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_st
ate=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
```

```
    fp = 0
    fn = 0
    for i in range(352):
        for j in range(1216):
            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
    # print("accuracy= " + str(tp / (tp + fp + fn)))
    return (tp / (tp + fp + fn))


#Model

def vgg16(features):

    # features=tf.expand_dims(features,0)

    conv1=tf.layers.conv2d(inputs=features,filters=64,
                           kernel_size=[3,3],padding="same",
                           activation=tf.nn.relu)

    conv2=tf.layers.conv2d(inputs=conv1,filters=64,
                           kernel_size=[3,3],padding="same",
                           activation=tf.nn.relu)


    pool1=tf.layers.max_pooling2d(inputs=conv2,
                                  pool_size=[2,2],strides=2,
                                  padding="same")

    conv3=tf.layers.conv2d(inputs=pool1,filters=128,
                           kernel_size=[3,3],padding="same",
                           activation=tf.nn.relu)

    conv4=tf.layers.conv2d(inputs=conv3,filters=128,
                           kernel_size=[3,3],padding="same",
                           activation=tf.nn.relu)

    pool2 = tf.layers.max_pooling2d(inputs=conv4,
                                    pool_size=[2, 2], strides=2,
                                    padding="same")

    conv5 = tf.layers.conv2d(inputs=pool2, filters=256,
                             kernel_size=[3, 3], padding="same",
                             activation=tf.nn.relu)

    conv6 = tf.layers.conv2d(inputs=conv5, filters=256,
                             kernel_size=[3, 3], padding="same",
                             activation=tf.nn.relu)

    conv7 = tf.layers.conv2d(inputs=conv6, filters=256,
                             kernel_size=[3, 3], padding="same",
                             activation=tf.nn.relu)

    pool3 = tf.layers.max_pooling2d(inputs=conv7,
                                    pool_size=[2, 2], strides=2,
                                    padding="same")

    conv8 = tf.layers.conv2d(inputs=pool3, filters=512,
                             kernel_size=[3, 3], padding="same",
```

```
                            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)))
```

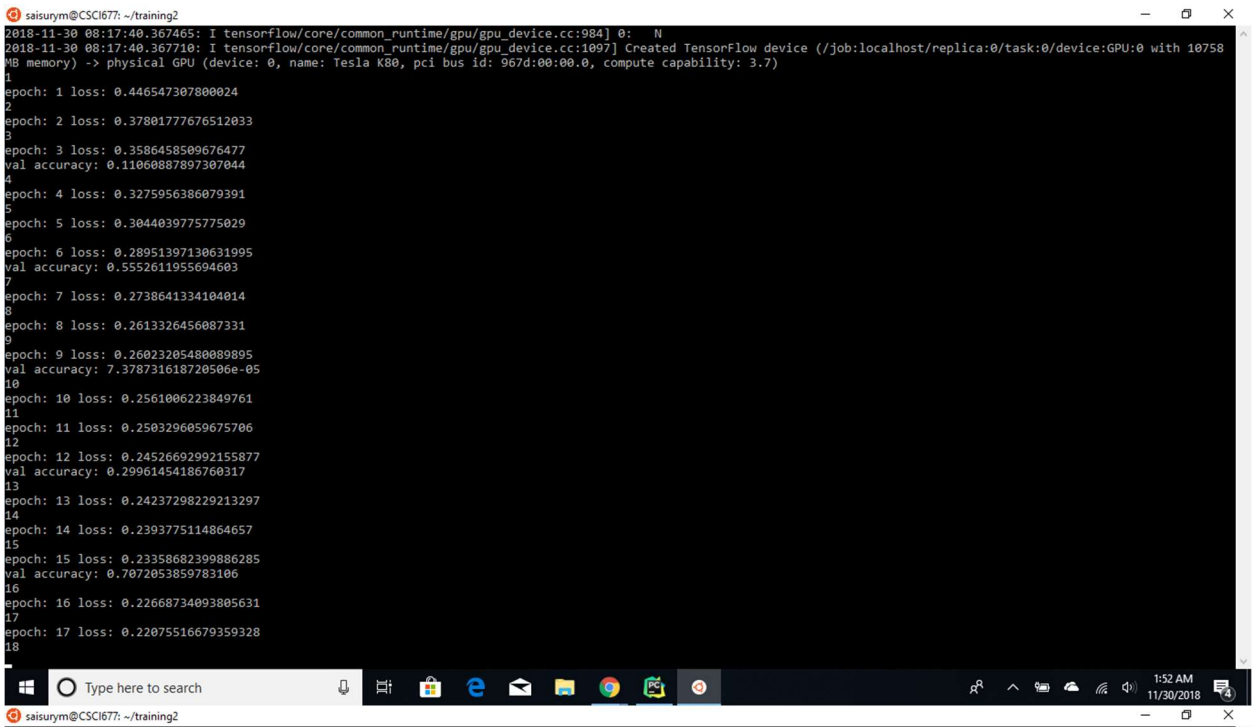
Testing Code

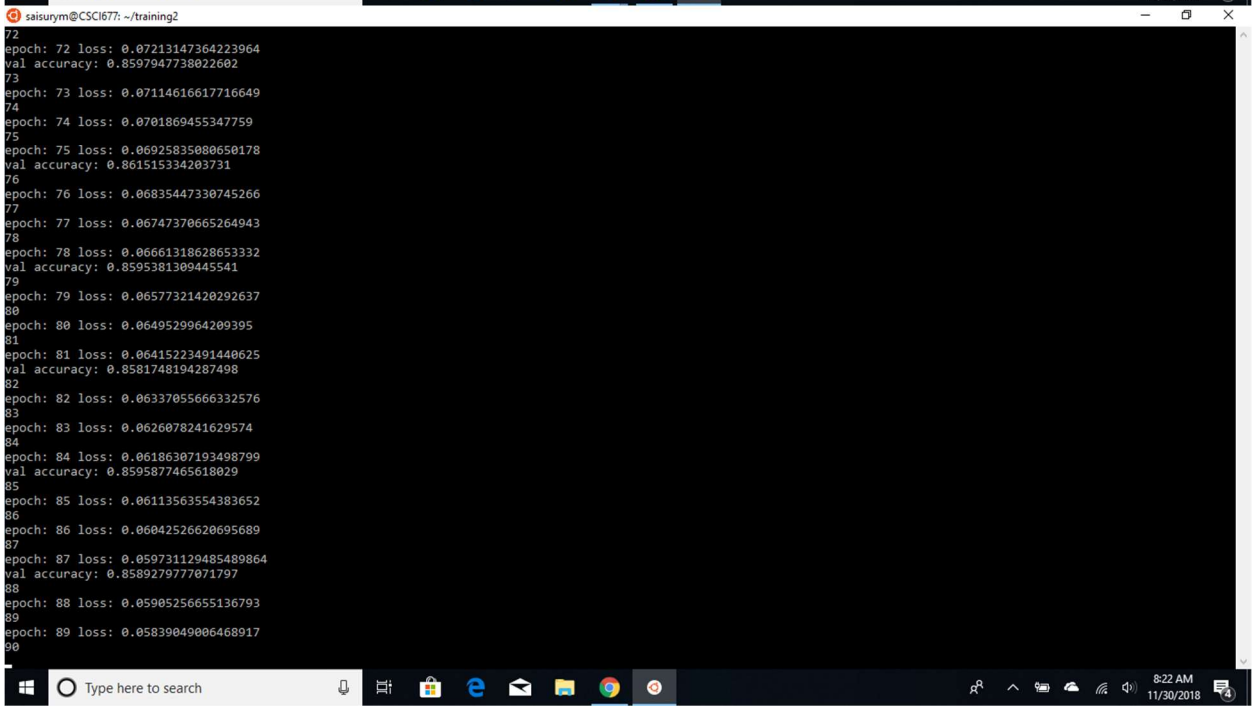
```
#import packages

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=[]
```

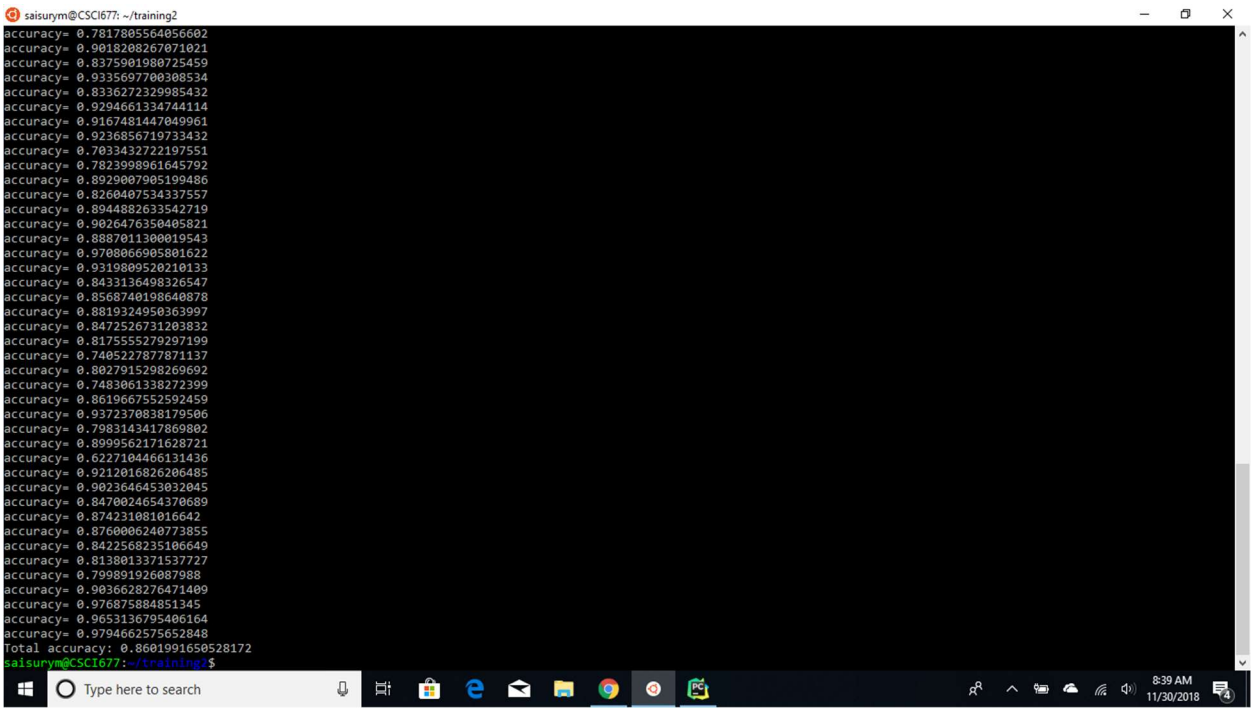
```
for i in range(len(X_test)):


   #printing accuracy
   pred=sess.run(loaded_acc,feed_dict={loaded_x:[X_test[i]],loaded_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:



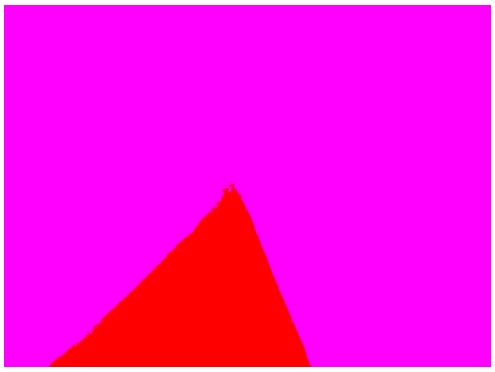


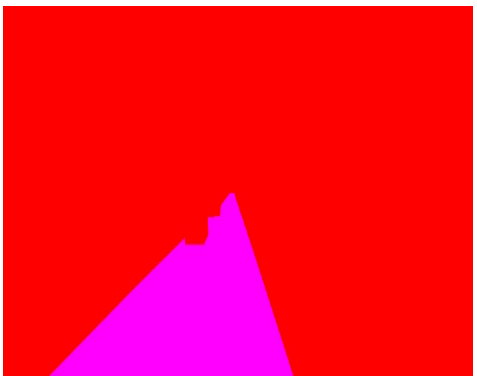
Testing Accuracy:

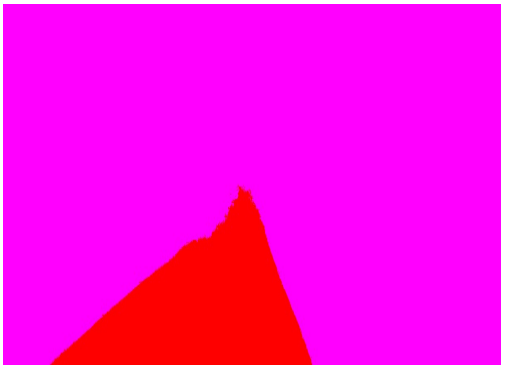


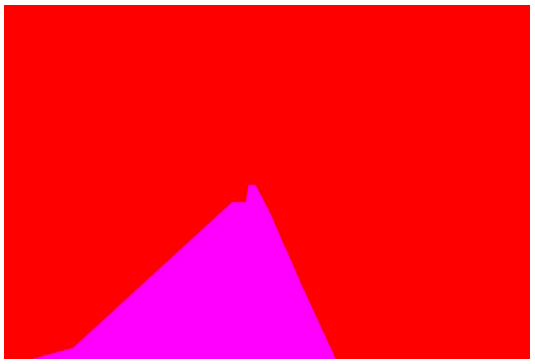
Result:

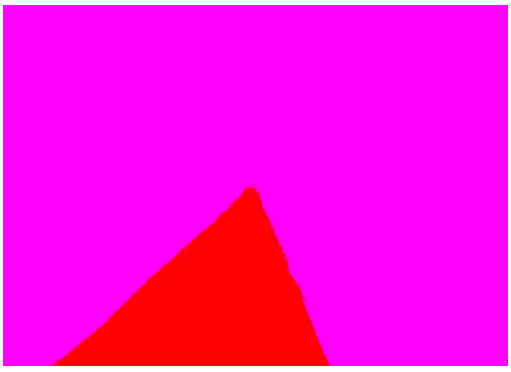
Results GroundTruths

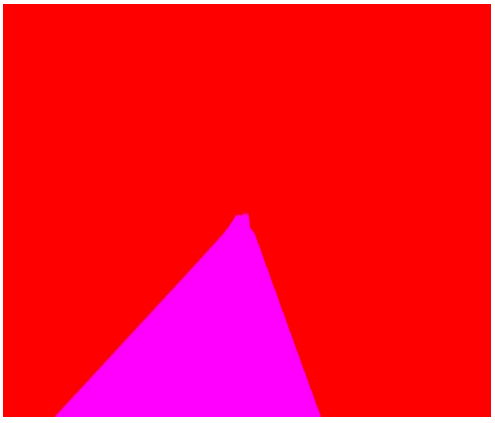


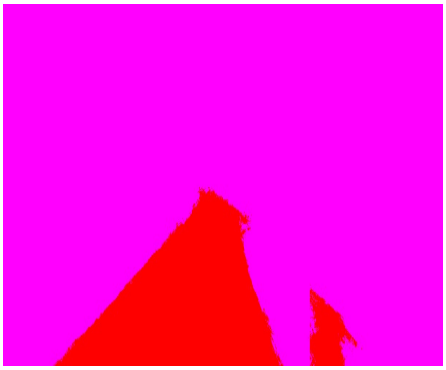


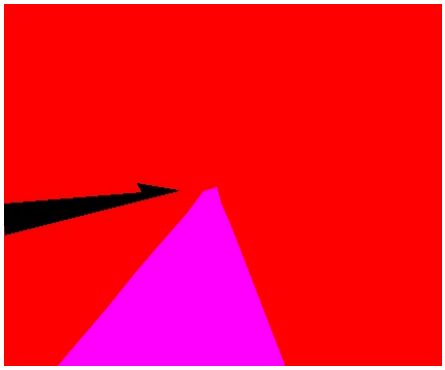












We can see from the results that those are similar to ground truth and last pair has some differences. First three can be considered as success and last one is considered as failure.

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

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