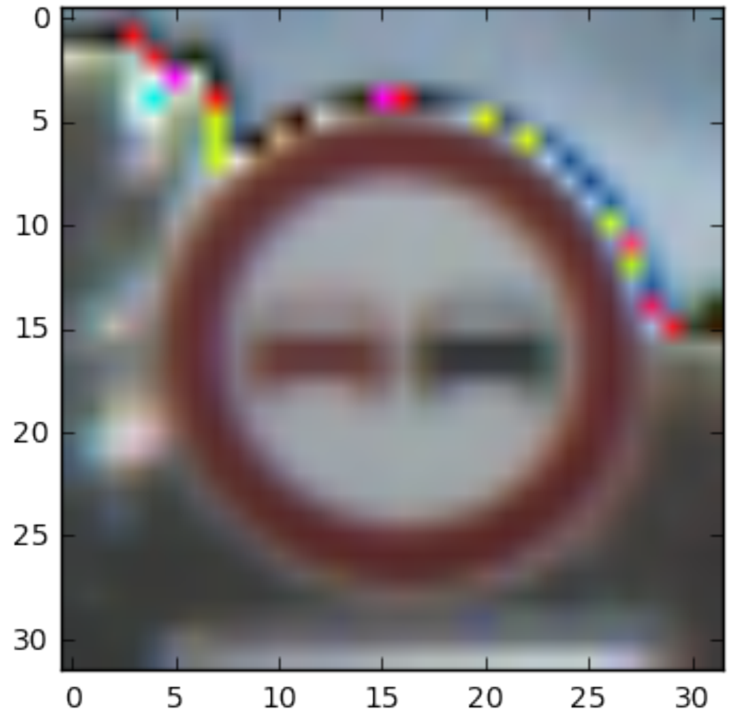
Analyze Performance

In [9]:
```
### 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 these new images.
find_imgs(output)
print("The model predicted 1 out of 5 signs correctly.")
```

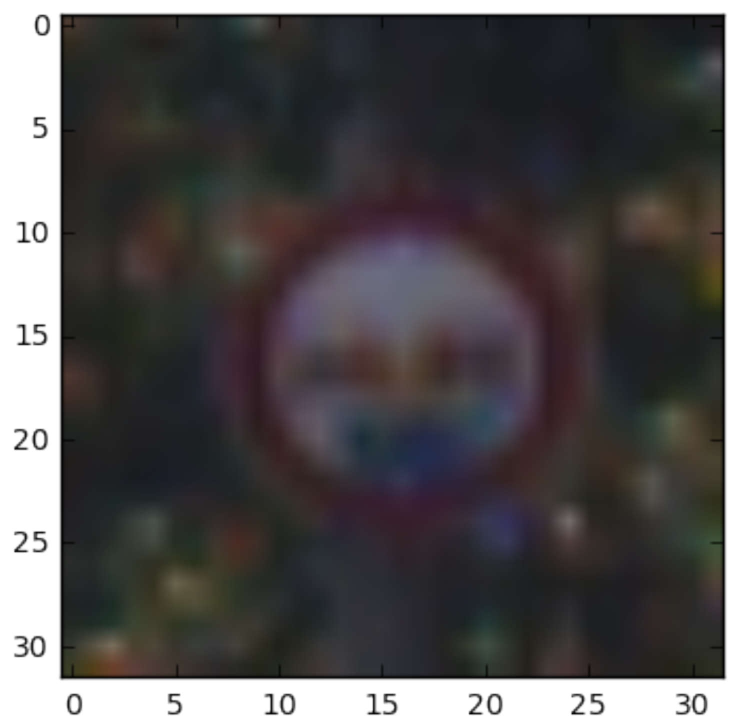
```
img  6
```



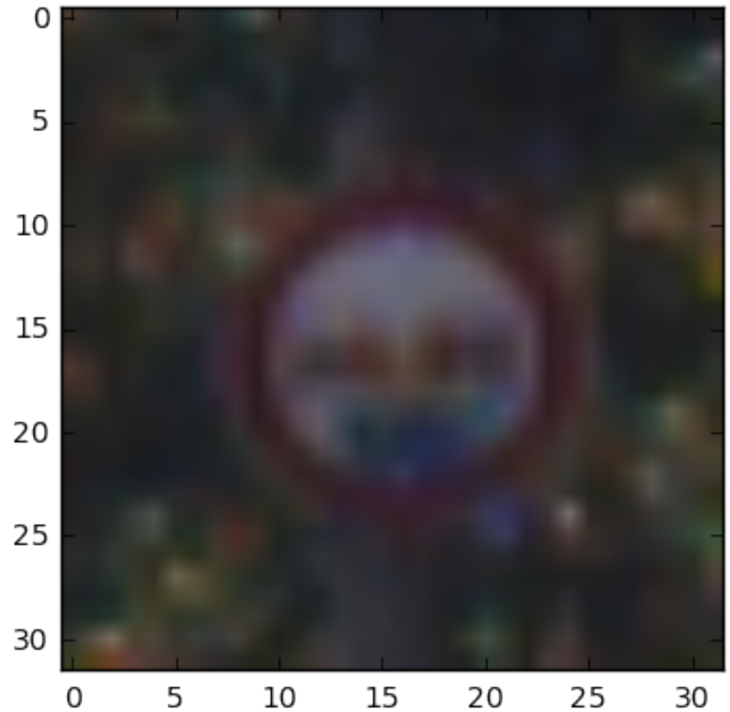

```
img  6
```



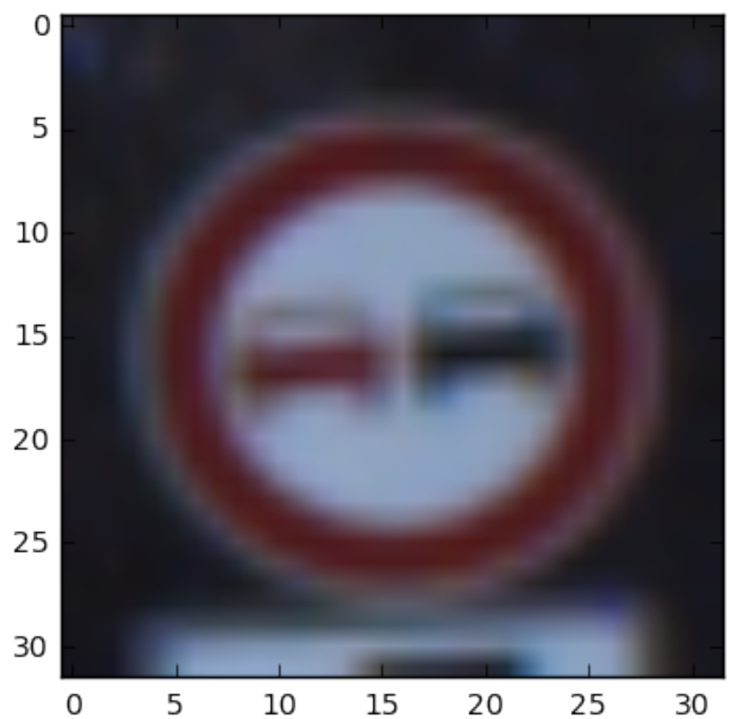
img 34



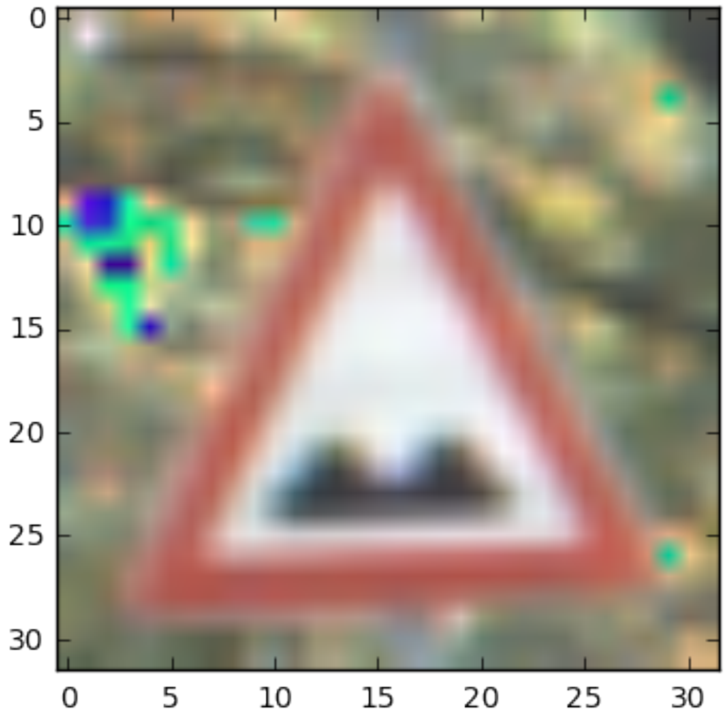
img 34



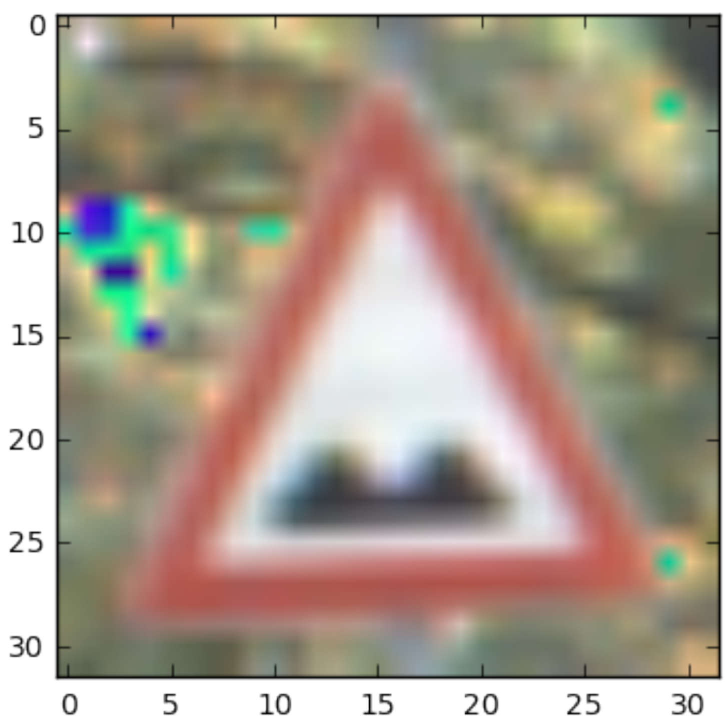
img 50



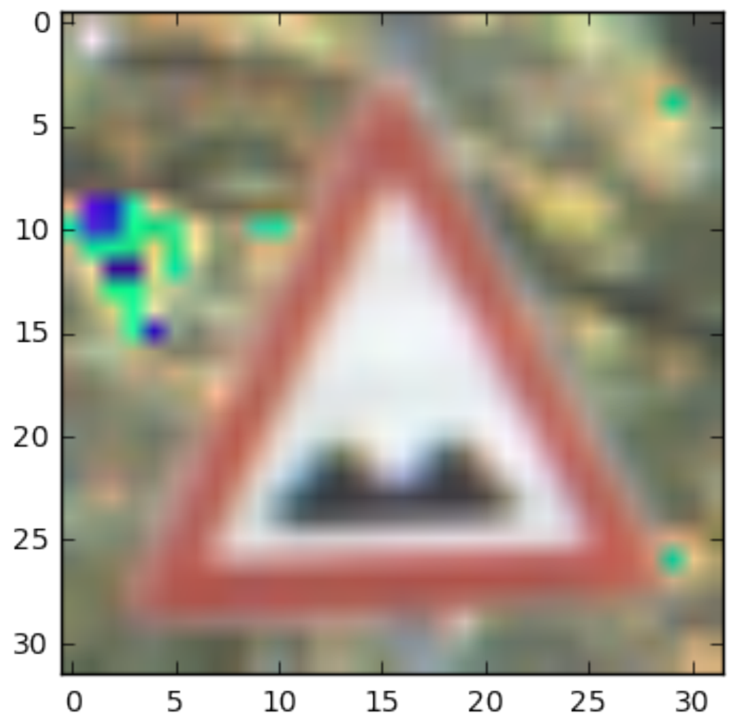
img 64



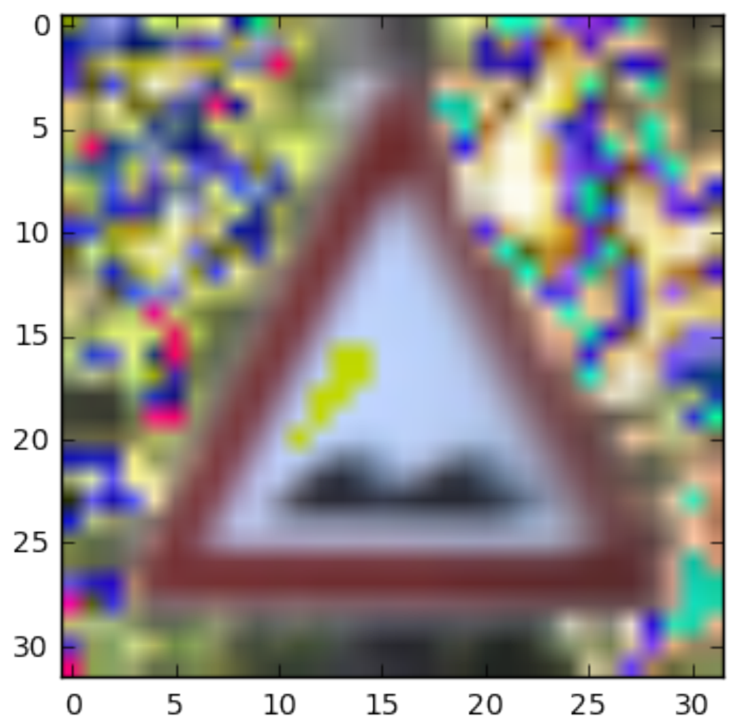
img 64



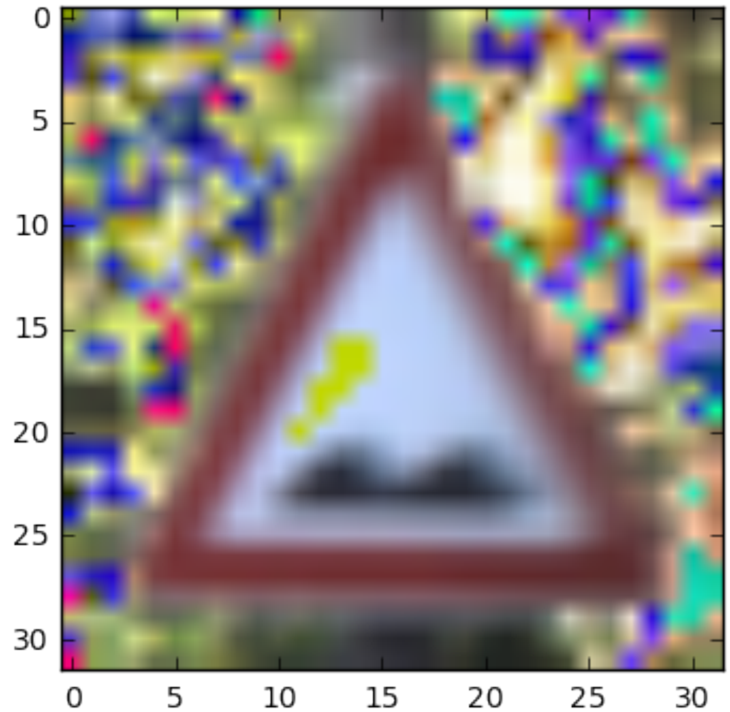
img 64



img 80



img 80




```
The model predicted 1 out of 5 signs correctly.
```

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,  0.0789
3497,
         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 [10]:
```
### Print out the top five softmax probabilities for the predictions o
n the German traffic sign images found on the web.
### Feel free to use as many code cells as needed.
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print(sess)
    accuracy = sess.run(tf.nn.top_k(tf.nn.softmax(logits), k=3), feed_
dict={x:web_imgs})
    print(accuracy)
```

```
<tensorflow.python.client.session.Session object at 0x7fb60c777160>
TopKV2(values=array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]], dtype=float32), indices=array([[11,  0,  1],
       [11,  0,  1],
       [11,  0,  1],
       [11,  0,  1],
       [11,  0,  1]], dtype=int32))
```

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 [11]:
```
### 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 th
e feature maps
# tf_activation: should be a tf variable name used during your trainin
g procedure that represents the calculated state of a specific weight
layer
# activation_min/max: can be used to view the activation contrast in m
ore detail, by default matplot sets min and max to the actual min and
max values of the output
# plt_num: used to plot out multiple different weight feature map sets
on the same block, just extend the plt number for each new feature map
entry

def outputFeatureMap(image_input, tf_activation, activation_min=-1, ac
tivation_max=-1 ,plt_num=1):
    # Here make sure to preprocess your image_input in a way your netw
ork expects
    # with size, normalization, ect if needed
    # image_input =
    # Note: x should be the same name as your network's tensorflow dat
a placeholder variable
    # If you get an error tf_activation is not defined it may be havin
g 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 ma
ps to show on each row and column
        plt.title('FeatureMap ' + str(featuremap)) # displays the feat
ure map number
        if activation_min != -1 & activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="n
earest", vmin =activation_min, vmax=activation_max, cmap="gray")
        elif activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="n
earest", vmax=activation_max, cmap="gray")
        elif activation_min !=-1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="n
earest", vmin=activation_min, cmap="gray")
        else:
            plt.imshow(activation[0,:,:, featuremap], interpolation="n
earest", cmap="gray")
```