Notebook

October 13, 2017

1 Self-Driving Car Engineer Nanodegree

1.1 Deep Learning

1.2 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.

1.3 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']
```

1.4 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.

1.4.1 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])
```

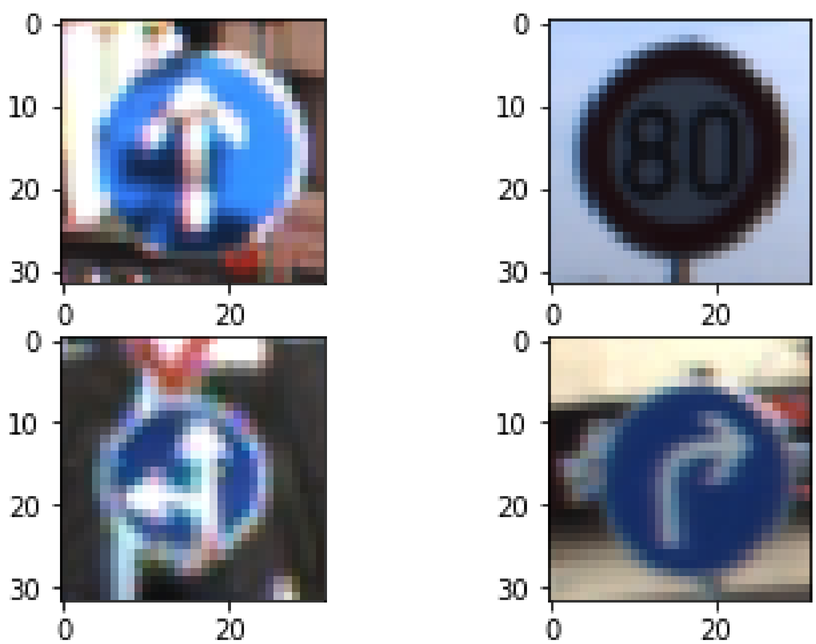
```
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
labels shape:  (34799,)
[41 41 41 41]
```

1.4.2 Visualization of the dataset

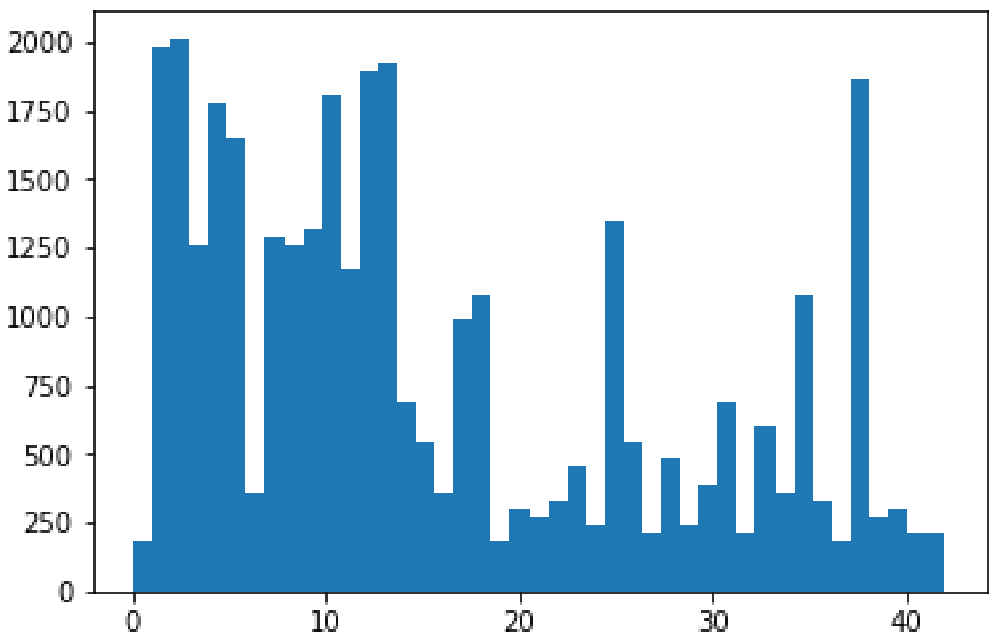
Next, it is presented 4 random images from the training dataset and a histogram of labels (output).

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)
```

```
19370 13165 4895 26662
```



```
Out[3]: (array([  180.,  1980.,  2010.,  1260.,  1770.,  1650.,   360.,  1290.,
                 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.

1.5 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.

1.5.1 Pre-process the Data Set

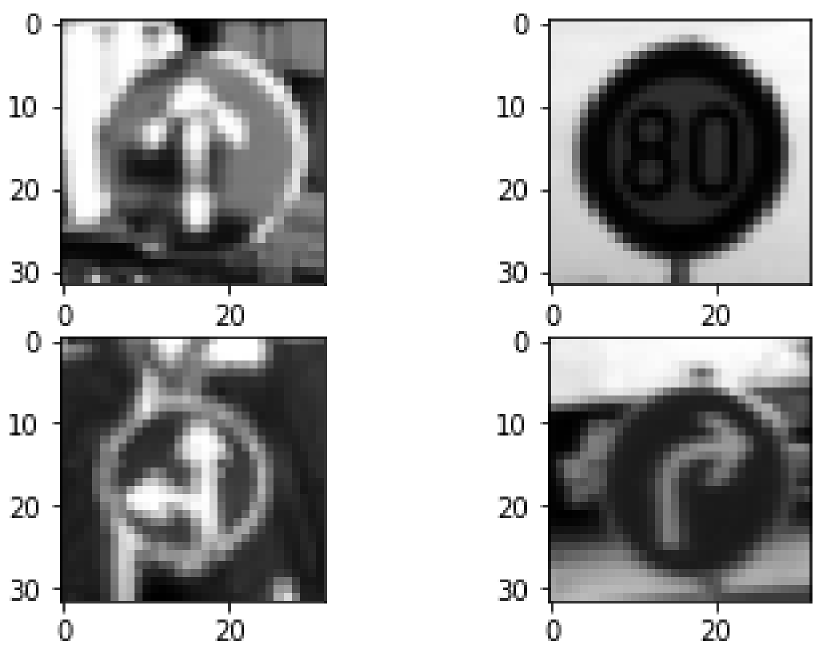
Pre-processing is made in two steps:

- 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_RGB2GRAY)
    # 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,:,
    # 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)
```

```python
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.
```

1.5.2 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-output-depth-f/238575/14
                #
                conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 16), mean = mu, stddev
                conv1_b = tf.Variable(tf.zeros(16))
                conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv

                # 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=

                # out = 1 + [W-F+2P]/S  => 1+(14-5+0)/1 = 10
                # Layer 2: Convolutional. Input = 14x14x16, Output = 10x10x32.
                conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 16, 32), mean = mu, stdde
                conv2_b = tf.Variable(tf.zeros(32))
                conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VALID') +

                # 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=
```

```
    # 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), mean = mu, stddev = si
    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), mean = mu, stddev = si
    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), mean = mu, stddev = si
    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_classes), mean = mu, stdde
    fc4_b  = tf.Variable(tf.zeros(n_classes))

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

    return logits
return LeNet
```

1.5.3 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_
            loss, accuracy = sess.run([loss_operation, accuracy_operation], feed_dict={x: ba
            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_hot_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.device_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-U

['/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)
        loss, validation_accuracy = evaluate(X_train, y_train)
        print("EPOCH {} ...".format(i+1))
        print("Validation Accuracy = {:.3f}".format(validation_accuracy))
        print()
        acc_epochs[i] = validation_accuracy
        loss_fn[i] = loss

    print(datetime.now().isoformat(' '), " - Finished training")
    saver.save(sess, './lenet/lenet')
    print("Model saved")
```

```
2017-10-13 23:16:19.360643  - Training...

EPOCH 1 ...
Validation Accuracy = 0.832

EPOCH 2 ...
Validation Accuracy = 0.912

EPOCH 3 ...
Validation Accuracy = 0.941

EPOCH 4 ...
Validation Accuracy = 0.972

EPOCH 5 ...
Validation Accuracy = 0.985

EPOCH 6 ...
Validation Accuracy = 0.987

EPOCH 7 ...
Validation Accuracy = 0.988

EPOCH 8 ...
Validation Accuracy = 0.992
```

```
EPOCH 9 ...
Validation Accuracy = 0.993

EPOCH 10 ...
Validation Accuracy = 0.994

EPOCH 11 ...
Validation Accuracy = 0.995

EPOCH 12 ...
Validation Accuracy = 0.995

EPOCH 13 ...
Validation Accuracy = 0.998

EPOCH 14 ...
Validation Accuracy = 0.996

EPOCH 15 ...
Validation Accuracy = 0.998

EPOCH 16 ...
Validation Accuracy = 0.999

EPOCH 17 ...
Validation Accuracy = 1.000

EPOCH 18 ...
Validation Accuracy = 0.996

EPOCH 19 ...
Validation Accuracy = 1.000

EPOCH 20 ...
Validation Accuracy = 0.996

EPOCH 21 ...
Validation Accuracy = 0.995

EPOCH 22 ...
Validation Accuracy = 0.999

EPOCH 23 ...
Validation Accuracy = 0.998

EPOCH 24 ...
Validation Accuracy = 0.999
```

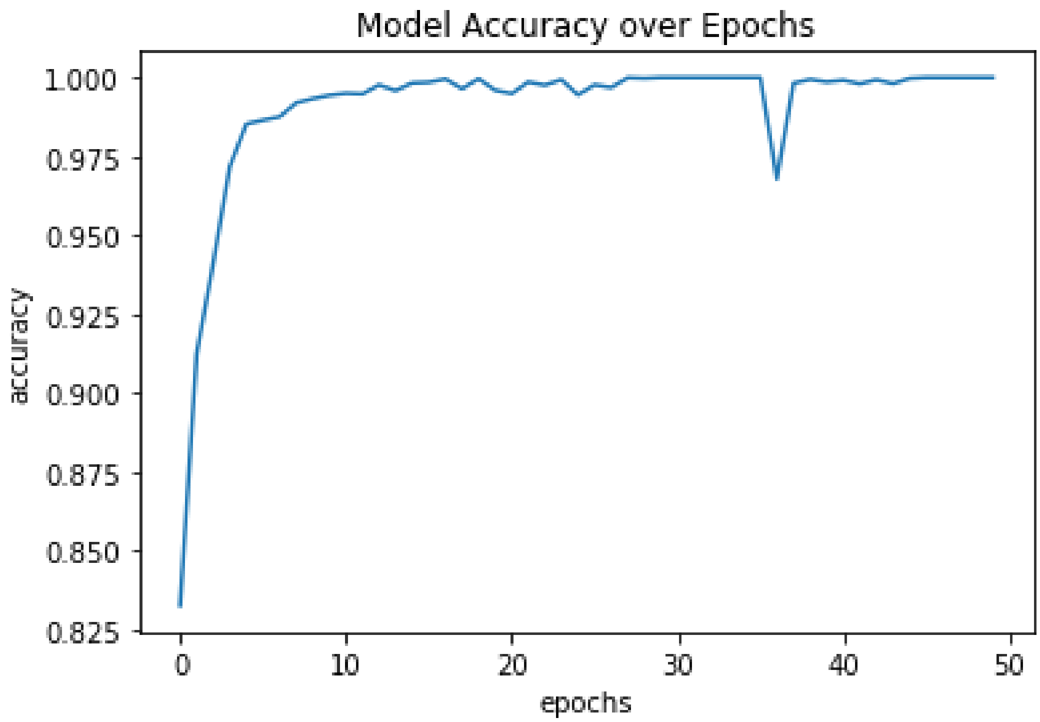
```
EPOCH 25 ...
Validation Accuracy = 0.995

EPOCH 26 ...
Validation Accuracy = 0.998

EPOCH 27 ...
Validation Accuracy = 0.997

EPOCH 28 ...
Validation Accuracy = 1.000

EPOCH 29 ...
Validation Accuracy = 1.000

EPOCH 30 ...
Validation Accuracy = 1.000

EPOCH 31 ...
Validation Accuracy = 1.000

EPOCH 32 ...
Validation Accuracy = 1.000

EPOCH 33 ...
Validation Accuracy = 1.000

EPOCH 34 ...
Validation Accuracy = 1.000

EPOCH 35 ...
Validation Accuracy = 1.000

EPOCH 36 ...
Validation Accuracy = 1.000

EPOCH 37 ...
Validation Accuracy = 0.968

EPOCH 38 ...
Validation Accuracy = 0.998

EPOCH 39 ...
Validation Accuracy = 0.999

EPOCH 40 ...
Validation Accuracy = 0.999
```

```
EPOCH 41 ...
Validation Accuracy = 0.999

EPOCH 42 ...
Validation Accuracy = 0.998

EPOCH 43 ...
Validation Accuracy = 0.999

EPOCH 44 ...
Validation Accuracy = 0.998

EPOCH 45 ...
Validation Accuracy = 1.000

EPOCH 46 ...
Validation Accuracy = 1.000

EPOCH 47 ...
Validation Accuracy = 1.000

EPOCH 48 ...
Validation Accuracy = 1.000

EPOCH 49 ...
Validation Accuracy = 1.000

EPOCH 50 ...
Validation Accuracy = 1.000

2017-10-13 23:20:04.733339  - Finished training
Model saved
```

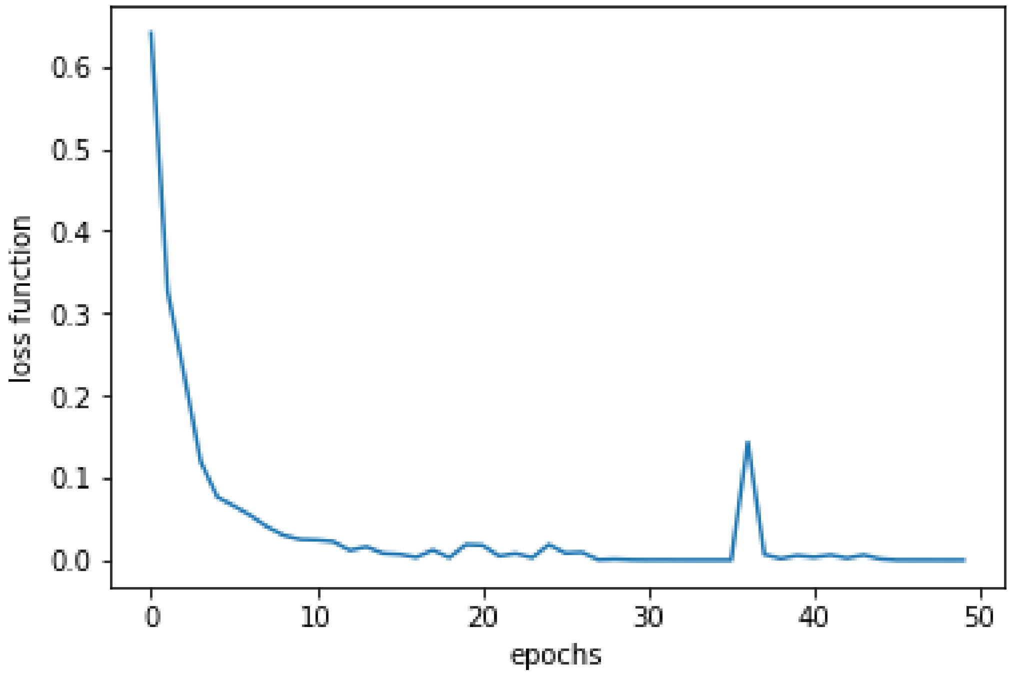
```
In [9]: model_r_max = np.max(acc_epochs)
        print("Average accuracy: ", np.mean(acc_epochs), " highest acc: ", model_r_max)
        plt.figure()
        plt.plot(acc_epochs)
        plt.xlabel("epochs")
        plt.ylabel("accuracy")
        plt.title("Model Accuracy over Epochs")
        plt.show()
        plt.savefig("./output/train_evolution_1FD32_L5_E"+str(EPOCHS)+"_B"+str(BATCH_SIZE)+"_R"+

        plt.figure()
        plt.plot(loss_fn)
        plt.xlabel("epochs")
```

```
    plt.ylabel("loss function")
    plt.show()
    plt.savefig("./output/train_loss_1FD32_L5_E"+str(EPOCHS)+"_B"+str(BATCH_SIZE)+"_R"+str(r
```

```
Average accuracy:  0.990127302522  highest acc:  1.0
```




```
&lt;matplotlib.figure.Figure at 0x7fbefa896ac8&gt;
```




```
<matplotlib.figure.Figure at 0x7fbef99e4d68>
```

```
In [11]: print("Validating: ")
         with tf.Session() as sess:
             saver.restore(sess, tf.train.latest_checkpoint('./lenet'))

             _, test_accuracy = evaluate(X_valid, y_valid)
             print("Valid Accuracy = {:.3f}".format(test_accuracy))
```

```
Validating:
INFO:tensorflow:Restoring parameters from ./lenet/lenet
Valid Accuracy = 0.953
```

2 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 [12]: 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 = 0.928
```

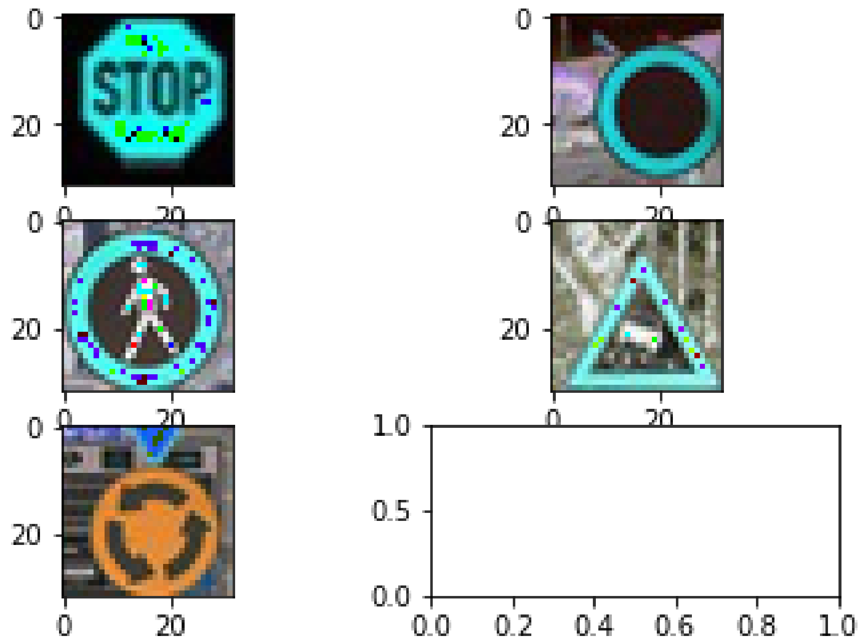
2.1 Step 3: Test a Model on New Images

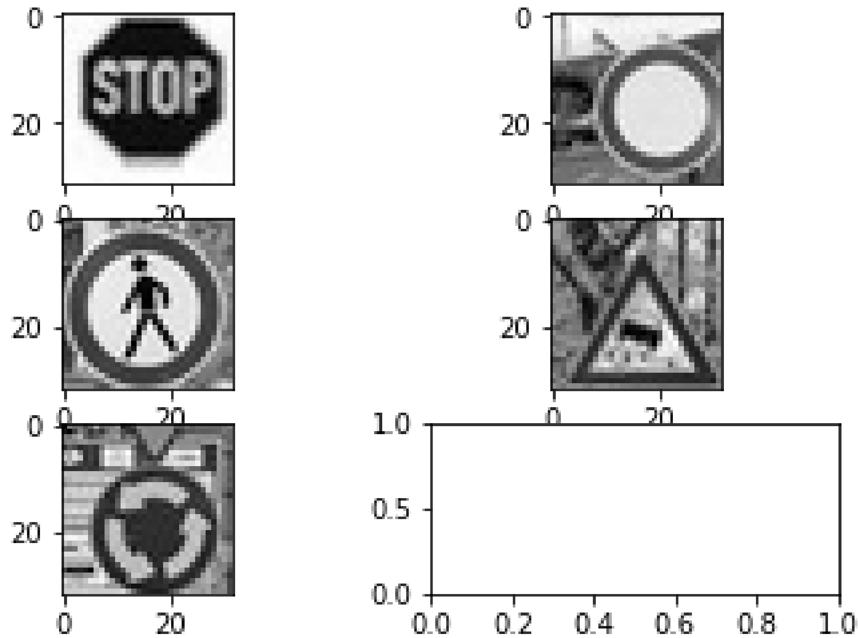
Ater testing the model, let's try its accuracy on 5 new images found on internet.

2.1.1 Load and Output the Images

```
In [13]: 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()
```





2.1.2 Predict the Sign Type for Each Image

```
In [14]: 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
[[ -9.19192505  10.0042429    1.29654014 -27.06676865  -6.24641037
  -14.05370712 -16.38176346 -24.73636436 -22.43359375 -32.57923508
  -15.50348568 -24.81461716  -0.75158906 -10.13997269  25.7023735
  -21.95749474 -63.27145004   7.78945684 -17.17030334 -41.61355209
  -12.64672089 -18.65994453 -17.57555962 -18.36603165 -30.31494141
```

```
   -4.64325762  -4.77839947 -36.7154808  -30.70487976 -10.96973801
  -13.49464512 -31.48449707 -18.63322258  -1.00251734  -6.9311986
   -9.1364069  -31.11994743   2.82251906   0.35492221 -10.02537918
   -0.45822456 -42.19474792 -25.49608612]
 [-12.22182369   2.51530051  -4.75072193  10.63594723   0.28659785
   -1.78447044 -28.05654907 -20.44342995 -24.02762222   0.56394231
  -37.52223206 -52.52075195   3.42226481  15.46515369 -28.59199524
   15.17523003 -46.20042419 -21.23786163 -19.79485321 -25.60715866
  -25.18725395 -52.93418503   4.13191986 -38.70389175 -33.02017212
  -22.57774544 -13.97543812 -43.82376099   0.4538973   -1.18482828
  -51.71731186 -40.26700974  -4.14393806  -0.88707876 -18.05291367
   16.77182198   4.63538361 -31.63377762 -14.9858551   -5.41299486
  -24.13656998 -20.73956871 -31.96615791]
 [ -5.69143295  29.33576584 -25.91496468 -54.19385529  10.3898983
    1.55502999 -10.21845627 -37.54892731 -54.8032074  -63.20101166
  -41.95126724  -7.24624586  19.18460464 -33.37414551   0.77267075
  -43.89196014 -70.97246552 -32.22324371  -1.09532821 -74.30755615
  -24.39016533 -43.32471848 -73.56188965 -60.44042587 -54.45734787
  -25.36089325   1.46401811 -14.94841862 -44.72235489 -39.66427612
  -48.85358429 -71.33682251   2.0989511  -31.49140358 -52.69199371
  -35.23439407 -41.98279572  10.47777081  -8.83584881 -33.73968887
   25.44639397 -24.49795914 -10.36846542]
 [-71.251091   -46.04450989 -15.90805054 -24.22593498 -67.97364807
   -7.32366276 -53.60364532 -34.92037201 -28.3302784  -21.54215813
  -10.39643478 -11.7344017  -63.26524353 -19.44701004 -13.43717194
  -46.24608231 -68.54214478 -27.11673737 -24.18252754  13.93457222
  -16.74518204  26.81198692  15.78407001  16.0741806   11.83925819
   38.60680771  -8.64576435 -19.98763084   7.22319984  18.42742538
    4.82588673  12.49958992 -68.86722565 -34.24888229 -12.9064436
  -33.37695312 -39.51373291 -22.86864471 -15.9846859  -16.82720566
  -18.65836716 -58.74277878 -53.68140411]
 [-19.70772552  -1.82331288 -14.92904282  -3.113343   -27.75069046
   -4.19802427 -17.84126663   2.98316622 -21.2712326    5.44001627
  -15.4081974  -18.75189972   7.32501936   1.30465043 -13.48704433
    0.33735278  -7.08003521  -5.87187576 -33.56417847 -20.77654266
  -30.19247055 -32.51596832 -34.6438446  -20.43834114 -34.10746384
  -34.59411621 -11.5264101  -47.34443283  -2.80893636 -18.365242
  -18.0318718  -50.21797562   1.61104047  -3.836303    -6.25076246
   -0.88979149 -28.62583733 -16.61203575 -15.19626522 -27.6097641
    7.95799875 -12.09927273  -9.39220428]]
(5, 43)


softmax
[[  7.00807267e-16   1.52190822e-07   2.51582730e-11   1.20963048e-23
    1.33296694e-14   5.42192490e-18   5.28556572e-19   1.24375622e-22
    1.24398462e-21   4.88223796e-26   1.27210307e-18   1.15013724e-22
    3.24481609e-12   2.71560616e-16   9.99999881e-01   2.00254812e-21
```

```
  0.00000000e+00  1.66157239e-08  2.40233154e-19  5.82190228e-30
  2.21411335e-17  5.41613147e-20  1.60189200e-19  7.26665890e-20
  4.69882385e-25  6.62308004e-14  5.78586116e-14  7.80315554e-28
  3.18156724e-25  1.18441596e-16  9.48312528e-18  1.45900786e-25
  5.56283292e-20  2.52471915e-12  6.72078292e-15  7.40815863e-16
  2.10077875e-25  1.15719226e-10  9.81163501e-12  3.04532829e-16
  4.35107202e-12  3.25576576e-30  5.81823793e-23]
[ 1.73494777e-13  4.36051550e-07  3.04750086e-10  1.46652607e-03
  4.69488057e-08  5.91794924e-09  2.30331301e-20  4.66323964e-17
  1.29447004e-18  6.19545872e-08  1.78426374e-24  5.46618932e-31
  1.08000904e-06  1.83479473e-01  1.34837799e-20  1.37301475e-01
  3.03786247e-28  2.10703267e-17  8.91993853e-17  2.66752228e-19
  4.05948708e-19  3.61519126e-31  2.19597086e-06  5.47355034e-25
  1.60945510e-22  5.51780423e-18  3.00400370e-14  3.27144662e-27
  5.54985640e-08  1.07793445e-08  1.22071016e-30  1.14660970e-25
  5.59071178e-10  1.45178722e-08  5.09184274e-16  6.77744985e-01
  3.63310619e-06  6.43844037e-22  1.09366014e-14  1.57153110e-10
  1.16085131e-18  3.46797788e-17  4.61775193e-22]
[ 6.01269302e-16  9.79914486e-01  9.91060056e-25  5.18485095e-37
  5.79565373e-09  8.43661512e-13  6.50141338e-18  8.78085252e-30
  2.81899773e-37  0.00000000e+00  1.07553137e-31  1.27005391e-16
  3.82467806e-05  5.70978477e-28  3.85827898e-13  1.54451626e-32
  0.00000000e+00  1.80488698e-27  5.95843633e-14  0.00000000e+00
  4.55316631e-24  2.72358397e-32  0.00000000e+00  0.00000000e+00
  3.98383459e-37  1.72476903e-24  7.70267910e-13  5.73867536e-20
  6.73216929e-33  1.05888550e-30  1.08140039e-34  0.00000000e+00
  1.45341595e-12  3.75217684e-27  2.32801889e-36  8.88636294e-29
  1.04215450e-31  6.32798480e-09  2.59099239e-17  3.96156014e-28
  2.00472791e-02  4.08788352e-24  5.59576683e-18]
[ 0.00000000e+00  1.72343523e-37  2.11103123e-24  5.15328490e-28
  0.00000000e+00  1.12887867e-20  0.00000000e+00  1.16828820e-32
  8.50332527e-30  7.54455399e-27  5.22587356e-22  1.37115713e-22
  0.00000000e+00  6.13117918e-26  2.49794888e-23  1.40881783e-37
  0.00000000e+00  2.86170891e-29  5.38188075e-28  1.92744067e-11
  9.13972273e-25  7.54346956e-06  1.22520091e-10  1.63757466e-10
  2.37136114e-12  9.99992490e-01  3.00930246e-21  3.57072373e-26
  2.34568140e-14  1.72267089e-09  2.13367726e-15  4.58961385e-12
  0.00000000e+00  2.28650740e-32  4.24693429e-23  5.46820694e-32
  1.18215759e-34  2.00238224e-27  1.95529884e-24  8.41997097e-25
  1.34910980e-25  0.00000000e+00  0.00000000e+00]
[ 5.95376287e-13  3.48251415e-05  7.08185524e-11  9.58606142e-06
  1.91326624e-16  3.24017878e-06  3.84932277e-12  4.25912486e-03
  1.24672028e-13  4.96954732e-02  4.38584100e-11  1.54846513e-12
  3.27313006e-01  7.94969033e-04  2.99501257e-10  3.02174769e-04
  1.81521202e-07  6.07609707e-07  5.71491330e-19  2.04460838e-13
  1.66465317e-17  1.63020238e-18  1.94140699e-19  2.86740602e-13
  3.31943237e-19  2.04039078e-19  2.12760609e-09  5.91999286e-25
  1.29969685e-05  2.27942188e-12  3.18130678e-12  3.34470214e-26
```

```
    1.07997516e-03   4.65224366e-06   4.15983521e-07   8.85760819e-05
    7.97452403e-17   1.31592914e-11   5.42117497e-11   2.20281969e-16
    6.16400123e-01   1.19977606e-09   1.79789978e-08]]
(5, 43)
0 14 1.0
1 15 0.137301
2 27 5.73868e-20
3 30 2.13368e-15
4 40 0.6164
```

2.1.3 Analyze Performance

```
In [15]: 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 = 0.400
```

2.1.4 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 [16]: 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: xtest21})
             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([[ 25.7023735 ,  10.0042429 ,   7.78945684,   2.82251906,
          1.29654014],
       [ 16.77182198,  15.46515369,  15.17523003,  10.63594723,
          4.63538361],
       [ 29.33576584,  25.44639397,  19.18460464,  10.47777081,  10.3898983 ],
       [ 38.60680771,  26.81198692,  18.42742538,  16.0741806 ,
         15.78407001],
       [  7.95799875,   7.32501936,   5.44001627,   2.98316622,
          1.61104047]], dtype=float32), indices=array([[14,  1, 17, 37,  2],
       [35, 13, 15,  3, 36],
       [ 1, 40, 12, 37,  4],
       [25, 21, 29, 23, 22],
```

```
       [40, 12,  9,  7, 32]], dtype=int32))
TopKV2(values=array([[  9.99999881e-01,   1.52190822e-07,   1.66157239e-08,
          1.15719226e-10,   2.51582730e-11],
       [  6.77744985e-01,   1.83479473e-01,   1.37301475e-01,
          1.46652607e-03,   3.63310619e-06],
       [  9.79914486e-01,   2.00472791e-02,   3.82467806e-05,
          6.32798480e-09,   5.79565373e-09],
       [  9.99992490e-01,   7.54346956e-06,   1.72267089e-09,
          1.63757466e-10,   1.22520091e-10],
       [  6.16400123e-01,   3.27313006e-01,   4.96954732e-02,
          4.25912486e-03,   1.07997516e-03]], dtype=float32), indices=array([[14,  1, 17, 37,  2
       [35, 13, 15,  3, 36],
       [ 1, 40, 12, 37,  4],
       [25, 21, 29, 23, 22],
       [40, 12,  9,  7, 32]], dtype=int32))
```

Model was able to predict only 2 out of 5 images, the first image with 100%. This first image contains the "stop" sign, which is the 14th label. The second was label 40th, Roundabout with 61%.

```
In [ ]:
```