PURDUE UNIVERSITY NORTHWEST

PROJECT

DEEP LEARNING CLASS

PROFESSOR RICARDO CALIX

APPLYING CONVOLUTION NEURAL NETWORK ON COLOUR IMAGES DATASET.

DATA SOURCE: KAGGLE DATASETS

DATA SET:

CLASSIFICATION OF SCENERY IMAGES:

Objective:

To classify scenery images that consist of 4 classes that are sea , mountain , forest and buildings.

SAMPLE IMAGES :




CONVOLUTION NEURAL NETWORK CODE:

```
##!/usr/bin/env python
## Deep Learning code
##Professor Ricardo Calix
## simple example of a convolutional neural network for RGB data
########################################################################

import sklearn
import tensorflow as tf
```

```
import numpy as np
from numpy import genfromtxt
from sklearn import datasets
#from sklearn.cross_validation import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score, f1_score
import pandas as pd
import matplotlib.pyplot as plt

############################################################
## set parameters

import warnings
warnings.filterwarnings("ignore")
np.set_printoptions(threshold=np.inf) #print all values in numpy array

############################################################
#parameters for the main loop

learning_rate = 0.001
n_epochs = 6000  ##27000
batch_size = 150
display_step = 10


# Parameters for the network
```

```
n_input = 7500 # 50x50x3


n_classes = 4 # (0-3 classes)

dropout = 0.75 # Dropout, probability to keep units


####################################################################################
##########

## this will create your own data set (i.e. your_mnist)

## put your images in testA (I used pngs)


from PIL import Image


#import cv2

import glob

import numpy as np


train = []

labels = []


####################################################################################
###########


files = glob.glob ("scenery/forest/*.jpg") # your images path


for myFile in files:

    my_im = Image.open(myFile).convert('RGB')    ## .convert('LA') ## is for greyscale

    #my_im.show()

    resized_my_im = my_im.resize((50,50))     ## resize from 150x150x3 to 50x50x3
```

```
    #resized_my_im.show()

    image = np.array(resized_my_im)

    #image = np.array(my_im)

    print(image.shape)

    new_image = image.reshape(image.shape[0]*image.shape[1]*image.shape[2])

    print(new_image.shape)

    #input_string = input("???")

    train.append(new_image)

    labels.append(0)



#####################################################################################
#############


files = glob.glob ("scenery/buildings/*.jpg") # your images path


for myFile in files:

    my_im = Image.open(myFile).convert('RGB')    ## .convert('LA') ## is for greyscale

    #my_im.show()

    resized_my_im = my_im.resize((50,50))     ## resize from 150x150x3 to 50x50x3

    #resized_my_im.show()

    image = np.array(resized_my_im)

    #image = np.array(my_im)

    print(image.shape)

    new_image = image.reshape(image.shape[0]*image.shape[1]*image.shape[2])

    print(new_image.shape)

    #input_string = input("???")

    train.append(new_image)

    labels.append(1)
```

```
#####################################################################################
###################

files = glob.glob ("scenery/mountain/*.jpg") # your images path

for myFile in files:
    my_im = Image.open(myFile).convert('RGB')    ## .convert('LA') ## is for greyscale
    #my_im.show()
    resized_my_im = my_im.resize((50,50))     ## resize from 150x150x3 to 50x50x3
    #resized_my_im.show()
    image = np.array(resized_my_im)
    #image = np.array(my_im)
    print(image.shape)
    new_image = image.reshape(image.shape[0]*image.shape[1]*image.shape[2])
    print(new_image.shape)
    #input_string = input("???")
    train.append(new_image)
    labels.append(2)


#####################################################################################
#############

files = glob.glob ("scenery/sea/*.jpg") # your images path

for myFile in files:
    my_im = Image.open(myFile).convert('RGB')    ## .convert('LA') ## is for greyscale
    #my_im.show()
```

```
    resized_my_im = my_im.resize((50,50))     ## resize from 150x150x3 to 50x50x3

    #resized_my_im.show()

    image = np.array(resized_my_im)

    #image = np.array(my_im)

    print(image.shape)

    new_image = image.reshape(image.shape[0]*image.shape[1]*image.shape[2])

    print(new_image.shape)

    #input_string = input("???")

    train.append(new_image)

    labels.append(3)

##################################################################################################
##########################

##################################################################################################
########################

train = np.array(train,dtype='float32')

labels = np.array(labels, dtype='float32')

# convert (number of images x height x width x number of channels) to (number of images x (height *
width *3))

# for example (120 * 40 * 40 * 3)-> (120 * 4800)

#train = np.reshape(train,[train.shape[0],train.shape[1]*train.shape[2]*train.shape[3]])

#train = np.reshape(train,[train.shape[0],train.shape[1]*train.shape[2]])

print(train.shape)

print(labels.shape)

#print(train.shape[1])

#input_string = input("train size is")
```

```
#####################################################################################
#########################


# save numpy array as .npy formats

np.save('train',train)


your_mnist = train


#####################################################################################
###########################

## normalization is very important


x=your_mnist

xmax, xmin = x.max(), x.min()

x = (x - xmin)/(xmax - xmin)


your_mnist = x



#####################################################################################
#######################


x_all = your_mnist

labels_all = labels


x_train, x_test, y_train, y_test = train_test_split(x_all, labels_all, test_size=0.30, random_state=42)
```

```
#####################################################################################
###########################

## print stats
precision_scores_list = []
accuracy_scores_list = []

def print_stats_metrics(y_test, y_pred):
    print('Accuracy: %.2f' % accuracy_score(y_test,   y_pred) )
    #Accuracy: 0.84
    accuracy_scores_list.append(accuracy_score(y_test,   y_pred) )
    confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
    print "confusion matrix"
    print(confmat)
    print pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
    precision_scores_list.append(precision_score(y_true=y_test, y_pred=y_pred, average='weighted'))
    print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred, average='weighted'))
    print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred, average='weighted'))
    print('F1-measure: %.3f' % f1_score(y_true=y_test, y_pred=y_pred, average='weighted'))

#####################################################################

########################################################################

def conv2d(x, W, b, strides=1):
    # Conv2D function, with bias and relu activation
    x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')
    x = tf.nn.bias_add(x, b)
    return tf.nn.relu(x)      ## relu removes negative values
```

```
################################################################################

def maxpool2d(x, k=2):
    # MaxPool2D function
    # padding='SAME' is very useful for uneven images. If maxpooling
    # image 25x25 -> 12.5 x 12.5 then it is rounded up to 13x13
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k, k, 1],
                          padding='SAME')

######################################################################

def layer(input, weight_shape, bias_shape):
    W = tf.Variable(tf.random_normal(weight_shape))
    b = tf.Variable(tf.random_normal(bias_shape))
    mapping = tf.matmul(input, W)
    result = tf.add( mapping ,  b )
    return result

######################################################################

def conv_layer(input, weight_shape, bias_shape):
    ##rr =raw_input()
    W = tf.Variable(tf.random_normal(weight_shape))
    b = tf.Variable(tf.random_normal(bias_shape))
    conv = conv2d(input, W, b)
    # Max Pooling (down-sampling)
```

```
    conv_max = maxpool2d(conv, k=2)
    return conv_max

#################################################################

def fully_connected_layer(conv_input, fc_weight_shape, fc_bias_shape, dropout):
    new_shape = [-1, tf.Variable(tf.random_normal(fc_weight_shape)).get_shape().as_list()[0]]
    fc = tf.reshape(conv_input, new_shape)
    w_fc = tf.Variable(   tf.random_normal(   fc_weight_shape  )    )
    mapping = tf.matmul(   fc ,  w_fc   )   # y = w * x
    fc = tf.add( mapping, tf.Variable(tf.random_normal(  fc_bias_shape  ))    )
    fc = tf.nn.relu(fc)
    # Apply Dropout
    fc = tf.nn.dropout(fc, dropout)
    return fc


############################################################
## define the architecture here

def inference_conv_net_3_convolutions(x, dropout):
    # Reshape input picture
    # shape = [-1, size_image_x, size_image_y, 3 channels (e.g. rgb)]
    # the imge for rgb and batches of 150 would be [150, 7500] because
    # there are 128 samples per batch and images are 50x50x3 = 7500
    # this has to be re-shaped bacause Convolutional layers only take 4 dimensional tensors as input
    # the -1 infers the number of batches and then we make the 7500 into 50x50x3
    x = tf.reshape(x, shape=[-1, 50, 50, 3])
```

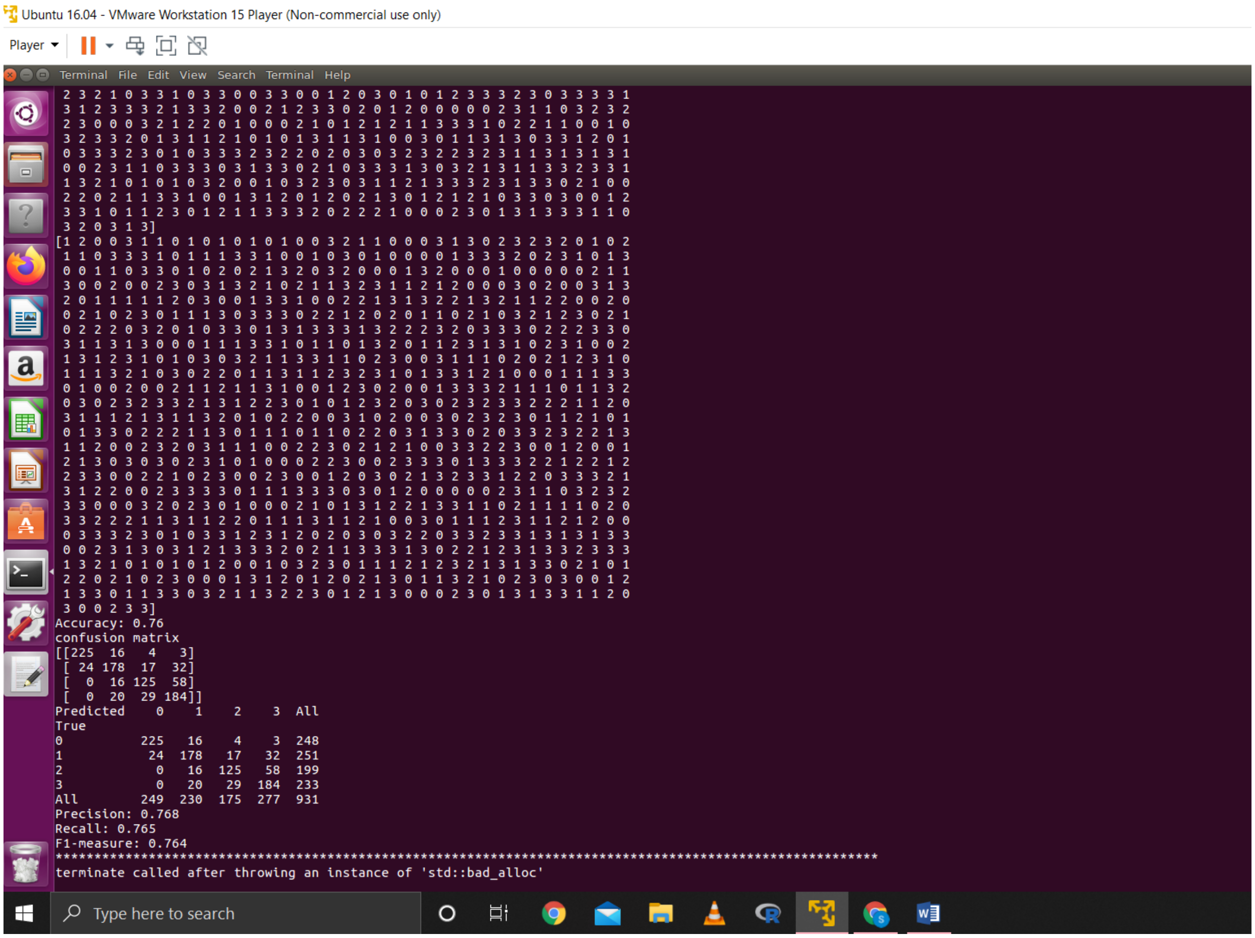
```
    # Convolution Layer 1, filter 5x5 conv, 3 inputs or 3 channels, 16 outputs
    # max pool will reduce image from 50x50 to 25x25
    conv1 = conv_layer(x, [5, 5, 3, 16], [16] )

    # Convolution Layer 2, filter 5x5 conv, 16 inputs, 36 outputs
    # max pool will reduce image from 25x25 to 13x13
    conv2 = conv_layer(conv1, [5, 5, 16, 36], [36] )

    # Convolution Layer 2, filter 5x5 conv, 36 inputs, 64 outputs
    # max pool will reduce image from 13x13 to 7x7
    conv3 = conv_layer(conv2, [5, 5, 36, 64], [64] )


    # Fully connected layer, 7*7*64 inputs, 1024 outputs
    # Reshape conv2 output to fit fully connected layer input
    # maxpool function padding=same rounds up 6.5 to 7
    fc1 = fully_connected_layer(conv3, [7*7*64, 1024], [1024] , dropout)

    # Output, 128 inputs, 10 outputs (class prediction)
    output = layer(fc1 ,[1024, n_classes], [n_classes] )
    return output


##############################################################

def loss_deep_conv_net(output, y_tf):
    xentropy = tf.nn.softmax_cross_entropy_with_logits(logits=output, labels=y_tf)
    loss = tf.reduce_mean(xentropy)
    return loss
```

```
##############################################################

def training(cost):
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
    train_op = optimizer.minimize(cost)
    return train_op

##############################################################

def evaluate(output, y_tf):
    correct_prediction = tf.equal(tf.argmax(output,1), tf.argmax(y_tf,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    return accuracy

##############################################################

x_tf = tf.placeholder(tf.float32, [None, n_input])   ## 50x50x3
y_tf = tf.placeholder(tf.float32, [None, n_classes])
keep_prob = tf.placeholder(tf.float32) #dropout (keep probability)

##################################################################
```

```
output = inference_conv_net_3_convolutions(x_tf, keep_prob)
#output = inference_conv_net2(x_tf, keep_prob)
#output = inference_conv_net(x_tf, keep_prob)
cost = loss_deep_conv_net(output, y_tf)

train_op = training(cost)
eval_op = evaluate(output, y_tf)


####################################################################
## for metrics

y_p_metrics = tf.argmax(output, 1)

####################################################################
# Initialize and run

#init = tf.global_variables_initializer()

init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)


##############################################################
# one-hot encoding

depth = 4
y_train_onehot = sess.run(tf.one_hot(y_train, depth))
```

```
y_test_onehot  = sess.run(tf.one_hot(y_test, depth))


##############################################################
## batch parameters

num_samples_train =  len(y_train)
print num_samples_train
num_batches = int(num_samples_train/batch_size)


##########################################################################################
######
dropout2 = 1.0


################################################################
# MAIN_LOOP()



for i in range(n_epochs):
   for batch_n in range(num_batches):
       sta= batch_n*batch_size
       end= sta + batch_size


       sess.run( train_op , feed_dict={x_tf: x_train[sta:end,:] , y_tf: y_train_onehot[sta:end, :], keep_prob:
dropout      })
```

```
        loss, acc = sess.run([cost, eval_op], feed_dict={x_tf: x_train[sta:end,:] , y_tf: 
y_train_onehot[sta:end, :], keep_prob: dropout2})



        result = sess.run(eval_op, feed_dict={x_tf: x_test, y_tf: y_test_onehot, keep_prob: dropout2})

        result2, y_pred = sess.run([eval_op, y_p_metrics], feed_dict={x_tf: x_test, y_tf: y_test_onehot, 
keep_prob: dropout2})



        print "test1 {},{}".format(i,result)

        print "test2 {},{}".format(i,result2)


        y_true = np.argmax(y_test_onehot, 1)

        print y_pred

        print y_true

        print_stats_metrics(y_true, y_pred)

        print 
"*************************************************************************************
**********************"



#########################################################################################
#####


print "<<<<<<<<<<<<<<<<<<DONE>>>>>>>>>>>>>>>>>>>>>>>"
```



Output:

RESULTS:

	PRECISION	RECALL	ACCURACY	F- MEASURE
COVOLUTION NEURAL NETWORK	0.768	0.764	0.76	0.764