CSCI677 Homework6

1 Introduction:

This is a programming assignment to create, train, and test a CNN for the task of semantic segmentation following the FCN32s. The network of this task is similar to VGG16. The dataset we use for this assignment is based on KITTI Road semantic segmentation dataset. Each image is a 352x1216 RGB image. In the folder named label, each file stores a (352,1216) numpy array In this dataset, 0 for Non-Road , 1 for Road , and -1 for Void.

2 Data Processing:

First of all, we need to do the data processing. This is a way to import data into our program.

```
def imageprocessing(path):
    img = utils.load_image(path)
    batch = img.reshape((1, 352, 1216, 3))
    return batch

def gtprocessing(path):
    gt = pickle.load(open(path, encoding='latin1'))
    return gt
# data Path
test_image_Path =
"C:/Users/yuhou/OneDrive/CSCI677/homework6/HW6_Dataset/image/test"
train_image_Path =
"C:/Users/yuhou/OneDrive/CSCI677/homework6/HW6_Dataset/image/train"
test_gt_Path =
"C:/Users/yuhou/OneDrive/CSCI677/homework6/HW6_Dataset/label/test"
train_gt_Path =
"C:/Users/yuhou/OneDrive/CSCI677/homework6/HW6_Dataset/label/train"
# test_image_files = glob.glob(test_image_Path+"\*.png")
# data list
test_image_list = os.listdir(test_image_Path)
train_image_list = os.listdir(train_image_Path)
test_gt_list = os.listdir(test_gt_Path)
train_gt_list = os.listdir(train_gt_Path)
# data
test_images = []
train_images = []
test_labels = []
train_labels = []
for path in (test_image_list):
    test_images.append(imageprocessing(test_image_Path+"/"+path))
for path in (test_gt_list):
    test_labels.append(gtprocessing(test_gt_Path + "/" + path))
for path in (train_image_list):
    train_images.append(imageprocessing(train_image_Path+"/"+path))
for path in (train_gt_list):
    train_labels.append(gtprocessing(train_gt_Path + "/" + path))
validation_images= train_images[199:]
train_images = train_images[0:199]
```

Print the dataset:

```
print(train_images.shape)
print(validation_images.shape)
print(test_images.shape)
```

```
print(train_labels.shape)
print(validation_labels.shape)
print(test_labels.shape)

Print:
(199, 352, 1216, 3)
(45, 352, 1216, 3)
(45, 352, 1216, 3)
(199, 352, 1216)
(45, 352, 1216)
(45, 352, 1216)
```

3 Architecture:

Construct a Fully Convolutional Network FCN-32s. FCN-32s network is based on VGG-16 network with some modifications:
Here are the codes for the architecture of FCN-32s. You can see that we changed the fully connected layers into fully convolutional network.

```
self.conv1_1 = self.conv_layer(bgr, 3, 64, "conv1_1")
self.conv1_2 = self.conv_layer(self.conv1_1, 64, 64, "conv1_2")
self.pool1 = self.max_pool(self.conv1_2, 'pool1')

self.conv2_1 = self.conv_layer(self.pool1, 64, 128, "conv2_1")
self.conv2_2 = self.conv_layer(self.conv2_1, 128, 128, "conv2_2")
self.pool2 = self.max_pool(self.conv2_2, 'pool2')

self.conv3_1 = self.conv_layer(self.pool2, 128, 256, "conv3_1")
self.conv3_2 = self.conv_layer(self.conv3_1, 256, 256, "conv3_2")
self.conv3_3 = self.conv_layer(self.conv3_2, 256, 256, "conv3_3")
self.pool3 = self.max_pool(self.conv3_3, 'pool3')

self.conv4_1 = self.conv_layer(self.pool3, 256, 512, "conv4_1")
self.conv4_2 = self.conv_layer(self.conv4_1, 512, 512, "conv4_2")
self.conv4_3 = self.conv_layer(self.conv4_2, 512, 512, "conv4_3")
self.pool4 = self.max_pool(self.conv4_3, 'pool4')

self.conv5_1 = self.conv_layer(self.pool4, 512, 512, "conv5_1")
self.conv5_2 = self.conv_layer(self.conv5_1, 512, 512, "conv5_2")
self.conv5_3 = self.conv_layer(self.conv5_2, 512, 512, "conv5_3")
self.pool5 = self.max_pool(self.conv5_3, 'pool5')

self.conv6 = self.conv_layer(self.pool5, 512, 4096, "conv6")
self.conv7 = self.conv_layer(self.conv6, 4096, 4096, "conv7")
self.conv8 = self.conv_layer(self.conv7, 4096, 1, "conv8")

self.conv9 = tf.layers.conv2d_transpose(self.conv8, strides=(32,32),
kernel_size = 64, filters =1, padding ='same' )

self.prob = tf.nn.sigmoid(self.conv9, name="prob")

self.data_dict = None
```

4 Training and Validation:

For this part, we should configure the cross entropy (loss function). Shown as below:

```
# setup::::
Entropy = tf.nn.sigmoid_cross_entropy_with_logits(logits=fcn.conv9,
labels=true_out)
mask = tf.minimum(true_out,0)+1
costEntropy = tf.reduce_sum(Entropy*mask)/tf.reduce_sum(mask)
train =
tf.train.GradientDescentOptimizer(0.0001).minimize(costEntropy)
```

Then we can create a session and run it. Shown as below:

```
for epk in range(EPOCH):

    # simple training/// many steps:
    print ("@@@@@@@@@@@@@@@@@@@@@@@@@@@@ start training EPOCH =", epk)
    lossFTrain = []
    for i in range(len(train_images)):
    # for i in range(3):
        if i % 50 ==0:
            print ("EPOCH",epk, "Train-",i)
        # get data
        img1 = train_images[i:i+1]
        img1_true_result = np.expand_dims(train_labels[i:i+1], -1)

        # train
        sess.run(train, feed_dict={images: img1, true_out:
img1_true_result, train_mode: True})

        # get entropy
        lossF = sess.run(costEntropy, feed_dict={images: img1,
true_out: img1_true_result, train_mode: False})


        lossFTrain.append(lossF)

    lossTall = sum(lossFTrain)/len(lossFTrain)
    T.append(lossTall)
    print("*****lossFall",lossTall)

    print ("@@@@@@@@@@@@@@@@@@@@@@@@@@@@ start validation EPOCH = ",
epk)


    # validatiion step:
    lossFValidation = []
    for i in range(len(validation_images)):
    # for i in range(3):
        if i % 20 ==0:
            print("EPOCH", epk, "Validation-", i)
        # get data
        img1 = validation_images[i:i + 1]
        img1_true_result = np.expand_dims(validation_labels[i:i + 1],
```

```
-1)

        # get prob and entropy
        prob = sess.run(fcn.prob, feed_dict={images: img1, true_out:
img1_true_result, train_mode: False})
        lossF = sess.run(costEntropy, feed_dict={images:
img1,true_out: img1_true_result, train_mode: False})

        lossFValidation.append(lossF)

    lossVall = sum(lossFValidation)/len(lossFValidation)
    V.append(lossVall)
    print("******lossVall",lossVall)

    # test_save

fcn.save_npy(sess,"C:/Users/yuhou/OneDrive/CSCI677/homework6/hw6/test-
save-"+str(epk)+".npy")
```

You can see after each epoch, I saved all parameters (weight we have learned). The training process is very slow, so we should save the weight as soon as possible in case the process crash.

```
(env) yuhou@yuhou:/data/home/yuhou/hw6$ ls
HW6_Dataset      test-save-11.npy  test-save-18.npy  test-save-24.npy  test-save-30.npy  test-save-37.npy  test-save-43.npy  test-save-4.npy
hw6.py           test-save-12.npy  test-save-19.npy  test-save-25.npy  test-save-31.npy  test-save-38.npy  test-save-44.npy  test-save-5.npy
lossCurve.png    test-save-13.npy  test-save-1.npy   test-save-26.npy  test-save-32.npy  test-save-39.npy  test-save-45.npy  test-save-6.npy
prob.npy         test-save-14.npy  test-save-20.npy  test-save-27.npy  test-save-33.npy  test-save-3.npy   test-save-46.npy  test-save-7.npy
__pycache__      test-save-15.npy  test-save-21.npy  test-save-28.npy  test-save-34.npy  test-save-40.npy  test-save-47.npy  test-save-8.npy
test-save-0.npy  test-save-16.npy  test-save-22.npy  test-save-29.npy  test-save-35.npy  test-save-41.npy  test-save-48.npy  test-save-9.npy
test-save-10.npy test-save-17.npy  test-save-23.npy  test-save-2.npy   test-save-36.npy  test-save-42.npy  test-save-49.npy  test-save.npy
(env) yuhou@yuhou:/data/home/yuhou/hw6$
```

Figure 1

5 Result:

1- Loss Function Curve

I tried twice. For every training, I used learning rate as 0.0001 and 0.05. Only one of them got converged. I noticed that the instruction said the recommended learning rate should be 0.001, but I already started training, due to the long training time, I didn't stop it, but in order to see the influence of different learning rate, I tried 0.05 for the second time. I did not try a lot of times to train the network, because I did not use GPU on Azure and on my laptop. This also have an influence on my result of IoU and the visualization. The pictures below are my loss function curves. The epochs of them are both 50. The red line is training, and the blue line is validation.

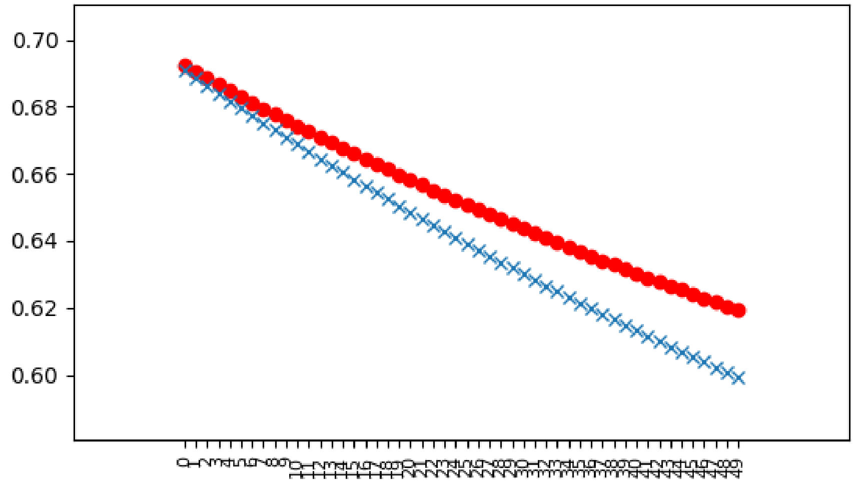


Figure 2 Learning rate as 0.001

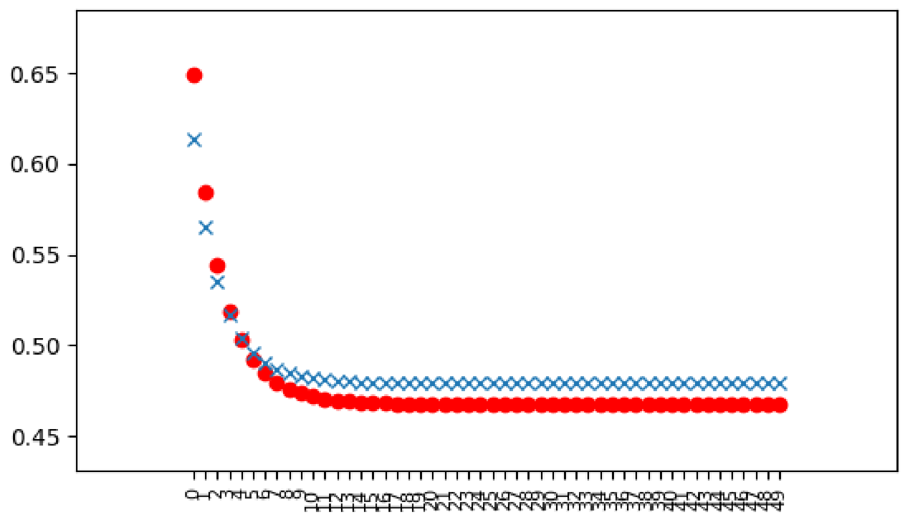


Figure 3 Learning rate as 0.05

2- IoU

We are going to use pixel-level intersection-over-union (IoU) metrics to evaluate the network output. Pixel-level IoU = TP/(TP+FP+FN), where TP, FP, and FN are the numbers of true positive, false positive, and false negative pixels, respectively. For the whole testing data. The overall IoU is 12.637%. This IoU is very low. I think there are something wrong with my code, the network learning nothing from the training set.

6 Effects of parameter choices:

Due to the long duration of training, we just tested two sessions. We tried learning rate as 0.001 and 0.05 to see the difference. We can see if learning rate is 0.05 (higher), the loss function will be smaller than the learning rate as 0.001. Also, the loss function can drop down soon. Shown as figure 4 and figure 5. They are the same to the figure 2 and figure 3.

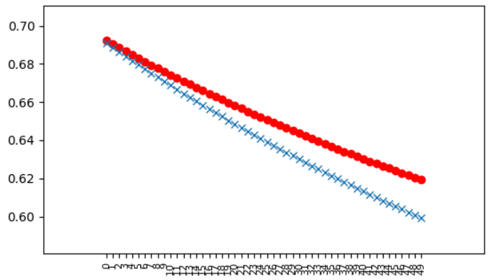


Figure 4 Learning rate as 0.001

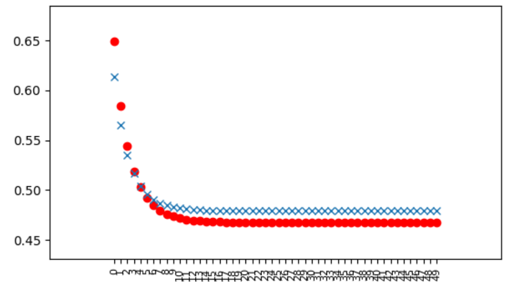


Figure 5 Learning rate as 0.05

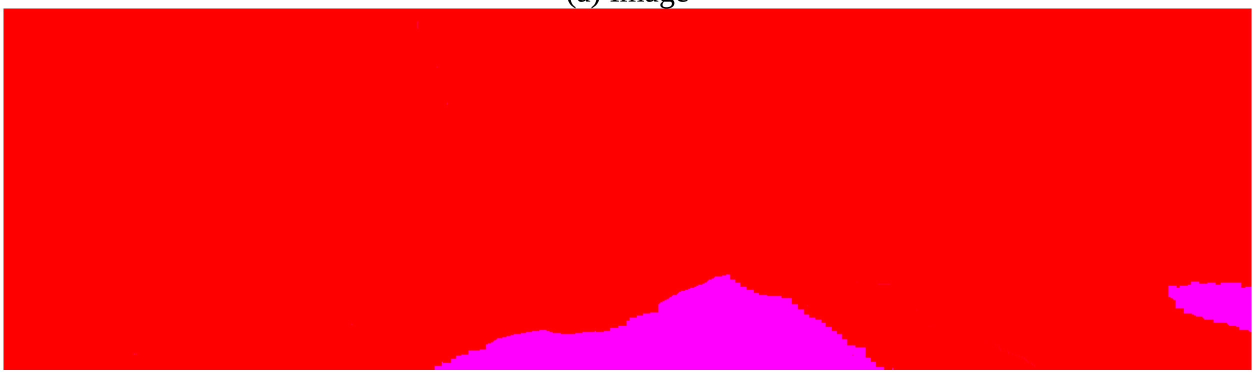
7 Visualization and Discussion:

In order to make sure we can get the result before the training crash. We saved the sigmoid function into a ".npy" file. We can use this file to visualize result.

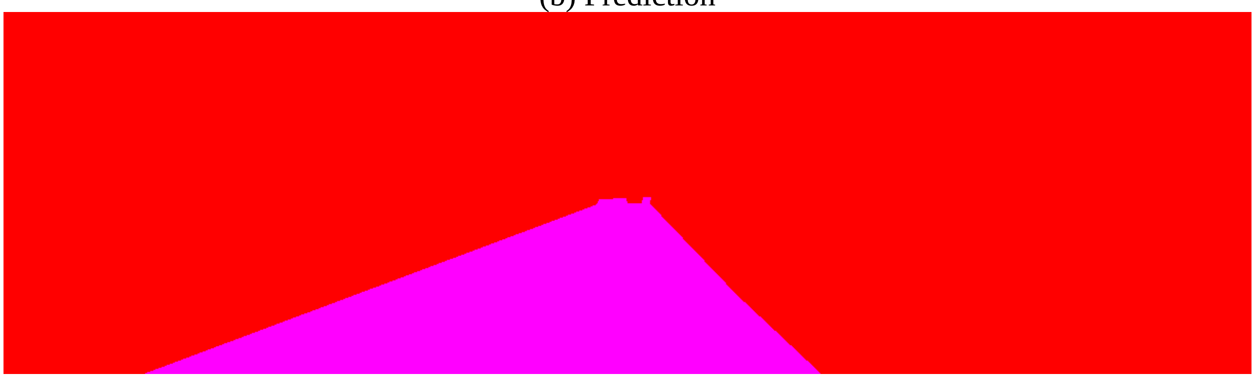
```
# visualization of data
prob = np.load("./prob.npy")

# which image you want to check
number = 15
print(prob.shape)   #(45, 352, 1216, 1)
x = prob[15][0]

# This is for B chanel, cuz R and G chanel are the same
x[x>0.5] = 255
x[x<=0.5] = 0
x = np.squeeze(x, axis=2)
print(x.shape)
# This is for R chanel
R = np.zeros([352,1216],dtype=np.uint8)
R.fill(255) # or img[:] = 255
print(R.shape)
# This is for G chanel
G = np.zeros([352,1216],dtype=np.uint8)
G.fill(0) # or img[:] = 255
print(G.shape)

# Create the image
img = [x,G,R]
img = np.array(img)
img = np.swapaxes(np.swapaxes(img,0,2),0,1)# shape (3,352,1216)->>
(352, 1216, 3)
print(img.shape)
```

However, in this homework, we did not get a good result. I just present two failure examples. It seems that few things the code have learned from the training, so it predicted a bad result from the testing data.

(a) Image



(b) Prediction

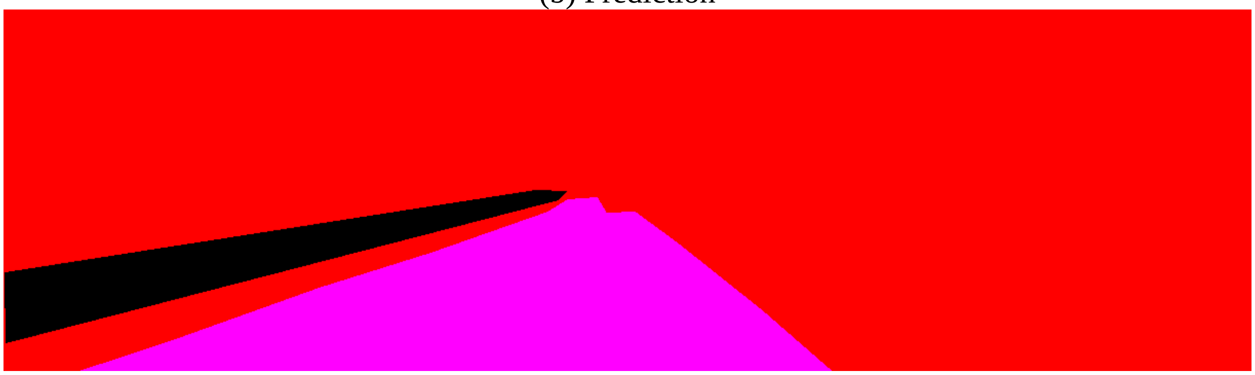


(c) Ground Truth

Figure 6 uu_000003.png (IoU = 25.295%)

(a) Image

(b) Prediction



(c) Ground Truth

Figure 7 nm_000014.png (IoU = 19.436%)

Other results are even worse. They are all red like figure 8.

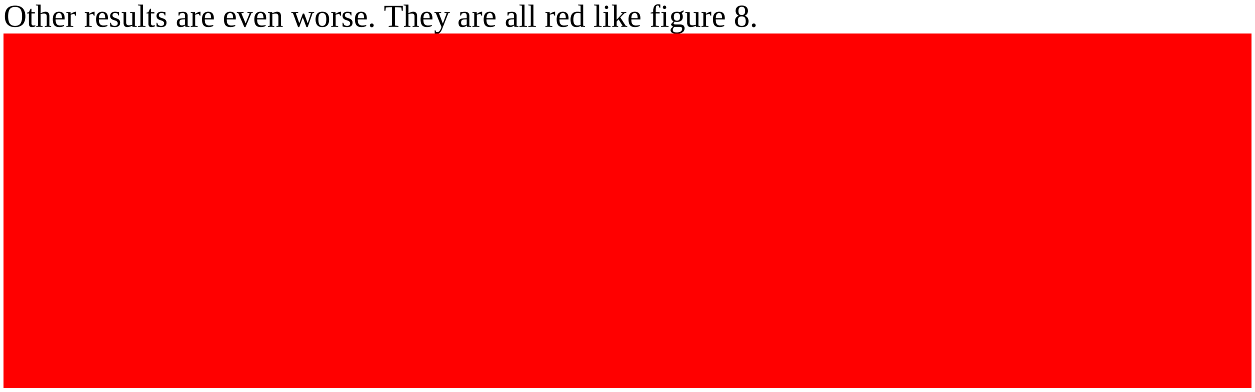


Figure 8 failure examples

There are some reasons I got the bad results in this homework. I think the main reason is that the training is not enough. For the first session, it even did not converge. On the other hand, the second session got converged, but I guess it falls into a local optimization.

8 Appendix (code):

1-util.py

```
import skimage
import skimage.io
```

```python
import skimage.transform
import numpy as np

# [height, width, depth]
def load_image(path):
    # load image
    img = skimage.io.imread(path)
    img = img / 255.0
    assert (0 <= img).all() and (img <= 1.0).all()
    # print "Original Image Shape: ", img.shape
    # we crop image from center
    short_edge = min(img.shape[:2])
    yy = int((img.shape[0] - short_edge) / 2)
    xx = int((img.shape[1] - short_edge) / 2)
    crop_img = img[yy: yy + short_edge, xx: xx + short_edge]
    resized_img = skimage.transform.resize(crop_img, (352, 1216))
    return resized_img

if __name__ == "__main__":
    test()
```

2- fcn32_trainable.py

```python
import tensorflow as tf

import numpy as np
from functools import reduce

FCN_MEAN = [103.939, 116.779, 123.68]


class Fcn32:

    def __init__(self, fcn32_npy_path=None, trainable=True,
dropout=0.5):
        if fcn32_npy_path is not None:
            self.data_dict = np.load(fcn32_npy_path,
encoding='latin1').item()
        else:
            self.data_dict = None

        self.var_dict = {}
        self.trainable = trainable
        self.dropout = dropout

    def build(self, rgb, train_mode=None):
        """

        :param rgb: rgb image [batch, height, width, 3] values scaled
[0, 1]
        :param train_mode: a bool tensor, usually a placeholder: if
True, dropout will be turned on
```

```
        """

        rgb_scaled = rgb * 255.0

        # Convert RGB to BGR
        red, green, blue = tf.split(axis=3, num_or_size_splits=3,
value=rgb_scaled)
        assert red.get_shape().as_list()[1:] == [352, 1216, 1]
        assert green.get_shape().as_list()[1:] == [352, 1216, 1]
        assert blue.get_shape().as_list()[1:] == [352, 1216, 1]
        bgr = tf.concat(axis=3, values=[
            blue - FCN_MEAN[0],
            green - FCN_MEAN[1],
            red - FCN_MEAN[2],
        ])
        assert bgr.get_shape().as_list()[1:] == [352, 1216, 3]

        self.conv1_1 = self.conv_layer(bgr, 3, 64, "conv1_1")
        self.conv1_2 = self.conv_layer(self.conv1_1, 64, 64,
"conv1_2")
        self.pool1 = self.max_pool(self.conv1_2, 'pool1')

        self.conv2_1 = self.conv_layer(self.pool1, 64, 128, "conv2_1")
        self.conv2_2 = self.conv_layer(self.conv2_1, 128, 128,
"conv2_2")
        self.pool2 = self.max_pool(self.conv2_2, 'pool2')

        self.conv3_1 = self.conv_layer(self.pool2, 128, 256,
"conv3_1")
        self.conv3_2 = self.conv_layer(self.conv3_1, 256, 256,
"conv3_2")
        self.conv3_3 = self.conv_layer(self.conv3_2, 256, 256,
"conv3_3")
        self.pool3 = self.max_pool(self.conv3_3, 'pool3')

        self.conv4_1 = self.conv_layer(self.pool3, 256, 512,
"conv4_1")
        self.conv4_2 = self.conv_layer(self.conv4_1, 512, 512,
"conv4_2")
        self.conv4_3 = self.conv_layer(self.conv4_2, 512, 512,
"conv4_3")
        self.pool4 = self.max_pool(self.conv4_3, 'pool4')

        self.conv5_1 = self.conv_layer(self.pool4, 512, 512,
"conv5_1")
        self.conv5_2 = self.conv_layer(self.conv5_1, 512, 512,
"conv5_2")
        self.conv5_3 = self.conv_layer(self.conv5_2, 512, 512,
"conv5_3")
        self.pool5 = self.max_pool(self.conv5_3, 'pool5')

        self.conv6 = self.conv_layer(self.pool5, 512, 4096, "conv6")
```

```
        self.conv7 = self.conv_layer(self.conv6, 4096, 4096, "conv7")
        self.conv8 = self.conv_layer(self.conv7, 4096, 1, "conv8")

        self.conv9 = tf.layers.conv2d_transpose(self.conv8,
strides=(32,32), kernel_size = 64, filters =1, padding ='same' )

        # print(self.conv8.shape)
        # self.prob = tf.nn.softmax(self.conv9, name="prob")
        self.prob = tf.nn.sigmoid(self.conv9, name="prob")
        # print(self.prob.shape)

        self.data_dict = None

    def avg_pool(self, bottom, name):
        return tf.nn.avg_pool(bottom, ksize=[1, 2, 2, 1], strides=[1,
2, 2, 1], padding='SAME', name=name)

    def max_pool(self, bottom, name):
        return tf.nn.max_pool(bottom, ksize=[1, 2, 2, 1], strides=[1,
2, 2, 1], padding='SAME', name=name)

    def conv_layer(self, bottom, in_channels, out_channels, name):
        with tf.variable_scope(name):
            filt, conv_biases = self.get_conv_var(3, in_channels,
out_channels, name)

            conv = tf.nn.conv2d(bottom, filt, [1, 1, 1, 1],
padding='SAME')
            bias = tf.nn.bias_add(conv, conv_biases)
            relu = tf.nn.relu(bias)

            return relu

    def fc_layer(self, bottom, in_size, out_size, name):
        with tf.variable_scope(name):
            weights, biases = self.get_fc_var(in_size, out_size, name)

            x = tf.reshape(bottom, [-1, in_size])
            fc = tf.nn.bias_add(tf.matmul(x, weights), biases)

            return fc

    def get_conv_var(self, filter_size, in_channels, out_channels,
name):
        initial_value = tf.truncated_normal([filter_size, filter_size,
in_channels, out_channels], 0.0, 0.001)
        filters = self.get_var(initial_value, name, 0, name +
"_filters")

        initial_value = tf.truncated_normal([out_channels], .0, .001)
        biases = self.get_var(initial_value, name, 1, name +
"_biases")
```

```
        return filters, biases

    def get_fc_var(self, in_size, out_size, name):
        initial_value = tf.truncated_normal([in_size, out_size], 0.0,
0.001)
        weights = self.get_var(initial_value, name, 0, name +
"_weights")

        initial_value = tf.truncated_normal([out_size], .0, .001)
        biases = self.get_var(initial_value, name, 1, name +
"_biases")

        return weights, biases

    def get_var(self, initial_value, name, idx, var_name):
        if self.data_dict is not None and name in self.data_dict:
            value = self.data_dict[name][idx]
        else:
            value = initial_value

        if self.trainable:
            var = tf.Variable(value, name=var_name)
        else:
            var = tf.constant(value, dtype=tf.float32, name=var_name)

        self.var_dict[(name, idx)] = var

        assert var.get_shape() == initial_value.get_shape()

        return var

    def save_npy(self, sess, npy_path="./save.npy"):
        assert isinstance(sess, tf.Session)

        data_dict = {}

        for (name, idx), var in list(self.var_dict.items()):
            var_out = sess.run(var)
            if name not in data_dict:
                data_dict[name] = {}
            data_dict[name][idx] = var_out

        np.save(npy_path, data_dict)
        print(("file saved", npy_path))
        return npy_path

    def get_var_count(self):
        count = 0
        for v in list(self.var_dict.values()):
            count += reduce(lambda x, y: x * y,
```

```
v.get_shape().as_list())
        return count
```

3- test.py

```
"""
Simple tester for the FCN32_trainable
"""
import collections
import tensorflow as tf
import pickle

import numpy as np

import fcn32_trainable as fcn32
import utils

import glob


import matplotlib.pyplot as plt
import os
# os.listdir("somedirectory")

SOFTMAX = True

def imageprocessing(path):
    img = utils.load_image(path)
    batch = img.reshape((1, 352, 1216, 3))
    return batch

def gtprocessing(path):
    gt = pickle.load(open(path, encoding='latin1'))
    return gt


# data Path
test_image_Path =
"C:/Users/yuhou/OneDrive/CSCI677/homework6/HW6_Dataset/image/test"
train_image_Path =
"C:/Users/yuhou/OneDrive/CSCI677/homework6/HW6_Dataset/image/train"
test_gt_Path =
"C:/Users/yuhou/OneDrive/CSCI677/homework6/HW6_Dataset/label/test"
train_gt_Path =
"C:/Users/yuhou/OneDrive/CSCI677/homework6/HW6_Dataset/label/train"

# data list
test_image_list = os.listdir(test_image_Path)
train_image_list = os.listdir(train_image_Path)
test_gt_list = os.listdir(test_gt_Path)
train_gt_list = os.listdir(train_gt_Path)
```

```
# data
test_images = []
train_images = []
test_labels = []
train_labels = []
for path in (test_image_list):
    test_images.append(imageprocessing(test_image_Path+"/"+path))
for path in (test_gt_list):
    test_labels.append(gtprocessing(test_gt_Path + "/" + path))
for path in (train_image_list):
    train_images.append(imageprocessing(train_image_Path+"/"+path))
for path in (train_gt_list):
    train_labels.append(gtprocessing(train_gt_Path + "/" + path))

validation_images= train_images[199:]
train_images = train_images[0:199]

# train_images, train_labels
# validation_images, validation_labels
# test_images, test_labels



with tf.device('/cpu:0'):
    EPOCH = 50
    sess = tf.Session()

    images = tf.placeholder(tf.float32, [1, 352, 1216, 3])
    true_out = tf.placeholder(tf.float32, [1, 352, 1216, 1])
    train_mode = tf.placeholder(tf.bool)
    fcn = fcn32.Fcn32()
    fcn.build(images, train_mode)

    print(fcn.get_var_count())
    sess.run(tf.global_variables_initializer())

    # setup::::
    Entropy =
tf.nn.sigmoid_cross_entropy_with_logits(logits=fcn.conv9,
labels=true_out)
    mask = tf.minimum(true_out,0)+1
    costEntropy = tf.reduce_sum(Entropy*mask)/tf.reduce_sum(mask)
    train =
tf.train.GradientDescentOptimizer(0.0001).minimize(costEntropy)

    T=[]
    V=[]
    E = range(EPOCH)

    for epk in range(EPOCH):

        # simple 1-step training/// many steps:
```

```
        print ("@@@@@@@@@@@@@@@@@@@@@@@@@@@@ start training EPOCH =",
epk)
        lossFTrain = []
        for i in range(len(train_images)):
        # for i in range(3):
            if i % 50 ==0:
                print ("EPOCH",epk, "Train-",i)
            # get data
            img1 = train_images[i:i+1]
            img1_true_result = np.expand_dims(train_labels[i:i+1], -1)

            # train
            sess.run(train, feed_dict={images: img1, true_out:
img1_true_result, train_mode: True})

            # get entropy
            lossF = sess.run(costEntropy, feed_dict={images: img1,
true_out: img1_true_result, train_mode: False})

            lossFTrain.append(lossF)

        lossTall = sum(lossFTrain)/len(lossFTrain)
        T.append(lossTall)
        print("*****lossFall",lossTall)

        print ("@@@@@@@@@@@@@@@@@@@@@@@@@@@@ start validation EPOCH =
", epk)


        # validatiion step:
        lossFValidation = []
        for i in range(len(validation_images)):
        # for i in range(3):
            if i % 20 ==0:
                print("EPOCH", epk, "Validation-", i)
            # get data
            img1 = validation_images[i:i + 1]
            img1_true_result = np.expand_dims(validation_labels[i:i +
1], -1)

            # get prob and entropy
            prob = sess.run(fcn.prob, feed_dict={images: img1,
true_out: img1_true_result, train_mode: False})
            lossF = sess.run(costEntropy, feed_dict={images:
img1,true_out: img1_true_result, train_mode: False})

            lossFValidation.append(lossF)

        lossVall = sum(lossFValidation)/len(lossFValidation)
        V.append(lossVall)
        print("******lossVall",lossVall)
```

```
        # test_save
        fcn.save_npy(sess, "C:/Users/yuhou/OneDrive/桌面
/CSCI677/homework6/hw6/test-save-"+str(epk)+".npy")

    print("T",T)
    print("V",V)
    print("E",E)

    plt.figure()
    plt.plot(E,T,'ro')
    plt.plot(E,V,'x')
    plt.xticks(E,E, rotation = 'vertical',fontsize = 8)
    plt.margins(0.2)
    plt.subplots_adjust(bottom=0.3)
    plt.savefig('C:/Users/yuhou/OneDrive/桌面
/CSCI677/homework6/hw6/lossCurve.png')
    plt.show()


    # test_save
    fcn.save_npy(sess,"C:/Users/yuhou/OneDrive/桌面
/CSCI677/homework6/hw6/test-save.npy")

    # Test step:
    lossTTValidation = []
    PROB = []
    for i in range(len(test_images)):
    # for i in range(3):
        if i % 20 == 0:
            print("Test-", i)
        # get data
        img1 = test_images[i:i + 1]
        img1_true_result = np.expand_dims(test_labels[i:i + 1], -1)

        # get prob and entropy
        prob = sess.run(fcn.prob, feed_dict={images: img1, true_out:
img1_true_result, train_mode: False})
        PROB.append(prob)
        lossF = sess.run(costEntropy, feed_dict={images: img1,
true_out: img1_true_result, train_mode: False})

        lossTTValidation.append(lossF)

    lossVall = sum(lossTTValidation) / len(lossTTValidation)

    print("******lossVall", lossVall)
```

```
    np.save('C:/Users/yuhou/OneDrive/桌面
/CSCI677/homework6/hw6/prob.npy',np.array(PROB))
```

9 Reference:

[1] https://www.youtube.com/watch?v=JuRQAi5RR3A
[2] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_image_display/py_image_display.html
[3] https://www.tensorflow.org/api_docs/python/tf/initializers/local_variables
[4] https://www.tensorflow.org/api_docs/python/tf/convert_to_tensor