

Name: Sai Surya Salalith Mantha

USC_ID: 9463553709

Email ID: Saisurym@usc.edu

(HI,

Sorry for the late submission. I mailed professor and got extension. So please accept my submission.

Thank you,

Salalith)

Steps:

- 1) Loaded the dataset and split into train and validation data.
- 2) Build the model
- 3) Calculate the loss function and optimize the loss
- 4) Run for 100 epochs and validate every 3rd epoch
- 5) Generated the test result using the saved model and displayed the images.

I ran the algorithm with given specifications for 100 epochs and got a training loss of 0.05 and validation accuracy of 85%. I ran the code for the test images and got an accuracy of 86%.

Training Code:

Train.py

#import necessary packages

```
import tensorflow as tf
import cv2
import pickle
import os
import sys
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib
matplotlib.use('agg')
import pylab as plt
```

#reading data

```
train=sorted(os.listdir("HW6_Dataset/image/train"))
test=sorted(os.listdir("HW6_Dataset/image/test"))
```

```
train_labels=sorted(os.listdir("HW6_Dataset/label/train"))
test_labels=sorted(os.listdir("HW6_Dataset/label/test"))
img=cv2.imread("HW6_Dataset/image/train/"+train[2])
```

```
X_train=[]
for i in train:
    X_train.append(cv2.imread("HW6_Dataset/image/train/"+i))
```

```
y_train=[]
def unpickle(file):
```

```
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

```
img_label=unpickle("HW6_Dataset/label/train/"+train_labels[2])
```

```
for i in train_labels:
    y_train.append(unpickle("HW6_Dataset/label/train/"+i))
```

```
y_train=np.array(y_train)
y_train=np.where(y_train<0,0,y_train)
```

train and validation split

```
X_train,X_test,y_train,y_test=train_test_split(X_train,y_train,test_size=0.2,random_state=1)
```

place holder for data

```
x = tf.placeholder(tf.float32, shape=(None, 352, 1216, 3), name='input_x')
y = tf.placeholder(tf.float32, shape=(None, 352, 1216), name='output_y')
```

Calculating Accuracy

```
def calc(pred,label):
    # label = y_train[pos]
    pred=np.where(pred>0.5,1,0)
    tp = 0
```



```

        activation=tf.nn.relu)
conv9 = tf.layers.conv2d(inputs=conv8, filters=512,
        kernel_size=[3, 3], padding="same",
        activation=tf.nn.relu)
conv10 = tf.layers.conv2d(inputs=conv9, filters=512,
        kernel_size=[3, 3], padding="same",
        activation=tf.nn.relu)

pool4 = tf.layers.max_pooling2d(inputs=conv10,
        pool_size=[2, 2], strides=2,
        padding="same")

conv11 = tf.layers.conv2d(inputs=pool4, filters=512,
        kernel_size=[3, 3], padding="same",
        activation=tf.nn.relu)
conv12 = tf.layers.conv2d(inputs=conv11, filters=512,
        kernel_size=[3, 3], padding="same",
        activation=tf.nn.relu)
conv13 = tf.layers.conv2d(inputs=conv12, filters=512,
        kernel_size=[3, 3], padding="same",
        activation=tf.nn.relu)

pool5 = tf.layers.max_pooling2d(inputs=conv13,
        pool_size=[2, 2], strides=2,
        padding="same")

conv14=tf.layers.conv2d(inputs=pool5,filters=4096,
        kernel_size=[7,7],activation=tf.nn.relu,padding="same")

conv15=tf.layers.conv2d(inputs=conv14,filters=4096,
        kernel_size=[1,1],activation=tf.nn.relu,padding="same")

conv16 = tf.layers.conv2d(inputs=conv15, filters=4096,
        kernel_size=[1, 1],padding="same")

deconv=tf.layers.conv2d_transpose(inputs=conv16,filters=1,
        kernel_size=[64,64],strides=32,padding="same")

    return deconv

#logits

logits=vgg16(x)

#loss function

loss=tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=tf.squeeze(logits,[
3]),labels=y))
pred=tf.sigmoid(tf.squeeze(logits,[0,3]),name="predi")

#optimizer

optimizer=tf.train.MomentumOptimizer(learning_rate=0.001,momentum=0.99).minimize(loss)
model_path = './image_classification'

```

```

with tf.Session() as sess:
    sess.run(tf.initializers.global_variables())
    loss_array=[]
    for epochs in range(1,100):
        print(epochs)
        rand_index = np.random.choice(len(X_train), size=len(X_train))

        for pos in range(len(X_train)):
            sess.run(optimizer, feed_dict = {x: [X_train[rand_index[pos]]], y:
[y_train[rand_index[pos]]]})
            cost=sess.run(loss, feed_dict={x: [X_train[rand_index[pos]]], y:
[y_train[rand_index[pos]]]})
            loss_array.append(cost)
            # print("image"+str(pos)+" loss= "+str(cost))
        print("epoch: "+str(epochs)+" loss: "+str(sum(loss_array)/len(loss_array)))

    model_saver = tf.train.Saver()
    save_path_model = model_saver.save(sess, model_path)
    plt.plot(loss_array)
    plt.show()
    plt.savefig("myfig")

    if(epochs%3==0):
        acc=[]
        for pos in range(len(X_test)):
            pred1=sess.run(pred, feed_dict = {x: [X_test[pos]], y: [y_test[pos]]})
            acc.append(calc(pred1, y_test[pos]))
        print("val accuracy: "+str(sum(acc)/len(acc)))

```

Testing Code

#import packages

```

import os
import pickle
import random
import sys
import cv2
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
import pylab as plt
from PIL import Image

```

#reading data

def unpickle(file):

```

    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict

```

#calculating accuracy

```
def calc(pred, label, lab):
    # label = y_train[pos]
    pred=np.where(pred>0.5,1,0)
    tp = 0
    fp = 0
    fn = 0
    img=[]
    for i in range(352):
        temp=[]
        for j in range(1216):
            if(pred[i][j]==1):
                temp.append([255,0,0])
            else:
                temp.append([255,0,255])
            if (label[i][j] == 1 and pred[i][j] ==1):
                tp += 1
            if (label[i][j] == 1 and pred[i][j] ==0):
                fn += 1
            if (label[i][j] == 0 and pred[i][j] ==1):
                fp += 1
        img.append(temp)

    # cv2.imwrite()
    img_i=Image.fromarray(np.array(img).astype(np.uint8))
    # print(lab)
    img_i.save("images/"+lab)
    print("accuracy= " + str(tp / (tp + fp + fn)))
    return (tp / (tp + fp + fn))

test=sorted(os.listdir("HW6_Dataset/image/test"))
test_labels=sorted(os.listdir("HW6_Dataset/label/test"))

X_test=[]
for i in range(1,len(test)):
    X_test.append(cv2.imread("HW6_Dataset/image/test/"+test[i]))
ytest=[]
for i in range(len(test_labels)):
    ytest.append(unpickle("HW6_Dataset/label/test/"+test_labels[i]))

y_test=np.array(ytest)
y_test=np.where(y_test<0,0,y_test)
print(np.array(X_test).shape)
# print(y_test[0].shape)

model_path = './image_classification'

loaded_graph = tf.Graph()
with tf.Session(graph=loaded_graph) as sess:
    loader = tf.train.import_meta_graph(model_path + '.meta')
    loader.restore(sess, model_path)
    loaded_x = loaded_graph.get_tensor_by_name('input_x:0')
    loaded_y = loaded_graph.get_tensor_by_name('output_y:0')
    loaded_acc = loaded_graph.get_tensor_by_name('predi:0')

    accuracy=[]
```

```
for i in range(len(X_test)):

    #printing accuracy
    pred=sess.run(loader_acc,feed_dict={loader_x:[X_test[i]],loader_y:[y_test[i]]})
    accuracy.append(calc(pred, y_test[i],test[i+1]))
    print("Total accuracy: "+str(sum(accuracy)/len(accuracy)))
```

Training Loss and Validation
Accuracy:

```
saisuryam@CS1677: ~/training2
2018-11-30 08:17:40.367465: I tensorflow/core/common_runtime/gpu/gpu_device.cc:984] 0:  N
2018-11-30 08:17:40.367710: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1097] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 10758
MB memory) -> physical GPU (device: 0, name: Tesla K80, pci bus id: 967d:00:00.0, compute capability: 3.7)
1
epoch: 1 loss: 0.446547307800024
2
epoch: 2 loss: 0.37801777676512033
3
epoch: 3 loss: 0.3586458500676477
val accuracy: 0.11060887897307044
4
epoch: 4 loss: 0.3275956386079391
5
epoch: 5 loss: 0.3044039775775029
6
epoch: 6 loss: 0.28951397130631995
val accuracy: 0.5552611955694603
7
epoch: 7 loss: 0.2738641334104014
8
epoch: 8 loss: 0.2613326456087331
9
epoch: 9 loss: 0.26023205480089895
val accuracy: 7.378731618720506e-05
10
epoch: 10 loss: 0.2561006223849761
11
epoch: 11 loss: 0.2503296059675706
12
epoch: 12 loss: 0.24526692992155877
val accuracy: 0.29961454186760317
13
epoch: 13 loss: 0.24237298229213297
14
epoch: 14 loss: 0.2393775114864657
15
epoch: 15 loss: 0.23358682399886285
val accuracy: 0.7072053859783106
16
epoch: 16 loss: 0.22668734093805631
17
epoch: 17 loss: 0.22075516679359328
18
epoch: 18 loss: 0.21575516679359328
19
epoch: 19 loss: 0.21075516679359328
20
epoch: 20 loss: 0.20575516679359328
21
epoch: 21 loss: 0.20075516679359328
22
epoch: 22 loss: 0.19575516679359328
23
epoch: 23 loss: 0.19075516679359328
24
epoch: 24 loss: 0.18575516679359328
25
epoch: 25 loss: 0.18075516679359328
26
epoch: 26 loss: 0.17575516679359328
27
epoch: 27 loss: 0.17075516679359328
28
epoch: 28 loss: 0.16575516679359328
29
epoch: 29 loss: 0.16075516679359328
30
epoch: 30 loss: 0.15575516679359328
31
epoch: 31 loss: 0.15075516679359328
32
epoch: 32 loss: 0.14575516679359328
33
epoch: 33 loss: 0.14075516679359328
34
epoch: 34 loss: 0.13575516679359328
35
epoch: 35 loss: 0.13075516679359328
36
epoch: 36 loss: 0.12575516679359328
37
epoch: 37 loss: 0.12075516679359328
38
epoch: 38 loss: 0.11575516679359328
39
epoch: 39 loss: 0.11075516679359328
40
epoch: 40 loss: 0.10575516679359328
41
epoch: 41 loss: 0.10075516679359328
42
epoch: 42 loss: 0.09575516679359328
43
epoch: 43 loss: 0.09075516679359328
44
epoch: 44 loss: 0.08575516679359328
45
epoch: 45 loss: 0.08075516679359328
46
epoch: 46 loss: 0.07575516679359328
47
epoch: 47 loss: 0.07075516679359328
48
epoch: 48 loss: 0.06575516679359328
49
epoch: 49 loss: 0.06075516679359328
50
epoch: 50 loss: 0.05575516679359328
51
epoch: 51 loss: 0.05075516679359328
52
epoch: 52 loss: 0.04575516679359328
53
epoch: 53 loss: 0.04075516679359328
54
epoch: 54 loss: 0.03575516679359328
55
epoch: 55 loss: 0.03075516679359328
56
epoch: 56 loss: 0.02575516679359328
57
epoch: 57 loss: 0.02075516679359328
58
epoch: 58 loss: 0.01575516679359328
59
epoch: 59 loss: 0.01075516679359328
60
epoch: 60 loss: 0.00575516679359328
61
epoch: 61 loss: 0.00075516679359328
62
epoch: 62 loss: 0.00075516679359328
63
epoch: 63 loss: 0.00075516679359328
64
epoch: 64 loss: 0.00075516679359328
65
epoch: 65 loss: 0.00075516679359328
66
epoch: 66 loss: 0.00075516679359328
67
epoch: 67 loss: 0.00075516679359328
68
epoch: 68 loss: 0.00075516679359328
69
epoch: 69 loss: 0.00075516679359328
70
epoch: 70 loss: 0.00075516679359328
71
epoch: 71 loss: 0.00075516679359328
72
epoch: 72 loss: 0.07213147364223964
val accuracy: 0.8597947738022602
73
epoch: 73 loss: 0.07114616617716649
74
epoch: 74 loss: 0.0701869455347759
75
epoch: 75 loss: 0.06925835080650178
val accuracy: 0.861515334203731
76
epoch: 76 loss: 0.06835447330745266
77
epoch: 77 loss: 0.06747370665264943
78
epoch: 78 loss: 0.06661318628653332
val accuracy: 0.8595381309445541
79
epoch: 79 loss: 0.06577321420292637
80
epoch: 80 loss: 0.0649529964209395
81
epoch: 81 loss: 0.06415223491440625
val accuracy: 0.8581748194287498
82
epoch: 82 loss: 0.06337055666332576
83
epoch: 83 loss: 0.0626078241629574
84
epoch: 84 loss: 0.06186307193498799
val accuracy: 0.859587746518029
85
epoch: 85 loss: 0.06113563554383652
86
epoch: 86 loss: 0.06042526620695689
87
epoch: 87 loss: 0.059731129485489864
val accuracy: 0.858927977071797
88
epoch: 88 loss: 0.05905256655136793
89
epoch: 89 loss: 0.05839049006468917
90
epoch: 90 loss: 0.05772841179359328
91
epoch: 91 loss: 0.05706632250000000
92
epoch: 92 loss: 0.05640423520000000
93
epoch: 93 loss: 0.05574214790000000
94
epoch: 94 loss: 0.05508006060000000
95
epoch: 95 loss: 0.05441797330000000
96
epoch: 96 loss: 0.05375588600000000
97
epoch: 97 loss: 0.05309380000000000
98
epoch: 98 loss: 0.05243171330000000
99
epoch: 99 loss: 0.05176962660000000
100
epoch: 100 loss: 0.05110754000000000
```

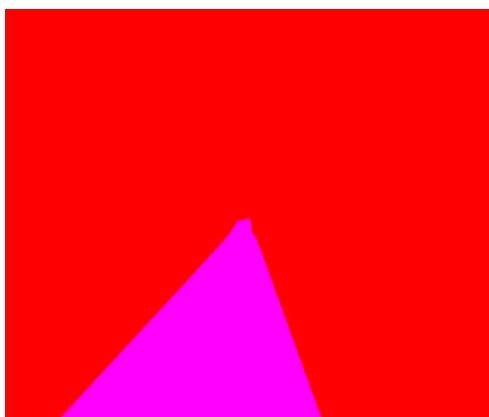
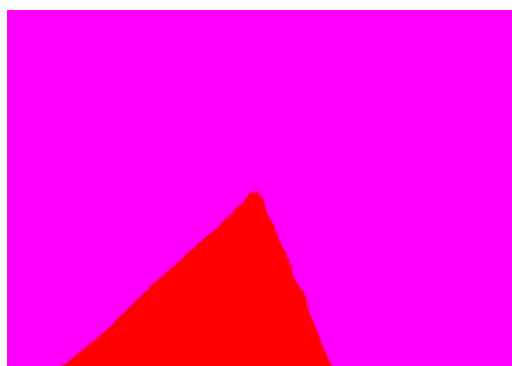
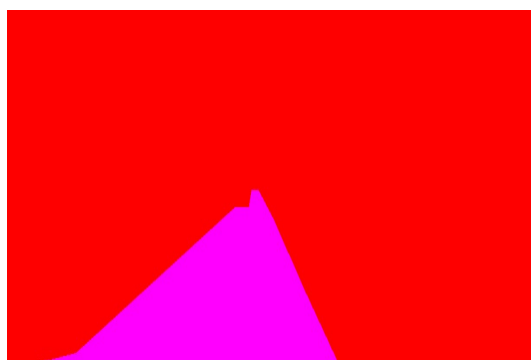
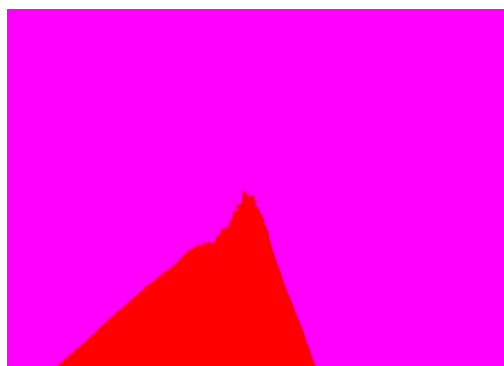
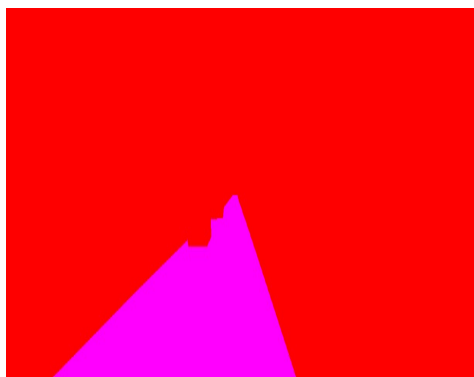
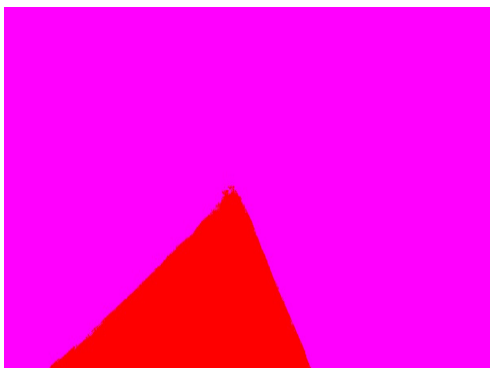
Testing Accuracy:

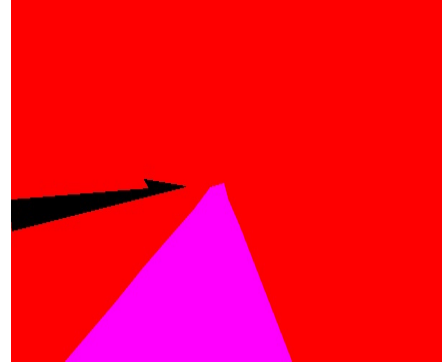

```
saisuryam@CSCI677: ~/training2
accuracy= 0.7817885564056602
accuracy= 0.9018208267071021
accuracy= 0.8375901980725459
accuracy= 0.9335697700308534
accuracy= 0.8336272329985432
accuracy= 0.9294661334744114
accuracy= 0.9167481447049961
accuracy= 0.9236856719733432
accuracy= 0.703242722197551
accuracy= 0.7023908061645702
accuracy= 0.8029007905190486
accuracy= 0.8260407534337557
accuracy= 0.8944882633542719
accuracy= 0.9026476350405821
accuracy= 0.8887011300019543
accuracy= 0.9708066905801622
accuracy= 0.9319809520210133
accuracy= 0.8433136498326547
accuracy= 0.8568740198640878
accuracy= 0.8819324950363997
accuracy= 0.8472526731203832
accuracy= 0.8175555279297109
accuracy= 0.7405227877871137
accuracy= 0.8027915298260602
accuracy= 0.7483061338272399
accuracy= 0.8619667552592459
accuracy= 0.9372370838179506
accuracy= 0.7983143417869802
accuracy= 0.8999562171628721
accuracy= 0.6227104466131436
accuracy= 0.9212016826206485
accuracy= 0.9023646453032045
accuracy= 0.8470024654370689
accuracy= 0.874231081016642
accuracy= 0.8760006240773855
accuracy= 0.8422568235106649
accuracy= 0.8138013371537727
accuracy= 0.799891926087988
accuracy= 0.9036628276471409
accuracy= 0.976875884851345
accuracy= 0.9653136795406164
accuracy= 0.9794662575652848
Total accuracy: 0.8601991650528172
saisuryam@CSCI677: ~/training2 $
```

Result:

Results

GroundTruths





We can see from the results that those are similar to ground truth and last pair has some differences. First three can be considered as success and last one is considered as failure.

I even tried using Adam optimizer but the training loss is not impressive.

In Conclusion, the results of the this model is reasonable.