

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

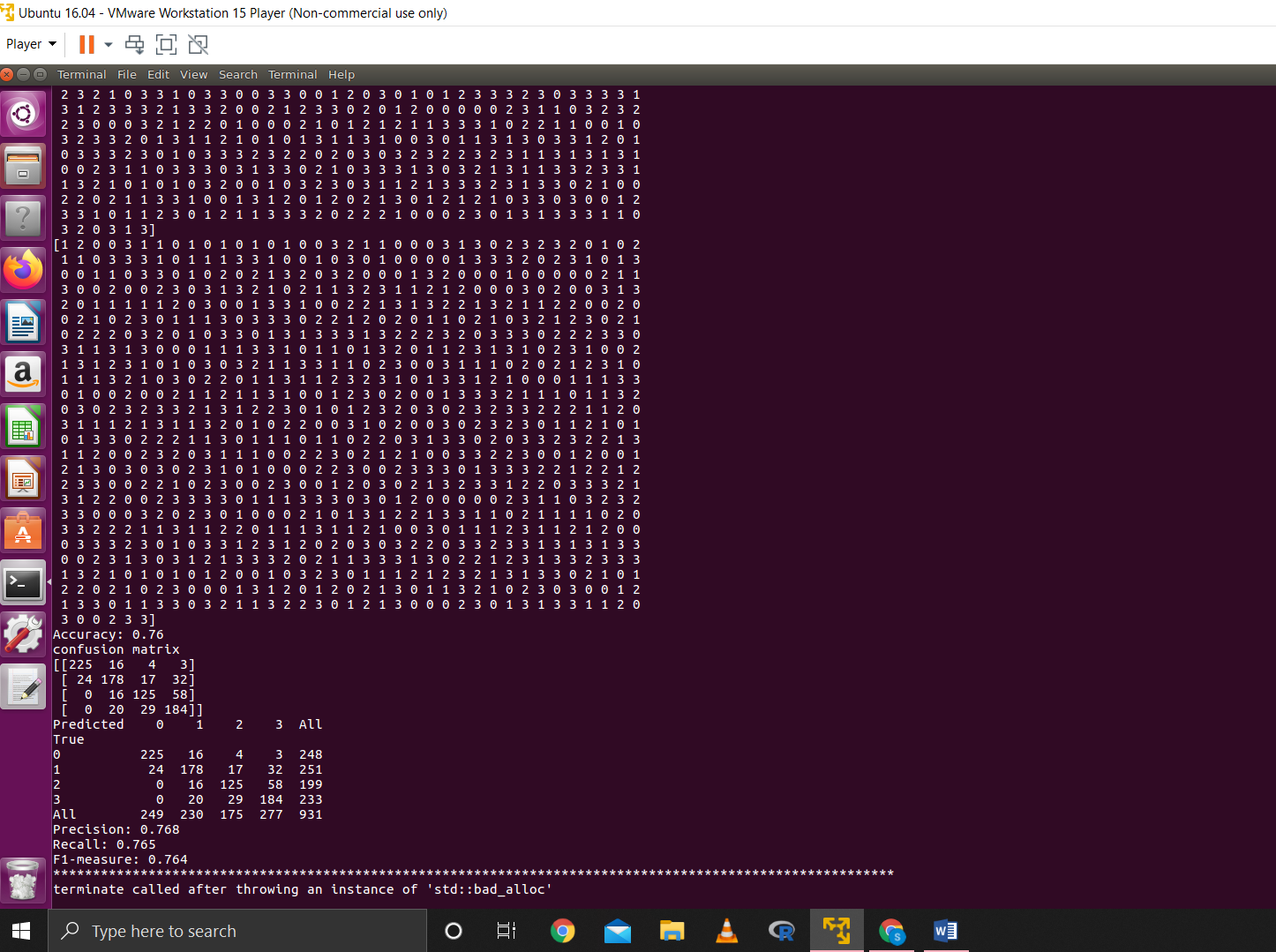
print y\_true

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 |