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.

In [1]:

```
import tensorflow as tf
import numpy as np
```

Step 0: Load The Data

In [2]:

```
import pickle

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

training_file = 'traffic-signs-data/train.p'
validation_file='traffic-signs-data/valid.p'
testing_file = 'traffic-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 [3]:

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

# TODO: Number of training examples
n_train = X_train.shape[0]

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

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

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

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

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

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

Include an exploratory visualization of the dataset

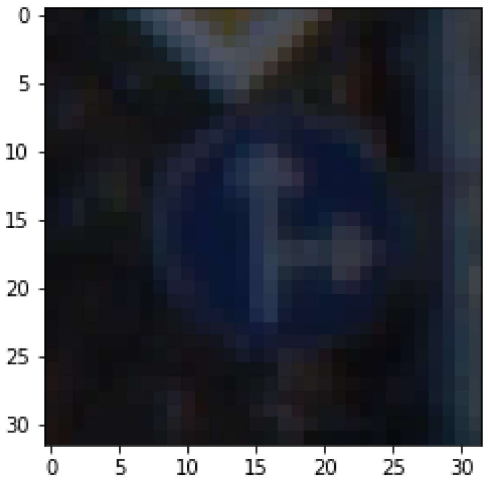
Visualize the German Traffic Signs Dataset using the pickled file(s). This is open ended, suggestions include: plotting traffic sign images, plotting the count of each sign, etc.

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.

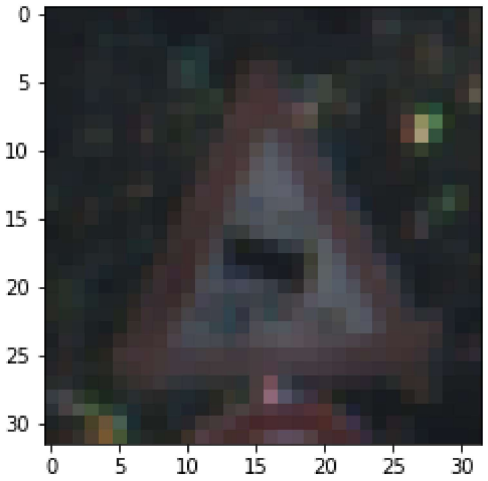
NOTE: It's recommended you start with something simple first. If you wish to do more, come back to it after you've completed the rest of the sections. It can be interesting to look at the distribution of classes in the training, validation and test set. Is the distribution the same? Are there more examples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

In [4]:

```
### Data exploration visualization code goes here.
### Feel free to use as many code cells as needed.
#!pip install matplotlib
import matplotlib.pyplot as plt
# Visualizations will be shown in the notebook.


%matplotlib inline
plt.imshow(X_train[1000 ])
plt.show()
print(y_train[1000])

plt.imshow(X_train[2000 ])
plt.show()
print(y_train[2000])
```



36



23

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 [5]:

```
### Preprocess the data here. It is required to normalize the data. Other preprocessing steps co
uld include
### converting to grayscale, etc.
### Feel free to use as many code cells as needed.


row = 32
col = 32
channel=3

X_train = X_train.reshape(X_train.shape[0],row,col,channel)
X_valid = X_valid.reshape(X_valid.shape[0],row,col,channel)
X_test = X_test.reshape(X_test.shape[0],row,col,channel)


i_shape = (row,col,channel)

X_train = X_train.astype('float32')
X_valid = X_valid.astype('float32')
X_test = X_test.astype('float32')

#Normalization... all pixcel values will be between 0 and 1.
X_train /= 255
X_valid /= 255
X_test /= 255

#all pixcel values will be between -1 and 1.
#I tested it too and got similar performance.
#X_train = (X_train-128)/128
#X_valid = (X_valid-128)/128
#X_test = (X_test-128)/128
```

Model Architecture

In [6]:

```
### Define your architecture here.
### Feel free to use as many code cells as needed.
x = tf.placeholder(tf.float32, (None, 32, 32, 3))
y = tf.placeholder(tf.int32, (None))

one_hot_y = tf.one_hot(y, 43)


# This architecture is based on LeNet.
# I added more filters to recognize more patterns. First convolution layer has 32 filters.
# second convolution has 64 filters.
########################################################################################################
#Layer 1: Convolutional. Input = 32x32x3. Output = 28x28x32
L = tf.nn.conv2d(x,
                 tf.Variable(tf.truncated_normal(shape=(5,5,3,32),mean=0,stddev=0.1)),
                 strides=[1,1,1,1],
                 padding='VALID')    +    tf.Variable(tf.zeros(32))
L = tf.nn.relu(L)

#max Pooling. Input = 28x28x32. Output = 14x14x32
L = tf.nn.max_pool(L, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')
```

```
#########################################################################################################
#Layer 2: Convolutional. Output = 10x10x64
L = tf.nn.conv2d(L,
                 tf.Variable(tf.truncated_normal(shape=(5,5,32,64),mean=0,stddev=0.1)),
                 strides=[1,1,1,1],
                 padding='VALID')      + tf.Variable(tf.zeros(64))
L = tf.nn.relu(L)

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

#flatten
L   = tf.contrib.layers.flatten(L)

#########################################################################################################
#Layer 3: Fully Connected. Input = 5*5*64 Output = 256.
L   = tf.matmul(L,  tf.Variable(tf.truncated_normal(shape=(5*5*64,256),mean=0,stddev=0.1))) + tf
.Variable(tf.zeros(256))
L    = tf.nn.relu(L)
#########################################################################################################
#Layer 4: drop out
keep_prob1  = tf.placeholder_with_default(1.0, shape=())
L = tf.nn.dropout(L, keep_prob1)

#########################################################################################################
#Layer 5: Fully Connected. Input = 256. Output = 128.
L   = tf.matmul(L, tf.Variable(tf.truncated_normal(shape=(256,128),mean=0,stddev=0.1))) + tf.Var
iable(tf.zeros(128))
L    = tf.nn.relu(L)
#########################################################################################################
#Layer 6: drop out
keep_prob2  = tf.placeholder_with_default(1.0, shape=())
L = tf.nn.dropout(L, keep_prob2)
#########################################################################################################
#Layer 7: Fully Connected. Input = 128. Output = 43.
#this one will be used next cell.
logits = tf.matmul(L,tf.Variable(tf.truncated_normal(shape=(128, 43), mean = 0, stddev = 0.1)) )
 +  tf.Variable(tf.zeros(43))
```

In [7]:

```
rate = 0.001

BATCH_SIZE =128

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 = rate)
training  = optimizer.minimize(loss)

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))
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_SIZE]
        accuracy = sess.run(accuracy_op  , feed_dict={x: batch_x, y: batch_y})
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples
```

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 [8]:

```
### Train your model here.
### Calculate and report the accuracy on the training and validation set.
### Once a final model architecture is selected,
### the accuracy on the test set should be calculated and reported as well.
### Feel free to use as many code cells as needed.


# Training steps
with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())
  cnt = 30000
  best=0
  for step in range(cnt):
    sel4batch  = np.random.choice( len(X_train)-1,BATCH_SIZE, replace=False)
    Xbatch = X_train[sel4batch]
    ybatch = y_train[sel4batch]

    sess.run(training , feed_dict={x: Xbatch, y:ybatch,keep_prob1:0.7, keep_prob2:0.7})
    if step% 100 == 0 :
        print(  100*(step/ cnt) ,'% done ---------------------------')
        train_acc  = evaluate(X_train, y_train)
        val_acc  = evaluate(X_valid, y_valid)
        print("train accuracy =" ,train_acc ," / valication accuracy:",val_acc)
        #the best valication accuracy will be chosen for testing.
        if val_acc  > best:
            saver.save(sess, './my_traffic_sign_classifier')
            print("-- Model saved--")
            best = val_acc


  #print("...test accuracy...")
  #print(cnt, sess.run(accuracy_op , feed_dict={x: X_test , y: y_test }))

with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    test_acc  = evaluate(X_test, y_test)
    print("Test accuracy =", test_acc)
```

```
WARNING:tensorflow:From C:₩Users₩user1₩AppData₩Local₩Continuum₩anaconda3₩envs₩tens
orflow₩lib₩site-packages₩tensorflow₩python₩util₩tf_should_use.py:107: initialize_a
ll_variables (from tensorflow.python.ops.variables) is deprecated and will be remo
ved after 2017-03-02.
Instructions for updating:
Use `tf.global_variables_initializer` instead.
0.0 % done ----------------------------
train accuracy = 0.0490243972528  / valication accuracy: 0.0458049886621
-- Model saved--
0.33333333333333337 % done ----------------------------
train accuracy = 0.658323515044  / valication accuracy: 0.588435373933
-- Model saved--
0.6666666666666667 % done ----------------------------
train accuracy = 0.88901979942  / valication accuracy: 0.830158730159
-- Model saved--
1.0 % done ----------------------------
train accuracy = 0.944050116383  / valication accuracy: 0.881632653088
-- Model saved--
1.3333333333333335 % done ----------------------------
train accuracy = 0.971033650392  / valication accuracy: 0.925623582766
-- Model saved--
1.6666666666666667 % done ----------------------------
train accuracy = 0.972413000374  / valication accuracy: 0.930839002268
-- Model saved--
2.0 % done ----------------------------
train accuracy = 0.98267191586  / valication accuracy: 0.929024943311
2.3333333333333335 % done ----------------------------
train accuracy = 0.988103106411  / valication accuracy: 0.953287981859
-- Model saved--
2.666666666666667 % done ----------------------------
train accuracy = 0.990833069916  / valication accuracy: 0.940816326531
3.0 % done ----------------------------
train accuracy = 0.993074513635  / valication accuracy: 0.952380952138
3.3333333333333335 % done ----------------------------
train accuracy = 0.993735452168  / valication accuracy: 0.952380952381
3.6666666666666665 % done ----------------------------
train accuracy = 0.992700939682  / valication accuracy: 0.9392290247
4.0 % done ----------------------------
train accuracy = 0.99609184172  / valication accuracy: 0.955102041087
-- Model saved--
4.333333333333334 % done ----------------------------
train accuracy = 0.995919422972  / valication accuracy: 0.953968254239
4.666666666666667 % done ----------------------------
train accuracy = 0.996063105262  / valication accuracy: 0.952607710021
5.0 % done ----------------------------
train accuracy = 0.997040144832  / valication accuracy: 0.9569160998
-- Model saved--
5.333333333333334 % done ----------------------------
train accuracy = 0.997557401075  / valication accuracy: 0.961904761661
-- Model saved--
5.666666666666666 % done ----------------------------
train accuracy = 0.997701083364  / valication accuracy: 0.958730159
6.0 % done ----------------------------
train accuracy = 0.997614873991  / valication accuracy: 0.965532879819
-- Model saved--
6.333333333333334 % done ----------------------------
train accuracy = 0.997614873991  / valication accuracy: 0.964399092971
6.666666666666667 % done ----------------------------
train accuracy = 0.998706859393  / valication accuracy: 0.966439909567
-- Model saved--
7.000000000000001 % done ----------------------------
```

```
train accuracy = 0.99887927814  / valication accuracy: 0.966893424036
-- Model saved--
7.333333333333333 % done --------------------------
train accuracy = 0.996235524009  / valication accuracy: 0.962131519274
7.666666666666666 % done --------------------------
train accuracy = 0.997298772953  / valication accuracy: 0.965306122206
8.0 % done --------------------------
train accuracy = 0.998218339607  / valication accuracy: 0.965532879819
8.333333333333332 % done --------------------------
train accuracy = 0.998448231271  / valication accuracy: 0.964172335871
8.666666666666668 % done --------------------------
train accuracy = 0.998074657318  / valication accuracy: 0.964625850367
9.0 % done --------------------------
train accuracy = 0.998764332308  / valication accuracy: 0.958276643748
9.333333333333334 % done --------------------------
train accuracy = 0.999252852093  / valication accuracy: 0.970521541977
-- Model saved--
9.666666666666666 % done --------------------------
train accuracy = 0.999425270841  / valication accuracy: 0.975736961451
-- Model saved--
10.0 % done --------------------------
train accuracy = 0.999195379178  / valication accuracy: 0.975283446739
10.333333333333334 % done --------------------------
train accuracy = 0.998649386477  / valication accuracy: 0.968027211155
10.666666666666668 % done --------------------------
train accuracy = 0.99887927814  / valication accuracy: 0.961451247193
11.0 % done --------------------------
train accuracy = 0.999540216673  / valication accuracy: 0.965759637215
11.333333333333332 % done --------------------------
train accuracy = 0.999310325009  / valication accuracy: 0.963945578502
11.666666666666666 % done --------------------------
train accuracy = 0.999051696888  / valication accuracy: 0.97074829959
12.0 % done --------------------------
train accuracy = 0.999597689589  / valication accuracy: 0.971201814086
12.333333333333334 % done --------------------------
train accuracy = 0.998850541682  / valication accuracy: 0.967573695902
12.666666666666668 % done --------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.974149660134
13.0 % done --------------------------
train accuracy = 0.998448231271  / valication accuracy: 0.96462585034
13.333333333333334 % done --------------------------
train accuracy = 0.999339061467  / valication accuracy: 0.971428571185
13.666666666666666 % done --------------------------
train accuracy = 0.99706888129  / valication accuracy: 0.963945577988
14.000000000000002 % done --------------------------
train accuracy = 0.999281588551  / valication accuracy: 0.961678004805
14.333333333333334 % done --------------------------
train accuracy = 0.998390758355  / valication accuracy: 0.963945577988
14.666666666666666 % done --------------------------
train accuracy = 0.99887927814  / valication accuracy: 0.965759636945
15.0 % done --------------------------
train accuracy = 0.999454007299  / valication accuracy: 0.968707483264
15.333333333333332 % done --------------------------
train accuracy = 0.999683898963  / valication accuracy: 0.974376417504
15.666666666666668 % done --------------------------
train accuracy = 0.999626426047  / valication accuracy: 0.967346938776
16.0 % done --------------------------
train accuracy = 0.998620650019  / valication accuracy: 0.97029478458
16.333333333333332 % done --------------------------
train accuracy = 0.999425270841  / valication accuracy: 0.975510203838
16.666666666666664 % done --------------------------
```

```
train accuracy = 0.999798844794  / valication accuracy: 0.976417233317
-- Model saved--
17.0 % done --------------------------
train accuracy = 0.999597689589  / valication accuracy: 0.965759637215
17.333333333333336 % done --------------------------
train accuracy = 0.999683898963  / valication accuracy: 0.966666666964
17.666666666666668 % done --------------------------
train accuracy = 0.999712635421  / valication accuracy: 0.972335600664
18.0 % done --------------------------
train accuracy = 0.998591913561  / valication accuracy: 0.973696144881
18.333333333333332 % done --------------------------
train accuracy = 0.999626426047  / valication accuracy: 0.977097505939
-- Model saved--
18.666666666666668 % done --------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.968253968011
19.0 % done --------------------------
train accuracy = 0.999597689589  / valication accuracy: 0.977551020678
-- Model saved--
19.333333333333332 % done --------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.971655329069
19.666666666666664 % done --------------------------
train accuracy = 0.998764332308  / valication accuracy: 0.957369614512
20.0 % done --------------------------
train accuracy = 0.998160866692  / valication accuracy: 0.967573696415
20.333333333333332 % done --------------------------
train accuracy = 0.999109169804  / valication accuracy: 0.96485260798
20.666666666666668 % done --------------------------
train accuracy = 0.999339061467  / valication accuracy: 0.970294784337
21.0 % done --------------------------
train accuracy = 0.999626426047  / valication accuracy: 0.972108843537
21.333333333333336 % done --------------------------
train accuracy = 0.998074657318  / valication accuracy: 0.965532880089
21.666666666666668 % done --------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.972108843537
22.0 % done --------------------------
train accuracy = 0.999367797925  / valication accuracy: 0.969614512742
22.333333333333332 % done --------------------------
train accuracy = 0.999597689589  / valication accuracy: 0.967346939073
22.666666666666664 % done --------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.97097505696
23.0 % done --------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.981859410701
-- Model saved--
23.333333333333332 % done --------------------------
train accuracy = 0.998362021897  / valication accuracy: 0.964399092998
23.666666666666668 % done --------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.975056689342
24.0 % done --------------------------
train accuracy = 0.999712635421  / valication accuracy: 0.973696145125
24.333333333333336 % done --------------------------
train accuracy = 0.999683898963  / valication accuracy: 0.978231292787
24.666666666666668 % done --------------------------
train accuracy = 0.999454007299  / valication accuracy: 0.969614512742
25.0 % done --------------------------
train accuracy = 0.999798844794  / valication accuracy: 0.975056689099
25.333333333333336 % done --------------------------
train accuracy = 0.999655162505  / valication accuracy: 0.973469388025
25.666666666666664 % done --------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.979138321995
26.0 % done --------------------------
train accuracy = 0.99916664272  / valication accuracy: 0.96666666645
```

```
26.333333333333332 % done ----------------------------
train accuracy = 0.999281588551  / valication accuracy: 0.973469388025
26.66666666666668 % done ----------------------------
train accuracy = 0.999712635421  / valication accuracy: 0.967120181406
27.0 % done ----------------------------
train accuracy = 0.999798844794  / valication accuracy: 0.973696144881
27.333333333333332 % done ----------------------------
train accuracy = 0.999540216673  / valication accuracy: 0.975963718821
27.66666666666668 % done ----------------------------
train accuracy = 0.999597689589  / valication accuracy: 0.978911564896
28.000000000000004 % done ----------------------------
train accuracy = 0.999655162505  / valication accuracy: 0.975056689099
28.333333333333332 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.978684807013
28.66666666666668 % done ----------------------------
train accuracy = 0.999281588551  / valication accuracy: 0.968027210911
28.999999999999996 % done ----------------------------
train accuracy = 0.999626426047  / valication accuracy: 0.966893424307
29.333333333333332 % done ----------------------------
train accuracy = 0.999712635421  / valication accuracy: 0.971882086438
29.66666666666668 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.975736961749
30.0 % done ----------------------------
train accuracy = 0.999310325009  / valication accuracy: 0.97052154222
30.333333333333336 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.975056689099
30.66666666666664 % done ----------------------------
train accuracy = 0.999798844794  / valication accuracy: 0.974603174603
31.0 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.972789115646
31.333333333333336 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.977551020408
31.66666666666664 % done ----------------------------
train accuracy = 0.999913790626  / valication accuracy: 0.97641723383
32.0 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.976643990984
32.33333333333333 % done ----------------------------
train accuracy = 0.998994223972  / valication accuracy: 0.965532879819
32.66666666666664 % done ----------------------------
train accuracy = 0.999626426047  / valication accuracy: 0.969160998003
33.0 % done ----------------------------
train accuracy = 0.999109169804  / valication accuracy: 0.975736961451
33.33333333333333 % done ----------------------------
train accuracy = 0.999770108336  / valication accuracy: 0.967346938803
33.66666666666664 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.97460317436
34.0 % done ----------------------------
train accuracy = 0.999396534383  / valication accuracy: 0.968480725921
34.333333333333336 % done ----------------------------
train accuracy = 0.999137906262  / valication accuracy: 0.97619047619
34.66666666666667 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.978911564626
35.0 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.97074829959
35.333333333333336 % done ----------------------------
train accuracy = 0.999913790626  / valication accuracy: 0.970975056689
35.66666666666667 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.974149659864
36.0 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.975283446982
36.333333333333336 % done ----------------------------
```

```
train accuracy = 0.999971263542  / valication accuracy: 0.979138322293
36.666666666666664 % done ----------------------------
train accuracy = 0.998591913561  / valication accuracy: 0.96938775548
37.0 % done ----------------------------
train accuracy = 0.999339061467  / valication accuracy: 0.969387755156
37.333333333333336 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.970975056689
37.666666666666664 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.975736961722
38.0 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.972562358033
38.333333333333336 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.978231292787
38.666666666666664 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.977551020435
39.0 % done ----------------------------
train accuracy = 0.999367797925  / valication accuracy: 0.965079365106
39.33333333333333 % done ----------------------------
train accuracy = 0.999511480215  / valication accuracy: 0.976643990686
39.666666666666664 % done ----------------------------
train accuracy = 0.999655162505  / valication accuracy: 0.975283446793
40.0 % done ----------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.977097505669
40.33333333333333 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.973015873016
40.666666666666664 % done ----------------------------
train accuracy = 0.998936751056  / valication accuracy: 0.958956916127
41.0 % done ----------------------------
train accuracy = 0.999770108336  / valication accuracy: 0.969387754886
41.333333333333336 % done ----------------------------
train accuracy = 0.999540216673  / valication accuracy: 0.967573696145
41.66666666666667 % done ----------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.97074829959
42.0 % done ----------------------------
train accuracy = 0.99902296043  / valication accuracy: 0.977097505426
42.333333333333336 % done ----------------------------
train accuracy = 0.999568953131  / valication accuracy: 0.979818594402
42.66666666666667 % done ----------------------------
train accuracy = 0.999683898963  / valication accuracy: 0.977551020408
43.0 % done ----------------------------
train accuracy = 0.999798844794  / valication accuracy: 0.97074829932
43.333333333333336 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.981179138322
43.666666666666664 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.973922902305
44.0 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.977777777534
44.333333333333336 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.975510204352
44.666666666666664 % done ----------------------------
train accuracy = 0.999655162505  / valication accuracy: 0.973242630656
45.0 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.976190476461
45.33333333333333 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.977777777534
45.666666666666664 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.975056689342
46.0 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.973469387512
46.33333333333333 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.974829932243
```

```
46.666666666666664 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.975283446469
47.0 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.965079365079
47.333333333333336 % done ----------------------------
train accuracy = 0.999396534383  / valication accuracy: 0.976190476461
47.66666666666667 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.969614512472
48.0 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.96462585061
48.333333333333336 % done ----------------------------
train accuracy = 0.999655162505  / valication accuracy: 0.965306122449
48.66666666666667 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.97052154195
49.0 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.976643991254
49.333333333333336 % done ----------------------------
train accuracy = 0.999913790626  / valication accuracy: 0.967573696145
49.666666666666664 % done ----------------------------
train accuracy = 0.999224115636  / valication accuracy: 0.965079364836
50.0 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.965986394855
50.33333333333333 % done ----------------------------
train accuracy = 0.999712635421  / valication accuracy: 0.969614512742
50.66666666666667 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.973696145152
51.0 % done ----------------------------
train accuracy = 0.998764332308  / valication accuracy: 0.968934240633
51.33333333333333 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.969614512769
51.66666666666667 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.974376417261
52.0 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.97437641699
52.33333333333333 % done ----------------------------
train accuracy = 0.999367797925  / valication accuracy: 0.969160997732
52.666666666666664 % done ----------------------------
train accuracy = 0.999425270841  / valication accuracy: 0.962358276914
53.0 % done ----------------------------
train accuracy = 0.999252852093  / valication accuracy: 0.968027210668
53.333333333333336 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.968480725921
53.666666666666664 % done ----------------------------
train accuracy = 0.999511480215  / valication accuracy: 0.960544217741
54.0 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.971428571212
54.333333333333336 % done ----------------------------
train accuracy = 0.999913790626  / valication accuracy: 0.966213151927
54.666666666666664 % done ----------------------------
train accuracy = 0.999913790626  / valication accuracy: 0.966439909351
55.00000000000001 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.969841270112
55.333333333333336 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.962811791681
55.666666666666664 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.968707483264
56.00000000000001 % done ----------------------------
train accuracy = 0.999137906262  / valication accuracy: 0.96689342382
56.333333333333336 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.96893424012
56.666666666666664 % done ----------------------------
```

```
train accuracy = 0.999798844794  / valication accuracy: 0.965079365079
56.99999999999999 % done ----------------------------
train accuracy = 0.999798844794  / valication accuracy: 0.972562358547
57.333333333333336 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.96893424012
57.666666666666664 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.967346938803
57.99999999999999 % done ----------------------------
train accuracy = 0.999683898963  / valication accuracy: 0.969841270166
58.333333333333336 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.972789115673
58.666666666666664 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.968253968254
59.0 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.971882085925
59.333333333333336 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.972335600664
59.66666666666667 % done ----------------------------
train accuracy = 0.999482743757  / valication accuracy: 0.964399092998
60.0 % done ----------------------------
train accuracy = 0.999597689589  / valication accuracy: 0.967800453515
60.333333333333336 % done ----------------------------
train accuracy = 0.99902296043  / valication accuracy: 0.960997732697
60.66666666666667 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.972335600664
61.0 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.97619047619
61.33333333333333 % done ----------------------------
train accuracy = 0.999339061467  / valication accuracy: 0.965532880089
61.66666666666667 % done ----------------------------
train accuracy = 0.999511480215  / valication accuracy: 0.963492063276
62.0 % done ----------------------------
train accuracy = 0.999051696888  / valication accuracy: 0.966213152225
62.33333333333333 % done ----------------------------
train accuracy = 0.999712635421  / valication accuracy: 0.969841269598
62.66666666666667 % done ----------------------------
train accuracy = 0.999396534383  / valication accuracy: 0.967346938776
63.0 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.969841269841
63.33333333333333 % done ----------------------------
train accuracy = 0.999770108336  / valication accuracy: 0.971882085952
63.66666666666667 % done ----------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.965306122152
64.0 % done ----------------------------
train accuracy = 0.999683898963  / valication accuracy: 0.971201813951
64.33333333333333 % done ----------------------------
train accuracy = 0.999712635421  / valication accuracy: 0.974603174657
64.66666666666666 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.970068027022
65.0 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.972562358331
65.33333333333333 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.982539682837
-- Model saved--
65.66666666666666 % done ----------------------------
train accuracy = 0.999597689589  / valication accuracy: 0.97868480731
66.0 % done ----------------------------
train accuracy = 0.999683898963  / valication accuracy: 0.979818594483
66.33333333333333 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.973242630169
66.66666666666666 % done ----------------------------
```

```
train accuracy = 0.999971263542  / valication accuracy: 0.98072562388
67.0 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.978231292787
67.33333333333333 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.975510203838
67.66666666666666 % done ----------------------------
train accuracy = 0.999913790626  / valication accuracy: 0.973242630169
68.0 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.971655328555
68.33333333333333 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.974376417531
68.66666666666667 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.966893424036
69.0 % done ----------------------------
train accuracy = 0.999770108336  / valication accuracy: 0.963492063546
69.33333333333334 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.978458049887
69.66666666666667 % done ----------------------------
train accuracy = 0.999597689589  / valication accuracy: 0.974376417288
70.0 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.970294784878
70.33333333333334 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.974603174901
70.66666666666667 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.974149659621
71.0 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.971201814113
71.33333333333334 % done ----------------------------
train accuracy = 0.999770108336  / valication accuracy: 0.971655328555
71.66666666666667 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.974376417504
72.0 % done ----------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.969841270112
72.33333333333334 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.973469388079
72.66666666666667 % done ----------------------------
train accuracy = 0.996350469892  / valication accuracy: 0.946485260771
73.0 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.968027210668
73.33333333333333 % done ----------------------------
train accuracy = 0.999540216673  / valication accuracy: 0.971428571483
73.66666666666667 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.971201814113
74.0 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.967346938586
74.33333333333333 % done ----------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.969387755129
74.66666666666667 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.971428571456
75.0 % done ----------------------------
train accuracy = 0.999655162505  / valication accuracy: 0.970975057014
75.33333333333333 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.972335600934
75.66666666666667 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.973242630413
76.0 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.975510203838
76.33333333333333 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.969614512228
76.66666666666667 % done ----------------------------
train accuracy = 0.999798844794  / valication accuracy: 0.971882086195
```

```
77.0 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.973469387539
77.33333333333333 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.973696144881
77.66666666666666 % done ----------------------------
train accuracy = 0.999770108336  / valication accuracy: 0.971882086168
78.0 % done ----------------------------
train accuracy = 0.998994223972  / valication accuracy: 0.962811791681
78.33333333333333 % done ----------------------------
train accuracy = 0.999511480215  / valication accuracy: 0.968934240741
78.66666666666666 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.983900226784
-- Model saved--
79.0 % done ----------------------------
train accuracy = 0.999252852093  / valication accuracy: 0.9730158728
79.33333333333333 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.973922902819
79.66666666666666 % done ----------------------------
train accuracy = 0.999655162505  / valication accuracy: 0.974603174684
80.0 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.971882085952
80.33333333333333 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.976643990957
80.66666666666666 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.977324263309
81.0 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.973696145395
81.33333333333333 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.97460317436
81.66666666666667 % done ----------------------------
train accuracy = 0.999913790626  / valication accuracy: 0.97641723383
82.0 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.975963719091
82.33333333333334 % done ----------------------------
train accuracy = 0.999655162505  / valication accuracy: 0.970975056581
82.66666666666667 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.973469387512
83.0 % done ----------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.971655329069
83.33333333333334 % done ----------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.973922902765
83.66666666666667 % done ----------------------------
train accuracy = 0.999913790626  / valication accuracy: 0.98072562361
84.0 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.981859410458
84.33333333333334 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.981405895962
84.66666666666667 % done ----------------------------
train accuracy = 0.999798844794  / valication accuracy: 0.970521541572
85.0 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.978231292787
85.33333333333334 % done ----------------------------
train accuracy = 0.999712635421  / valication accuracy: 0.977097505696
85.66666666666667 % done ----------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.978684807554
86.0 % done ----------------------------
train accuracy = 0.999683898963  / valication accuracy: 0.97482993173
86.33333333333333 % done ----------------------------
train accuracy = 0.99916664272  / valication accuracy: 0.960544217498
86.66666666666667 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.977551020435
```

```
87.0 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.977777777562
87.33333333333333 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.97482993173
87.66666666666667 % done ----------------------------
train accuracy = 0.999626426047  / valication accuracy: 0.96145124703
88.0 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.97278911543
88.33333333333333 % done ----------------------------
train accuracy = 0.999425270841  / valication accuracy: 0.973469388052
88.66666666666667 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.975963719091
89.0 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.97301587307
89.33333333333333 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.978458050184
89.66666666666666 % done ----------------------------
train accuracy = 0.999568953131  / valication accuracy: 0.978004535418
90.0 % done ----------------------------
train accuracy = 0.999913790626  / valication accuracy: 0.972108843808
90.33333333333333 % done ----------------------------
train accuracy = 0.999367797925  / valication accuracy: 0.975056689099
90.66666666666666 % done ----------------------------
train accuracy = 1.0  / valication accuracy: 0.975283446982
91.0 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.974149660161
91.33333333333333 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.977324263336
91.66666666666666 % done ----------------------------
train accuracy = 0.999913790626  / valication accuracy: 0.974829932297
92.0 % done ----------------------------
train accuracy = 0.999425270841  / valication accuracy: 0.968253968065
92.33333333333333 % done ----------------------------
train accuracy = 0.999770108336  / valication accuracy: 0.964399092727
92.66666666666666 % done ----------------------------
train accuracy = 0.999310325009  / valication accuracy: 0.967800453542
93.0 % done ----------------------------
train accuracy = 0.999683898963  / valication accuracy: 0.960090702705
93.33333333333333 % done ----------------------------
train accuracy = 0.999683898963  / valication accuracy: 0.96462585061
93.66666666666667 % done ----------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.966893424334
94.0 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.970294784608
94.33333333333334 % done ----------------------------
train accuracy = 0.999655162505  / valication accuracy: 0.980725623583
94.66666666666667 % done ----------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.973696145152
95.0 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.977324262795
95.33333333333334 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.975736961722
95.66666666666667 % done ----------------------------
train accuracy = 0.999770108336  / valication accuracy: 0.973696145395
96.0 % done ----------------------------
train accuracy = 0.999913790626  / valication accuracy: 0.975963719091
96.33333333333334 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.974829932243
96.66666666666667 % done ----------------------------
train accuracy = 0.999741371879  / valication accuracy: 0.977324263309
97.0 % done ----------------------------
```

```
train accuracy = 0.999540216673  / valication accuracy: 0.972562358601
97.33333333333334 % done ----------------------------
train accuracy = 0.999626426047  / valication accuracy: 0.971655329123
97.66666666666667 % done ----------------------------
train accuracy = 0.999885054168  / valication accuracy: 0.973469387512
98.0 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.974376417531
98.33333333333333 % done ----------------------------
train accuracy = 0.999511480215  / valication accuracy: 0.976417233587
98.66666666666667 % done ----------------------------
train accuracy = 0.999942527084  / valication accuracy: 0.973015873286
99.0 % done ----------------------------
train accuracy = 0.999971263542  / valication accuracy: 0.965306122719
99.33333333333333 % done ----------------------------
train accuracy = 0.999827581252  / valication accuracy: 0.97029478458
99.66666666666667 % done ----------------------------
train accuracy = 0.99985631771  / valication accuracy: 0.971428571429
INFO:tensorflow:Restoring parameters from .₩my_traffic_sign_classifier
Test accuracy = 0.970467141651
```

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 [9]:

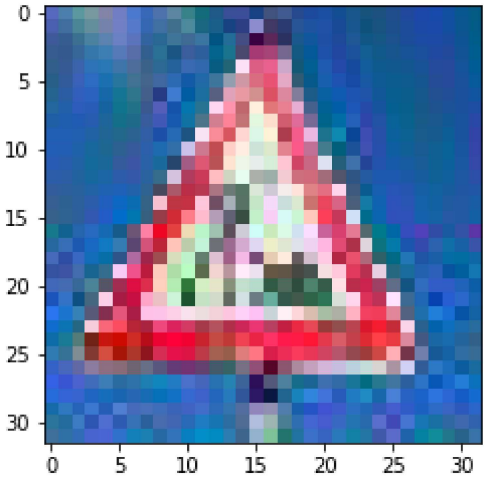
```
### Load the images and plot them here.
### Feel free to use as many code cells as needed.
#!pip install Pillow

import matplotlib.pyplot as plt

from PIL import Image

#I got these images from the web.
files=['caution.jpg','bumpy_road.jpg','30.jpg','100.jpg','stop.jpg']

#this will be having  new file names whose format is acceptable for the architecture.
new_files=[]

#show 5 sign images
for f in files:
    img = plt.imread(f)
    plt.imshow(img)
    plt.show()

# saving resized 32x32 images.
for f in files:
    im = Image.open(f)
    img2 = im.resize((32,32))
    #plt.imshow(img2)
    #plt.show()
    img2.save('resized_' + f)
    new_files.append('resized_' + f)




#show all resized files in new_files.
for f in new_files:
    img = plt.imread(f)
    plt.imshow(img)
    plt.show()
    print(img.shape)
```


0
25
50
75
100
125
150
175
200
225
0
50
100
150
200
250
300

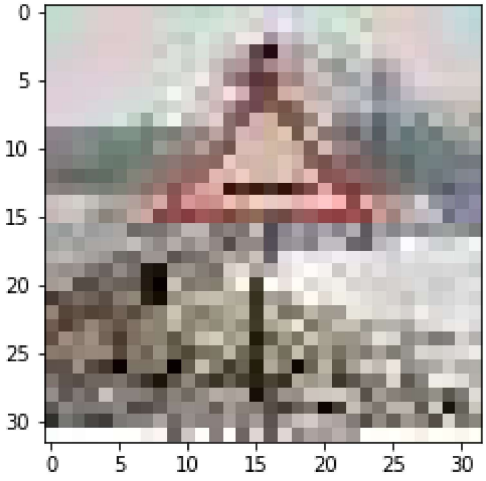

0
50
100
150
200
250
0
50
100
150


0
50
100
150
200
250
30
0
50
100
150

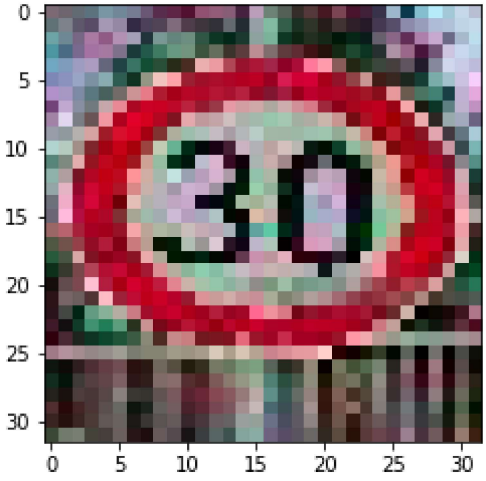

0
20
40
60
80
100
120
140
160
100
0
50
100
150
200
250



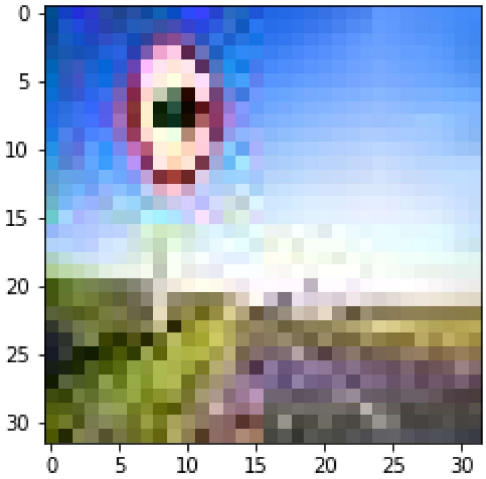
(32, 32, 3)



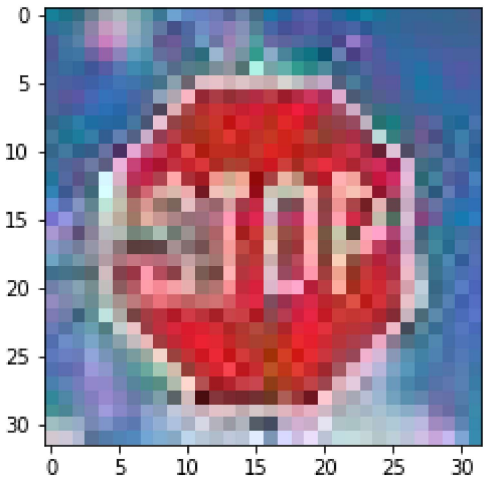
(32, 32, 3)



(32, 32, 3)



(32, 32, 3)



(32, 32, 3)

Predict the Sign Type for Each Image

In [10]:

```
### 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.

import numpy as np
X_real=[]

# I got these from signnames.csv
#caution(road work) , bumpy road , 30 , 100 ,stop
y_real=[25 ,22 , 1,7,14]

for f in new_files:
    img = plt.imread(f)
    X_real.append(img)
    print(img.shape)

X_real = np.array(X_real)

#preprocessing...
X_real = X_real.astype('float32')
X_real /= 255
```

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

In [11]:

```
correct_count = 0
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    prediction =  tf.argmax(logits, 1)
    predicted = prediction.eval(feed_dict={x: X_real }, session=sess)
    for (correct, predict) in zip(y_real,predicted):
        print("correct:",correct, "/predict:",predict)
        if correct == predict:
            correct_count += 1
```

```
INFO:tensorflow:Restoring parameters from .₩my_traffic_sign_classifier
correct: 25 /predict: 25
correct: 22 /predict: 26
correct: 1 /predict: 1
correct: 7 /predict: 13
correct: 14 /predict: 14
```

Analyze Performance

In [12]:

```
### 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 n
ew images.

print("The performance is " , (correct_count/5)*100 ,"%.")

# same output as below...
#with tf.Session() as sess:
#    saver.restore(sess, tf.train.latest_checkpoint('.'))
#    accuracy = evaluate(X_real, y_real)
#    print("Test Accuracy = " , accuracy )
```

```
The performance is  60.0 %.
```

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,  0.07893497,
         0.12789202],
       [ 0.28086119,  0.27569815,  0.08594638,  0.0178669 ,  0.18063401,
         0.15899337],
       [ 0.26076848,  0.23664738,  0.08020603,  0.07001922,  0.1134371 ,
         0.23892179],
       [ 0.11943333,  0.29198961,  0.02605103,  0.26234032,  0.1351348 ,
         0.16505091],
       [ 0.09561176,  0.34396535,  0.0643941 ,  0.16240774,  0.24206137,
         0.09155967]])
```

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

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

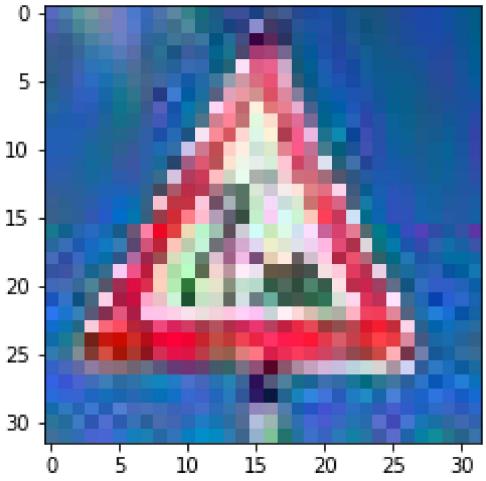
Looking just at the first row we get `[ 0.34763842,  0.24879643,  0.12789202]`, you can confirm these are the 3 largest probabilities in `a`. You'll also notice `[3, 0, 5]` are the corresponding indices.

In [13]:

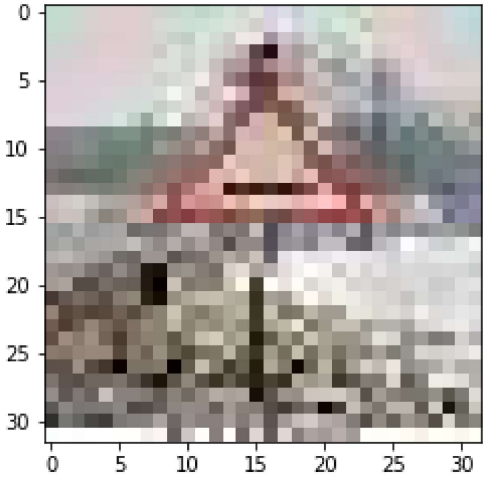
```
### Print out the top five softmax probabilities for the predictions on the German traffic sign
 images found on the web.
### Feel free to use as many code cells as needed.
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '-1'


with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    # trying to get the raw output of softmax
    out = logits.eval(feed_dict={x: X_real }, session=sess)
    top5 = sess.run(tf.nn.top_k(tf.constant(out), k=5))

    print("₩n₩n[Top 5 Probabilities of 5 traffic sign images]₩n")
    for fname, predicted in zip(new_files,top5[1]):
        img = plt.imread(fname)
        plt.imshow(img)
        plt.show()
        print("Top 5", predicted)
```

```
INFO:tensorflow:Restoring parameters from .Wmy_traffic_sign_classifier
```

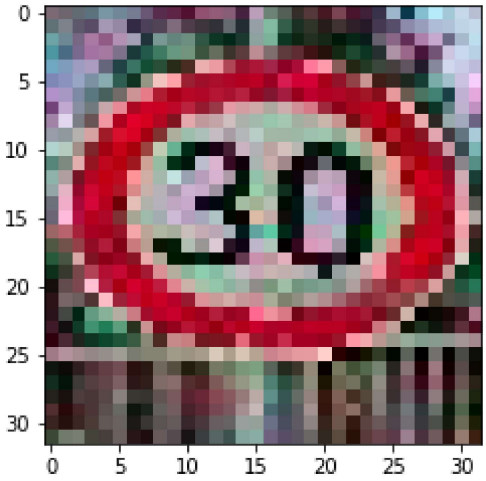
[Top 5 Probabilities of 5 traffic sign images]



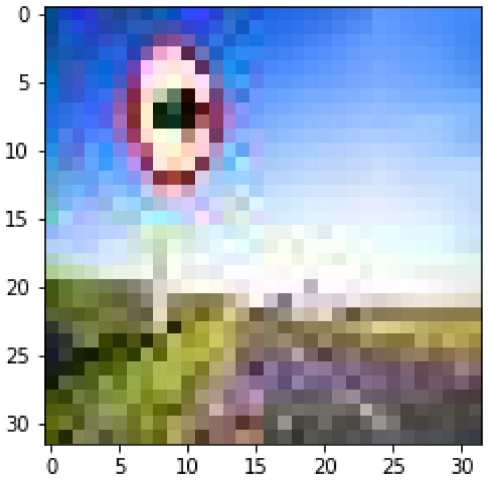
Top 5 [25 31 29 22 30]



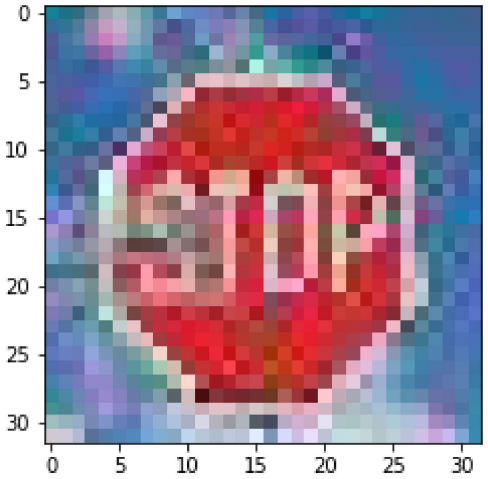
Top 5 [26 18 20 17 11]



Top 5 [1 5 2 3 15]



Top 5 [13 15 12 1 26]



Top 5 [14 17 5 1 3]

In [14]:

```
print(top5)

sum=0
for  i,pr,idx  in zip( range(5) , top5[0],top5[1]):
    softmax_values = pr/np.sum(pr)
    print("image", i+1,":",softmax_values, " / ", idx)
```

```
TopKV2(values=array([[ 44.44266129,  29.34724617,  21.29602814,  16.27615929,
         9.78890133],
       [  7.8233881 ,   7.09062338,   1.67975092,   0.57077837,
        -0.57676923],
       [ 33.67581558,   7.99736691,   7.37483215,   1.64076352,
         0.14412104],
       [ 15.23113823,  14.37021446,  13.47384167,   6.13779402,
         0.97665024],
       [ 62.75913239,   9.65332127,   8.77884579,   6.94516134,
        -4.66549683]], dtype=float32), indices=array([[25, 31, 29, 22, 30],
       [26, 18, 20, 17, 11],
       [ 1,  5,  2,  3, 15],
       [13, 15, 12,  1, 26],
       [14, 17,  5,  1,  3]]))
image 1 : [ 0.36683694  0.24223693  0.17578086  0.13434605  0.08079918]  /  [25 31
29 22 30]
image 2 : [ 0.47163591  0.42746091  0.10126442  0.03440959 -0.03477075]  /  [26 18
20 17 11]
image 3 : [ 0.66248077  0.15732659  0.14507991  0.03227759  0.00283519]  /  [ 1  5
  2  3 15]
image 4 : [ 0.30347174  0.28631833  0.26845863  0.12229205  0.0194592 ]  /  [13 15
12  1 26]
image 5 : [ 0.75186783  0.11564886  0.10517246  0.08320452 -0.05589365]  /  [14 17
  5  1  3]
```

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.

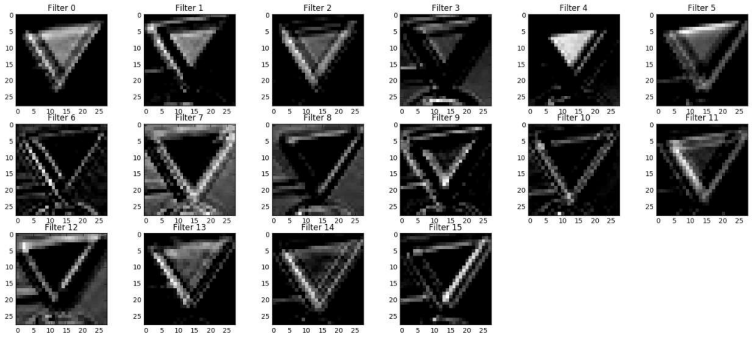
> **Note**: Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to \n", "**File -> Download as -> HTML (.html)**.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



Your output should look something like this (above)