Self-Driving Car Engineer Nanodegree

Deep Learning

Project: Build a Traffic Sign Recognition Classifier

This Jupyter notebook contains the code pipeline for a traffic sign classifier tuned for project 2 of Term 1. Python libraries used in this project:

- `pickle`: save and load binary python objects
- `numpy`: algebra calculations
- `matplotlib`: plots and image loading
- `tensorflow`: machine learning framework
- `sklearn`: machine learning framework

Packages `scikit-image` and `cv2` were tested, but not used in final form.

Step 0: Load The Data

Input data was provided in binary files.

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

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_train0, y_train0 = train['features'], train['labels']
X_valid0, y_valid0 = valid['features'], valid['labels']
X_test0, y_test0 = 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.

Basic Summary of the Data Set Using Python and Numpy

In [2]:

```
# Number of training examples
n_train = X_train0.shape[0]

# Number of validation examples
n_validation = X_valid0.shape[0]

# Number of testing examples.
n_test = X_test0.shape[0]

# What's the shape of an traffic sign image?
image_shape = X_train0.shape[1], X_train0.shape[2], X_train0.shape[
3]

# How many unique classes/labels there are in the dataset.
n_classes = np.max(np.unique(y_train0).shape)

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)
print("labels shape: ", y_train0.shape)
print(y_train0[0:4])
```

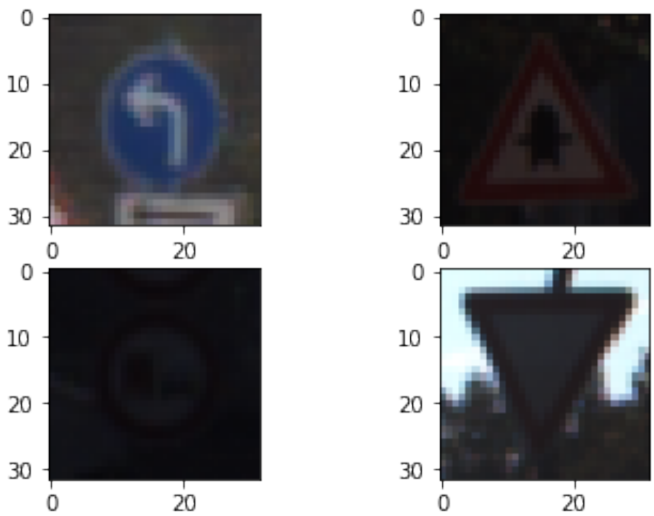
```
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
labels shape:  (34799,)
[41 41 41 41]
```

Visualization of the dataset

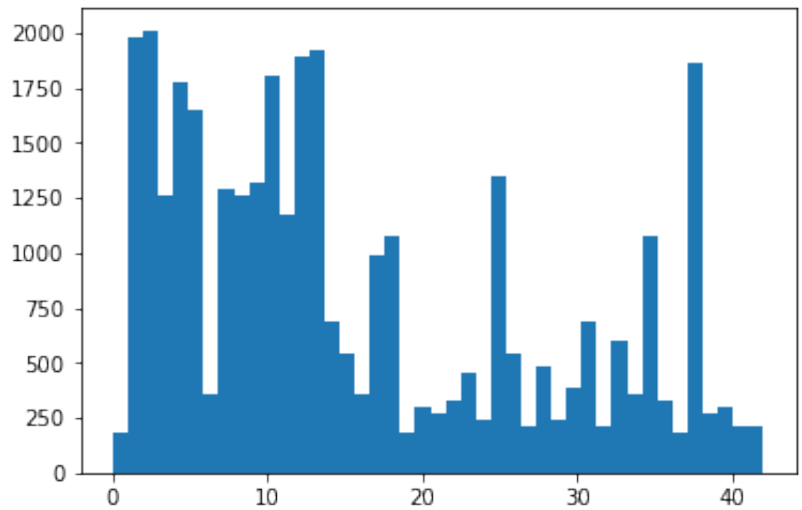
Next, it is presented 4 random images from the training dataset and a histogram of labels (output).

In [3]:
```
### Data exploration visualization code goes here.
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
# Visualizations will be shown in the notebook.
%matplotlib inline

im1 = np.round(np.random.random_sample()*n_train).astype(int)
im2 = np.round(np.random.random_sample()*n_train).astype(int)
im3 = np.round(np.random.random_sample()*n_train).astype(int)
im4 = np.round(np.random.random_sample()*n_train).astype(int)

print(im1, im2, im3, im4)

fig, axs = plt.subplots(nrows=2, ncols=2)
axs[0, 0].imshow(X_train0[im1,:,:,:])
axs[0, 1].imshow(X_train0[im2,:,:,:])
axs[1, 0].imshow(X_train0[im3,:,:,:])
axs[1, 1].imshow(X_train0[im4,:,:,:])

plt.show()

plt.figure()
plt.hist(train['labels'], n_classes)
```

```
20072 9319 18600 22108
```



```
Out[3]: (array([  180.,  1980.,  2010.,  1260.,  1770.,  1650.,   360.,  1
        290.,
                 1260.,  1320.,  1800.,  1170.,  1890.,  1920.,   690.,
        540.,
                  360.,   990.,  1080.,   180.,   300.,   270.,   330.,
        450.,
                  240.,  1350.,   540.,   210.,   480.,   240.,   390.,
        690.,
                  210.,   599.,   360.,  1080.,   330.,   180.,  1860.,
        270.,
                  300.,   210.,   210.]),
         array([  0.        ,   0.97674419,   1.95348837,   2.93023256,
                  3.90697674,   4.88372093,   5.86046512,   6.8372093 ,
                  7.81395349,   8.79069767,   9.76744186,  10.74418605,
                 11.72093023,  12.69767442,  13.6744186 ,  14.65116279,
                 15.62790698,  16.60465116,  17.58139535,  18.55813953,
                 19.53488372,  20.51162791,  21.48837209,  22.46511628,
                 23.44186047,  24.41860465,  25.39534884,  26.37209302,
                 27.34883721,  28.3255814 ,  29.30232558,  30.27906977,
                 31.25581395,  32.23255814,  33.20930233,  34.18604651,
                 35.1627907 ,  36.13953488,  37.11627907,  38.09302326,
                 39.06976744,  40.04651163,  41.02325581,  42.        ]),
         <a list of 43 Patch objects>)
```



Labels in the training set are not equally distributed.

Step 2: Design and Test a Model Architecture

This project is based on LeNet-5 architecture. LeNet consists of 2 convolutional layers, 1 max pool and 2 fully connected layer.

The architecture proposed here adds another fully connected layer and uses a deeper convolutional filter.

Pre-process the Data Set

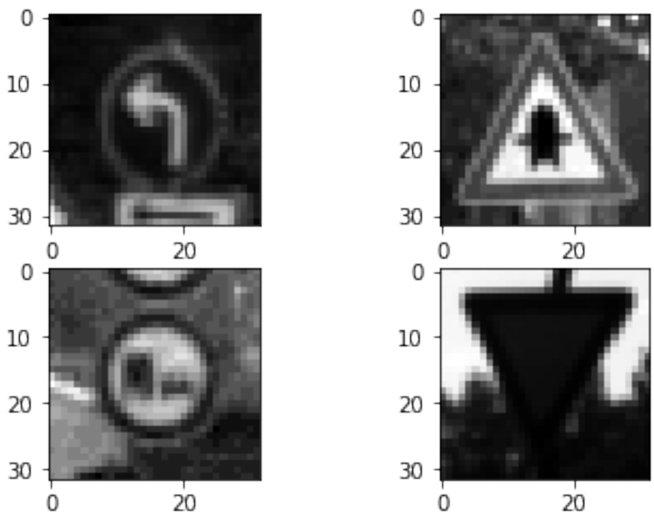
Pre-processing is made in two steps:

- grayscaling with: Gray = 0.299 Red + 0.587 Green + 0.114 Blue
- normalizing pixel intensity

In [4]:
```
### Preprocess the data here.
from sklearn.utils import shuffle
#from skimage import exposure
#from skimage.color import rgb2gray
from datetime import datetime

# training data
X_train = np.zeros((n_train, X_train0.shape[1], X_train0.shape[2],
1), np.int)

# validation data
X_valid = np.zeros((n_validation, X_valid0.shape[1], X_valid0.shape
[2], 1), np.int)

# grayscale luminosity
#lum = np.ndarray((3,), np.float, np.array([0.210, 0.720, 0.070]))
lum = np.ndarray((3,), np.float, np.array([0.299, 0.587, 0.114]))

y_train = y_train0
y_valid = y_valid0

for i in range(n_train):
    # grayscale with CV2
    #X_train[i,:,:,:] = cv2.cvtColor(X_train[i,:,:,:], cv2.COLOR_RG
B2GRAY)
    # grayscale with luminosity
    # 0.21 R + 0.72 G + 0.07 B.
    #X_train[i,:,:,0] = (X_train1[i,:,:,0]*0.21 + X_train1[i,:,:,1]
*0.72 + X_train1[i,:,:,2]*0.07).astype(int)
    # also suggested in CarND online foruns
    #X_train[i,:,:,0] = np.dot(X_train1[i,:,:,:], lum).astype(int)
```

```
    #p2, p98 = np.percentile(X_train1[i,:,:,:], (2, 98))
    #X_train1[i,:,:,:] = exposure.rescale_intensity(X_train1[i,:,:,
:], in_range=(p2, p98))
    #pass
    #X_train[i,:,:,0] = np.dot(X_train1[i,:,:,:]
    #X_train[i,:,:,0] = rgb2gray(X_train1[i,:,:,:])

    # grayscale
    X_train[i,:,:,0] = np.dot(X_train0[i,:,:,:], lum)


for i in range(n_validation):
    X_valid[i,:,:,0] = np.dot(X_valid0[i,:,:,:], lum).astype(int)

# normalized after grayscale and save computing costs
X_train = (X_train/255.0)-0.5
X_valid = (X_valid/255.0)-0.5

# check data
print("max min ",np.amin(X_train), np.amax(X_train))

fig, axs = plt.subplots(nrows=2, ncols=2)
axs[0, 0].imshow(X_train[im1,:,:,0], cmap=plt.cm.gray)
axs[0, 1].imshow(X_train[im2,:,:,0], cmap=plt.cm.gray)
axs[1, 0].imshow(X_train[im3,:,:,0], cmap=plt.cm.gray)
axs[1, 1].imshow(X_train[im4,:,:,0], cmap=plt.cm.gray)

print("Normalized grayscaled images.")
plt.show()

print("Done preprocessing.")
```

```
max min  -0.488235294118 0.5
Normalized grayscaled images.
```




```
Done preprocessing.
```

Model Architecture

The python function which creates the logitis was coded with a factory pattern, a form of closure. This makes it reusable with different number of output classes.

In [5]:
```
import tensorflow as tf
from tensorflow.contrib.layers import flatten
from tensorflow.python.client import device_lib

import platform

def factory(n_classes, mu = 0, sigma = 0.1):

    def LeNet(x):

        # W=32, F=5, P=0, S=1
        # out = 1 + [W-F+2P]/S  => 1 + (32-5+0)/1 = 28
        # Input = 32x32x1. Output = 28x28x6.
        # number of filters is arbitrary
        # https://discussions.udacity.com/t/define-input-depth-outp
ut-depth-f/238575/14
        #
        conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 1
6), mean = mu, stddev = sigma))
        conv1_b = tf.Variable(tf.zeros(16))
        conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], pa
dding='VALID') + conv1_b

        # Activation.
        conv1 = tf.nn.relu(conv1)

        # out = 1 + [W-F+2P]/S  => 1+(28-2+0)/2 = 14
        # Pooling. Input = 28x28x16, Output = 14x14x16.
        conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[
1, 2, 2, 1], padding='VALID')

        # out = 1 + [W-F+2P]/S  => 1+(14-5+0)/1 = 10
        # Layer 2: Convolutional. Input = 14x14x16, Output = 10x10x
32.
        conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 16,
32), mean = mu, stddev = sigma))
        conv2_b = tf.Variable(tf.zeros(32))
        conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1]
, padding='VALID') + conv2_b

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

        # out = 1 + [W-F+2P]/S  => 1+(10-2+0)/2 = 5
        # Pooling. Input = 10x10x16, Output = 5x5x32.
```

```
        conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[
1, 2, 2, 1], padding='VALID')

        # Flatten. Input = 5x5x32. Output = 800.
        fc0   = flatten(conv2)

        # out = 1 + [W-F+2P]/S  =>
        # Fully Connected. Input = 800. Output = 400.
        fc1_W = tf.Variable(tf.truncated_normal(shape=(800, 400), m
ean = mu, stddev = sigma))
        fc1_b = tf.Variable(tf.zeros(400))
        fc1   = tf.matmul(fc0, fc1_W) + fc1_b

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

        # connected layer
        fc2_W = tf.Variable(tf.truncated_normal(shape=(400, 129), m
ean = mu, stddev = sigma))
        fc2_b = tf.Variable(tf.zeros(129))
        fc2    = tf.matmul(fc1, fc2_W) + fc2_b

        fc2    = tf.nn.relu(fc2)

        # Fully Connected. Input = 129. Output = 86.
        fc3_W  = tf.Variable(tf.truncated_normal(shape=(129, 86), m
ean = mu, stddev = sigma))
        fc3_b  = tf.Variable(tf.zeros(86))
        fc3    = tf.matmul(fc2, fc3_W) + fc3_b

        # Activation.
        fc3    = tf.nn.relu(fc3)

        # Input = 86 Output = n_classes.
        fc4_W  = tf.Variable(tf.truncated_normal(shape=(86, n_class
es), mean = mu, stddev = sigma))
        fc4_b  = tf.Variable(tf.zeros(n_classes))

        # final
        logits = tf.matmul(fc3, fc4_W) + fc4_b

        return logits
    return LeNet
```

Train, Validate and Test the Model

Input data was already split into training, validation and testing. This separation helps to prevent overfiting.

Running the model on AWS enables computations on GPU. For 50 epochs it takes about 3 minutes.

In [6]:
```
def evaluate(X_data, y_data):
    num_examples = len(X_data)
    total_accuracy = 0
    total_loss = 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]
        loss, accuracy = sess.run([loss_operation, accuracy_operati
on], feed_dict={x: batch_x, y: batch_y})
        total_loss += (loss*len(batch_x))
        total_accuracy += (accuracy * len(batch_x))
    return total_loss/num_examples, total_accuracy / num_examples
```

In [7]:
```
### Training pipeline

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

EPOCHS = 50
BATCH_SIZE = 128
#BATCH_SIZE = 256
rate = 0.0005

LeNetFn = factory(n_classes)
logits = LeNetFn(x)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_
hot_y, logits=logits)
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate = rate)
training_operation = optimizer.minimize(loss_operation)

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

print("System:  ")
print(platform.uname())
print("")
devices = [x.name for x in device_lib.list_local_devices() if x.dev
ice_type == 'GPU']
print(devices)
print("")

acc_epochs = np.zeros((EPOCHS,), np.float)
loss_fn = np.zeros((EPOCHS,), np.float)
```

```
System:
uname_result(system='Linux', node='ip-172-31-32-67', release='4.4.
0-97-generic', version='#120-Ubuntu SMP Tue Sep 19 17:28:18 UTC 20
17', machine='x86_64', processor='x86_64')

['/gpu:0']
```

In [8]:
```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    num_examples = len(X_train)
    
    print(datetime.now().isoformat(' '), " - Training...")
    print()
    for i in range(EPOCHS):
        X_train, y_train = shuffle(X_train, y_train)
        for offset in range(0, num_examples, BATCH_SIZE):
            end = offset + BATCH_SIZE
            batch_x, batch_y = X_train[offset:end], y_train[offset:
end]
            sess.run(training_operation, feed_dict={x: batch_x, y:
batch_y})
            
        loss, validation_accuracy = evaluate(X_valid, y_valid)
        print("EPOCH {} ...".format(i+1))
        print("Validation Accuracy = {:.3f}".format(validation_accu
racy))
        print()
        acc_epochs[i] = validation_accuracy
        loss_fn[i] = loss
        
    print(datetime.now().isoformat(' '), " - Finished training")
    saver.save(sess, './lenet/lenet')
    print("Model saved")
```

```
2017-10-12 14:08:35.909454  - Training...

EPOCH 1 ...
Validation Accuracy = 0.703

EPOCH 2 ...
Validation Accuracy = 0.847

EPOCH 3 ...
Validation Accuracy = 0.867

EPOCH 4 ...
Validation Accuracy = 0.892

EPOCH 5 ...
Validation Accuracy = 0.890

EPOCH 6 ...
Validation Accuracy = 0.904

EPOCH 7 ...
Validation Accuracy = 0.911

EPOCH 8 ...
Validation Accuracy = 0.914
```

```
EPOCH 9 ...
Validation Accuracy = 0.903

EPOCH 10 ...
Validation Accuracy = 0.909

EPOCH 11 ...
Validation Accuracy = 0.915

EPOCH 12 ...
Validation Accuracy = 0.902

EPOCH 13 ...
Validation Accuracy = 0.900

EPOCH 14 ...
Validation Accuracy = 0.914

EPOCH 15 ...
Validation Accuracy = 0.906

EPOCH 16 ...
Validation Accuracy = 0.917

EPOCH 17 ...
Validation Accuracy = 0.925

EPOCH 18 ...
Validation Accuracy = 0.908

EPOCH 19 ...
Validation Accuracy = 0.931

EPOCH 20 ...
Validation Accuracy = 0.932

EPOCH 21 ...
Validation Accuracy = 0.927

EPOCH 22 ...
Validation Accuracy = 0.922

EPOCH 23 ...
Validation Accuracy = 0.933

EPOCH 24 ...
Validation Accuracy = 0.929

EPOCH 25 ...
Validation Accuracy = 0.920

EPOCH 26 ...
Validation Accuracy = 0.908
```

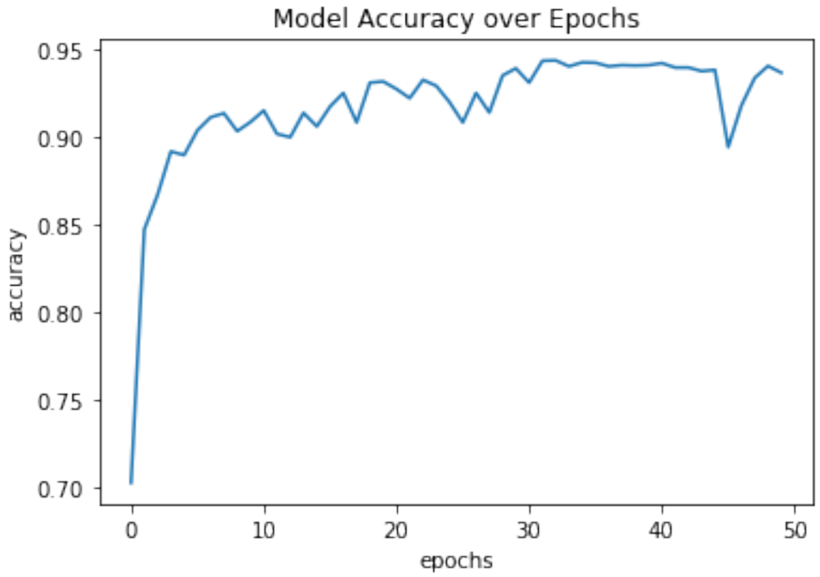
```
EPOCH 27 ...
Validation Accuracy = 0.925

EPOCH 28 ...
Validation Accuracy = 0.914

EPOCH 29 ...
Validation Accuracy = 0.935

EPOCH 30 ...
Validation Accuracy = 0.939

EPOCH 31 ...
Validation Accuracy = 0.931

EPOCH 32 ...
Validation Accuracy = 0.944

EPOCH 33 ...
Validation Accuracy = 0.944

EPOCH 34 ...
Validation Accuracy = 0.940

EPOCH 35 ...
Validation Accuracy = 0.943

EPOCH 36 ...
Validation Accuracy = 0.942

EPOCH 37 ...
Validation Accuracy = 0.940

EPOCH 38 ...
Validation Accuracy = 0.941

EPOCH 39 ...
Validation Accuracy = 0.941

EPOCH 40 ...
Validation Accuracy = 0.941

EPOCH 41 ...
Validation Accuracy = 0.942

EPOCH 42 ...
Validation Accuracy = 0.940

EPOCH 43 ...
Validation Accuracy = 0.940

EPOCH 44 ...
Validation Accuracy = 0.938
```

```
EPOCH 45 ...
Validation Accuracy = 0.938

EPOCH 46 ...
Validation Accuracy = 0.894

EPOCH 47 ...
Validation Accuracy = 0.918

EPOCH 48 ...
Validation Accuracy = 0.934

EPOCH 49 ...
Validation Accuracy = 0.941

EPOCH 50 ...
Validation Accuracy = 0.937

2017-10-12 14:11:29.978169  - Finished training
Model saved
```

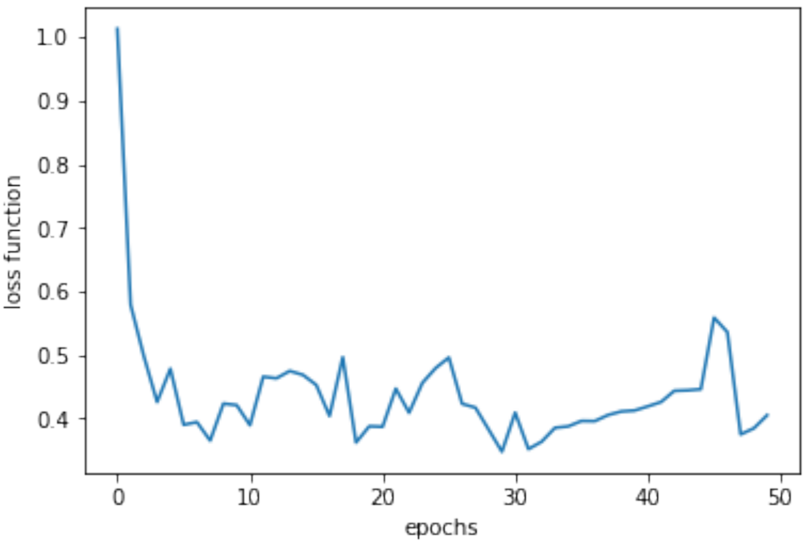
In [9]:

```
model_r_max = np.max(acc_epochs)
print("Average accuracy: ", np.mean(acc_epochs), " highest acc: ",
model_r_max)
plt.figure()
plt.plot(acc_epochs)
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.title("Model Accuracy over Epochs")
plt.show()
plt.savefig("./output/evolution_1FD32_L5_E"+str(EPOCHS)+"_B"+str(BA
TCH_SIZE)+"_R"+str(rate)+"_A94.png")

plt.figure()
plt.plot(loss_fn)
plt.xlabel("epochs")
plt.ylabel("loss function")
plt.show()
plt.savefig("./output/loss_1FD32_L5_E"+str(EPOCHS)+"_B"+str(BATCH_S
IZE)+"_R"+str(rate)+"_A94.png")
```

```
Average accuracy:  0.917446711906  highest acc:  0.943764172092
```




```
<matplotlib.figure.Figure at 0x7f2fdbd10470>
```




```
<matplotlib.figure.Figure at 0x7f2fc235ef98>
```

Evaluate

After training and validating, its time to test the model with "new" data. This data set was provided in a separated file, so the model was not exposed to it yet.

In [10]:
```
n_test = X_test0.shape[0]
X_test = np.zeros((n_test, X_test0.shape[1], X_test0.shape[2], 1), np.int)

for i in range(n_test):
    X_test[i,:,:,0] = np.dot(X_test0[i,:,:,:], lum).astype(int)

# normalized after grayscale and save computing costs
X_test = (X_test/255.0)-0.5

with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('./lenet'))

    _, test_accuracy = evaluate(X_test, y_test0)
    print("Test Accuracy = {:.3f}".format(test_accuracy))
```

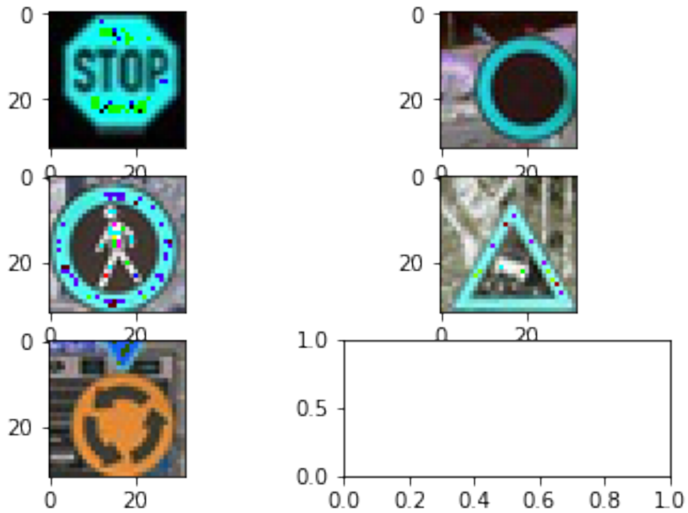
```
INFO:tensorflow:Restoring parameters from ./lenet/lenet
Test Accuracy = 0.935
```

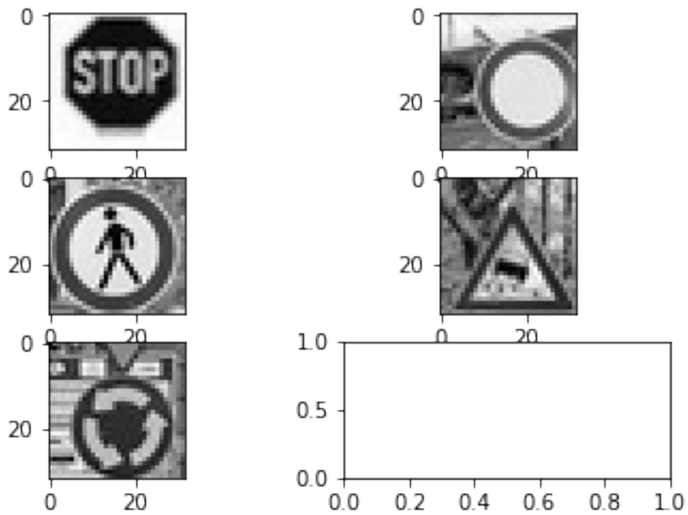
Step 3: Test a Model on New Images

Ater testing the model, let's try its accuracy on 5 new images found on internet.

Load and Output the Images

In [11]:
```
import matplotlib.image as mpimg

xtest20 = np.ndarray((5, 32, 32, 3), np.float)
xtest21 = np.ndarray((5, 32, 32, 1), np.float)
#xtest20[0, :, :, :] = mpimg.imread('./new-data/ni_01_14.png')
#xtest20[1, :, :, :] = mpimg.imread('./new-data/ni_02_15.png')
#xtest20[2, :, :, :] = mpimg.imread('./new-data/ni_03_27.png')
#xtest20[3, :, :, :] = mpimg.imread('./new-data/ni_04_30.png')
#xtest20[4, :, :, :] = mpimg.imread('./new-data/ni_05_40.png')

xtest20[0, :, :, :] = mpimg.imread('./new-data/ni_01_14.jpg')
xtest20[1, :, :, :] = mpimg.imread('./new-data/ni_02_15.jpg')
xtest20[2, :, :, :] = mpimg.imread('./new-data/ni_03_27.jpg')
xtest20[3, :, :, :] = mpimg.imread('./new-data/ni_04_30.jpg')
xtest20[4, :, :, :] = mpimg.imread('./new-data/ni_05_40.jpg')

ylabels = np.ndarray((5,), np.int, np.array([14,15,27,30,40]))
lum = np.ndarray((3,), np.float, np.array([0.299, 0.587, 0.114]))
for i in range(5):
    xtest21[i,:,:,0] = np.dot(xtest20[i,:,:,:], lum)

# normalized after grayscale and save computing costs
xtest21 = (xtest21/255.0)-0.5

fig, axs = plt.subplots(nrows=3, ncols=2)
axs[0, 0].imshow(xtest20[0,:,:,:])
axs[0, 1].imshow(xtest20[1,:,:,:])
axs[1, 0].imshow(xtest20[2,:,:,:])
axs[1, 1].imshow(xtest20[3,:,:,:])
axs[2, 0].imshow(xtest20[4,:,:,:])

plt.show()

fig, axs = plt.subplots(nrows=3, ncols=2)
axs[0, 0].imshow(xtest21[0,:,:,0], cmap=plt.cm.gray)
axs[0, 1].imshow(xtest21[1,:,:,0], cmap=plt.cm.gray)
axs[1, 0].imshow(xtest21[2,:,:,0], cmap=plt.cm.gray)
axs[1, 1].imshow(xtest21[3,:,:,0], cmap=plt.cm.gray)
axs[2, 0].imshow(xtest21[4,:,:,0], cmap=plt.cm.gray)

plt.show()
```





Predict the Sign Type for Each Image

In [16]:
```
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('./lenet'))
    r1 = sess.run(logits, feed_dict={x: xtest21})
    print("Logits")
    print(r1)
    print(r1.shape)

    print("")
    print("")
    print("softmax")
    r2 = sess.run(tf.nn.softmax(logits), feed_dict={x: xtest21})
    print(r2)
    print(r2.shape)
    for i in range(5):
        print(i, ylabels[i], r2[i, ylabels[i]])
```

```
INFO:tensorflow:Restoring parameters from ./lenet/lenet
```

```
Logits
[[ -23.82995796   12.22348881    5.24700212   -5.01921177  -12.051
56803
   -16.28949165  -26.54062653  -19.41469574  -15.23647499  -28.703
47595
   -30.18368912   -9.78508854    6.68434906    8.00753307   14.684
8774
   -13.73761368  -65.58116913  -11.43909836  -20.74489212  -43.697
57843
   -25.68615532  -35.35309982  -25.01387787  -42.83092499  -18.187
36267
    -9.23015785   -6.61341381  -49.62615204  -24.12494278  -22.123
5199
   -27.21355057  -33.24899292  -24.66412926    4.25927591  -11.667
42802
    -2.87310266  -10.11999893   -0.73470086  -10.88895035   -5.297
57071
   -11.99493122  -33.1705246   -40.28797531]
 [ -33.35050201   -4.46615314   -8.78530598    7.71556234   -6.949
57018
    -7.67655945  -47.17803955  -16.71727753  -16.28227043    0.770
72573
   -31.82898712  -34.33365631  -12.14495373   12.36383152    4.254
0307
     7.12515688  -46.0978775   -14.60758114  -17.73284531  -24.223
29712
   -26.09648514  -52.14083862    1.0644542   -42.82314682  -19.951
75362
   -11.42463303   -7.85647058  -51.46764374   -9.00272751   -8.336
936
   -54.39671707  -37.38504791  -24.6989212     1.53812647  -19.004
77219
     5.35240459   -7.40578794  -17.2496357   -18.35800743   -6.729
59661
   -23.40646362  -36.33037186  -55.51404572]
 [  -8.19537067    9.81828594  -25.48621559  -23.54010582   -4.855
68237
    -6.94281721  -17.68091583  -48.93915939  -34.76464844  -52.765
2092
   -50.41648865  -14.70374012    8.66214371  -27.15566635    6.447
21794
    -9.76159573  -59.79570007  -23.43365479   10.43781567  -56.436
71036
   -24.168396    -43.78590775  -52.15834808  -48.17084885  -38.039
3219
   -11.76746464  -14.47885036  -38.14334488  -47.70761871  -25.057
17087
   -35.11039734  -50.30174255   -8.42114544   -5.52239132  -41.710
50262
   -23.62919235  -21.55352402  -14.43875504    5.75120687  -34.407
27234
    22.58751869  -29.49789238  -41.74700546]
 [ -71.62384796  -43.7921257   -51.80265045  -22.00750351 -103.656
52466
```

```
   -17.76538467  -65.30267334  -43.8684082   -93.46403503  -34.294
49844
    -9.96207142    7.04454041  -23.77471352  -10.41671562  -49.051
77307
   -55.34400558  -59.59599686  -34.96619797  -25.52444267  -12.441
21361
    -9.41991901   -9.95913887    6.64266682    2.65481162  -15.480
34191
    25.82727432  -19.84401321  -48.59305573  -23.88368797   -1.995
40162
    16.85153961  -22.45292664  -52.59412003  -24.38441467   15.534
77764
   -14.75545406  -22.60277939  -34.56523895  -15.92673492  -75.492
72919
   -36.34951782  -34.59121704  -60.93301773]
 [ -34.50154877  -19.28138351  -18.63120842   -9.67574596  -48.081
60782
   -22.70354652  -40.78736115  -21.65718269  -38.04601288   10.550
99487
   -28.67878342  -15.69770241   16.99584007    4.14944553   -1.450
95742
    -3.30947709  -39.31541061  -15.2105341   -28.85735512  -28.477
15378
   -31.38887787  -49.80951309  -12.72123528  -19.36704826  -22.378
7117
   -23.6171627   -13.96886444  -53.98962784   -3.78920889   -8.172
67036
   -21.86774254  -58.35541534  -13.96012974   -8.60382366    8.489
84814
    14.00068855   -8.69783878  -20.01979637   -0.48454282  -34.449
13483
    -7.92018604  -15.24727726  -36.21956635]]
(5, 43)


softmax
[[  1.72576594e-17   7.84864798e-02   7.32731714e-05   2.54908761e
-09
    2.25045958e-12   3.24911234e-14   1.14750424e-18   1.42727201e
-15
    9.31288195e-14   1.31959516e-19   3.00324433e-20   2.17065480e
-11
    3.08444491e-04   1.15832000e-03   9.19946015e-01   4.16899425e
-13
    1.27251267e-35   4.15206194e-12   3.77405403e-16   4.06054904e
-26
    2.69678064e-18   1.70822124e-22   5.28217563e-18   9.65978667e
-26
    4.87001974e-15   3.78089365e-11   5.17645427e-10   1.08103062e
-28
    1.28490440e-17   9.50778403e-17   5.85471096e-19   1.40070859e
-21
    7.49386427e-18   2.72885809e-05   3.30446434e-12   2.17986038e
-08
```

```
    1.55289271e-11   1.84979854e-07   7.19763875e-12   1.92972394e
-09
    2.38159709e-12   1.51504703e-21   1.22844280e-24]
 [  1.37901412e-20   4.82927121e-08   6.42831843e-10   9.42613091e
-03
    4.03039024e-09   1.94814076e-09   1.36252795e-26   2.30826832e
-13
    3.56620793e-13   9.08300080e-06   6.31472135e-20   5.15929350e
-21
    2.23367956e-11   9.84126866e-01   2.95807724e-04   5.22304280e
-03
    4.01285870e-26   1.90333578e-12   8.36047201e-14   1.26900517e
-16
    1.94961210e-17   9.52860220e-29   1.21841103e-05   1.06084348e
-24
    9.09013482e-15   4.59042179e-11   1.62736613e-09   1.86807368e
-28
    5.17217047e-10   1.00651887e-09   9.98417054e-30   2.43998793e
-22
    7.88683097e-17   1.95662360e-05   2.34336544e-14   8.87211296e
-04
    2.55396326e-09   1.35545546e-13   4.47430416e-14   5.02204722e
-09
    2.87216723e-16   7.00531506e-22   3.26635951e-30]
 [  4.27717079e-14   2.84700809e-06   1.32384905e-21   9.26879383e
-21
    1.20660535e-12   1.49669692e-13   3.24816373e-18   8.63681426e
-32
    1.23670784e-25   1.88244069e-33   1.97132989e-32   6.37687166e
-17
    8.95947323e-07   2.49347143e-22   9.78030670e-08   8.93211913e
-15
    1.66502023e-36   1.03099161e-20   5.28989449e-06   4.78860009e
-35
    4.94494104e-21   1.49410922e-29   3.45365861e-33   1.86220134e
-31
    4.67836926e-27   1.20175499e-15   7.98503584e-17   4.21615185e
-27
    2.95941964e-31   2.03315017e-21   8.75206045e-26   2.21102112e
-32
    3.41275199e-14   6.19466175e-13   1.19047688e-28   8.47876526e
-21
    6.75746654e-20   8.31168020e-17   4.87617058e-08   1.76796462e
-25
    9.99990821e-01   2.39656606e-23   1.14780032e-28]
 [  0.00000000e+00   5.81579498e-31   1.93054881e-34   1.68093248e
-21
    0.00000000e+00   1.16917378e-19   0.00000000e+00   5.38867111e
-31
    0.00000000e+00   7.75132954e-27   2.86296104e-16   6.96133684e
-09
    2.87117652e-22   1.81704780e-16   3.02256001e-33   5.59359998e
-36
    7.96296989e-38   3.95968654e-27   4.99072789e-23   2.39959554e
```

```
-17
    4.92344624e-16   2.87136043e-16   4.65758454e-09   8.63490876e
-11
    1.14884158e-18   9.99839664e-01   1.46265598e-20   4.78178502e
-33
    2.57474683e-22   8.25459655e-13   1.26420739e-04   1.07672673e
-21
    8.74886179e-35   1.56052553e-22   3.38809732e-05   2.37178304e
-18
    9.26883745e-22   5.91282564e-27   7.35181481e-19   0.00000000e
+00
    9.92870620e-28   5.76119987e-27   2.09129012e-38]
 [  4.10227550e-23   1.67132253e-16   3.20204130e-16   2.48161994e
-12
    5.19136824e-29   5.45547948e-18   7.64066637e-26   1.55332723e
-17
    1.18490537e-24   1.51041662e-03   1.38618555e-20   6.01772524e
-15
    9.50730622e-01   2.50575840e-06   9.26221322e-09   1.44401058e
-09
    3.32960636e-25   9.79504665e-15   1.15949218e-20   1.69585225e
-20
    9.22232975e-22   9.22278498e-30   1.18058056e-13   1.53410730e
-16
    7.54931763e-18   2.18803603e-18   3.39044859e-14   1.41078647e
-31
    8.93768892e-10   1.11560969e-11   1.25839789e-17   1.79234750e
-33
    3.42018961e-14   7.24878447e-12   1.92287975e-04   4.75641415e
-02
    6.59833082e-12   7.98676300e-17   2.43457769e-08   4.32302686e
-23
    1.43603298e-11   9.44167636e-15   7.36036412e-24]]
(5, 43)
0 14 0.919946
1 15 0.00522304
2 27 4.21615e-27
3 30 0.000126421
4 40 1.43603e-11
```

Analyze Performance

In [17]:

```python
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('./lenet'))
    _, test_2_accuracy = evaluate(xtest21, ylabels)
    print("Test 2 Accuracy = {:.3f}".format(test_2_accuracy))
```

```
INFO:tensorflow:Restoring parameters from ./lenet/lenet
Test 2 Accuracy = 0.200
```

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

For each of these 5 new images, show the 5 highest probabilities.

In [18]:
```
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('./lenet'))

    prob1 = sess.run(tf.nn.top_k(logits, k=5), feed_dict={x: xtest2
1})
    print(prob1)

    prob2 = sess.run(tf.nn.top_k(tf.nn.softmax(logits), k=5), feed_
dict={x: xtest21})
    print(prob2)
```

```
INFO:tensorflow:Restoring parameters from ./lenet/lenet
TopKV2(values=array([[ 14.6848774 ,  12.22348881,   8.00753307,
6.68434906,
          5.24700212],
       [ 12.36383152,   7.71556234,   7.12515688,   5.35240459,
4.2540307 ],
       [ 22.58751869,  10.43781567,   9.81828594,   8.66214371,
          6.44721794],
       [ 25.82727432,  16.85153961,  15.53477764,   7.04454041,
          6.64266682],
       [ 16.99584007,  14.00068855,  10.55099487,   8.48984814,
          4.14944553]], dtype=float32), indices=array([[14,  1, 13
, 12,  2],
       [13,  3, 15, 35, 14],
       [40, 18,  1, 12, 14],
       [25, 30, 34, 11, 22],
       [12, 35,  9, 34, 13]], dtype=int32))
TopKV2(values=array([[  9.19946015e-01,   7.84864798e-02,   1.1583
2000e-03,
          3.08444491e-04,   7.32731714e-05],
       [  9.84126866e-01,   9.42613091e-03,   5.22304280e-03,
          8.87211296e-04,   2.95807724e-04],
       [  9.99990821e-01,   5.28989449e-06,   2.84700809e-06,
          8.95947323e-07,   9.78030670e-08],
       [  9.99839664e-01,   1.26420739e-04,   3.38809732e-05,
          6.96133684e-09,   4.65758454e-09],
       [  9.50730622e-01,   4.75641415e-02,   1.51041662e-03,
          1.92287975e-04,   2.50575840e-06]], dtype=float32), indi
ces=array([[14,  1, 13, 12,  2],
       [13,  3, 15, 35, 14],
       [40, 18,  1, 12, 14],
       [25, 30, 34, 11, 22],
       [12, 35,  9, 34, 13]], dtype=int32))
```

Model was able to predict only 1 out of 5 images, the first image with 91%. This image contains the "stop" sign, which is the 14th label.

In [ ]:

```

```