Project2 商品分類

1. 簡介

Training data 共有180張jpg圖片，共18樣商品，每張為256 \*256 pixels組成，其label為180\*1之向量；Validation data共有50張jpg圖片，共18樣商品，每張的尺寸為256 \*256 pixels，其label為50\*1之向量。將數據運用Tensorflow建立模型並訓練，。

1. 流程
2. 參數設置

learning\_rate = 5\*1e-3

num\_steps = 1000 (訓練1000次)

batch\_size = 90 (將資料集任取90筆為一組)

display\_step = 100 (每100次顯示一次準確率)

1. 圖片影像處理

先將圖片尺寸調整為32\*32pixels，再放入模型進行訓練，驗證集也相同調整為32\*32pixels；由於圖片為彩色三通道，再轉為灰階圖片。

1. 調整資料型態

將訓練集180張32\*32 pixels的圖片拉成1\*1024 pixels，並放入180\*1024的矩陣，以便訓練使用；label 轉為獨熱編碼，用來驗證輸出結果。將驗證集50張32\*32 pixels的圖片拉成1\*1024 pixels，並放入50\*1024的矩陣，以便取用；label 轉為獨熱編碼，用來驗證輸出結果。

1. 架構神經網路

建構卷積神經網路，選用Tensorflow套件，架構依序輸入層、2個隱藏層(包括卷積層與池化層)，再經過平滑層、ReLU激活函數，避免Overfitting 故Dropout 一半的特徵，最後輸出層以Softmax函數得到預測結果。運用cross\_entropy來得到loss值再修正Weights和biases。訓練數據批量每次90筆，採隨機選取180筆中90筆，

1. 結論

為縮短訓練時間、減少參數，故將圖片尺寸調整為32\*32 pixels，並採取灰階而非三通道像素值都放入訓練模型。模型尚有問題，準確率僅6%，無法突破。訓練分類器時有較多圖片、且有變化越好，當資料不足時能夠使用旋轉、裁切、增加噪點等方式增加資料量，但這次圖片種類繁多，故無選擇此方式。

1. 心得

對於模型的了解應更熟悉，在放入資料時較不容易產生錯誤，當準確率在經過幾次訓練後皆無提高，可能就是模型有問題，須從放入資料逐步確認，甚至驗證資料也可能有問題。

圖片為彩色，內容也較文字複雜，需訓練較多變數，但訓練集數目不多，且商品種類繁多，背景或是物品大小都會影響訓練與辨識，訓練較難提高準確率，此時只能對於圖片再進行處理增加資料量，或是拉長時間訓練較多變數，可能會有較好結果。

1. 參考來源

<https://blog.csdn.net/guyuealian/article/details/83473298>

<https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/703413/>

<https://medium.com/@CinnamonAITaiwan/cnn%E5%85%A5%E9%96%80-%E5%9C%96%E5%83%8F%E5%A2%9E%E5%BC%B7-fa654d36dafc>

<https://www.cnblogs.com/sdu20112013/p/10739946.html>

1. 程式碼：

import tensorflow as tf

import numpy as np

import time

import pandas as pd

from os import listdir

from os.path import isfile, isdir, join

from PIL import Image

import cv2

import math

import random

def compute\_accuracy(v\_xs, v\_ys):

global prediction

y\_pre = sess.run(prediction, feed\_dict={xs: v\_xs, keep\_prob: 1})

correct\_prediction = tf.equal(tf.argmax(y\_pre,1), tf.argmax(v\_ys,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

result = sess.run(accuracy, feed\_dict={xs: v\_xs, ys: v\_ys, keep\_prob: 1})

return result

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1,shape=shape)

return tf.Variable(initial)

def conv2d(x,W):

return tf.nn.conv2d(x,W,strides=[1,1,1,1],padding = 'SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')

df\_t = pd.read\_excel("./Training-label.xlsx",header=None)

df\_temp = pd.get\_dummies(df\_t[0]) #One-hot編碼

y\_t = np.array(df\_temp)

size = 32

img\_tra = np.zeros((180,32\*32))

mypath = "./Training"

files = listdir(mypath)

j=0

for i,f in enumerate(files):

fullpath = join(mypath, f)

#print(f)

ro = False

if isfile(fullpath):

im = cv2.resize(cv2.imread(fullpath,0),(size,size))

image = cv2.resize(cv2.imread(fullpath),(size,size))

retval, im\_bin = cv2.threshold(im,215, 255, cv2.THRESH\_BINARY\_INV)

img = np.array(im,dtype=np.float32).reshape(size\*\*2)

img\_tra[j,:]=1-img/255

j+=1

df\_v = pd.read\_excel('./Validation-label.xlsx',header=None)

df\_v\_temp = pd.get\_dummies(df\_v[0])

y\_v = np.array(df\_v\_temp)

img\_val = np.zeros((50,size\*size))

mypath = "./Validation"

files = listdir(mypath)

j=0

for i,f in enumerate(files):

fullpath = join(mypath, f)

#print(f)

ro = False

if isfile(fullpath):

im = cv2.resize(cv2.imread(fullpath,0),(size,size))

image = cv2.resize(cv2.imread(fullpath),(size,size))

retval, im\_bin = cv2.threshold(im,215, 255, cv2.THRESH\_BINARY\_INV)

img = np.array(im,dtype=np.float32).reshape(size\*\*2)

img\_val[j,:]=1-img/255

j+=1

img\_pix=32

xs=tf.placeholder(tf.float32,[None,img\_pix\*img\_pix])/255. #32\*32=1024

ys=tf.placeholder(tf.float32,[None,18])

keep\_prob = tf.placeholder(tf.float32)

x\_image = tf.reshape(xs,[-1,img\_pix,img\_pix,1])

# Parameters

learning\_rate = 5\*1e-3

num\_steps = 1000

batch\_size = 90

display\_step = 100

model\_path = "my\_project2/save\_net.ckpt"

##conv1 layer##

W\_conv1=weight\_variable([5,5,1,64])

b\_conv1=bias\_variable([64])

#print(W\_conv1)

#print(b\_conv1)

hid\_conv1 = tf.nn.relu(conv2d(x\_image,W\_conv1)+b\_conv1) #output size img\_pix\*img\_pix\*32

hid\_pool1 =max\_pool\_2x2(hid\_conv1) #output size 16\*16\*64

##conv2 layer##

W\_conv2=weight\_variable([5,5,64,128])

b\_conv2=bias\_variable([128])

hid\_conv2 = tf.nn.relu(conv2d(hid\_pool1,W\_conv2)+b\_conv2) #output size 14\*14\*64

hid\_pool2 =max\_pool\_2x2(hid\_conv2) #output size 8\*8\*128

##func1 layer##

W\_fc1 = weight\_variable([8\*8\*128,2048])

b\_fc1 = bias\_variable([2048])

#[n\_samples,7,7,64]->>[n\_samples,7\*7\*64]

hid\_pool2\_flat = tf.reshape(hid\_pool2,[-1,8\*8\*128])

hid\_fc1 = tf.nn.relu(tf.matmul(hid\_pool2\_flat,W\_fc1)+b\_fc1)

hid\_fc1\_drop = tf.nn.dropout(hid\_fc1,keep\_prob)

##func2 layer##

W\_fc2 = weight\_variable([2048,18])

b\_fc2 = bias\_variable([18])

y\_conv = tf.matmul(hid\_fc1\_drop, W\_fc2) + b\_fc2

prediction = tf.nn.softmax(y\_conv)

cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(labels=ys, logits=prediction))

train\_step = tf.train.AdamOptimizer(learning\_rate=learning\_rate).minimize(cross\_entropy)

correct\_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(ys, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))

init = tf.global\_variables\_initializer()

saver = tf.train.Saver()

tStart = time.time()#計時開始

list1 = [0]

for n in range(1,180): list1=list1+[n]

with tf.Session() as sess:

sess.run(init)

for i in range(num\_steps+1):

#images\_batch,labels\_batch = get\_batch\_images(img\_tra,y\_t,batch\_size,labels\_nums=18,one\_hot=False,shuffle=False)

#images\_batch,labels\_batch=images\_batch.eval(),labels\_batch.eval()

list1 = [0]

for n in range(1,180):

list1=list1+[n]

list2 = random.sample(list1, len(list1))

list2=list2[0:batch\_size]

#print(len(list2))

images\_batch=img\_tra[i%180,:]

labels\_batch=y\_t[i%180,:]

for ind in list2:

images\_batch = np.vstack([images\_batch,img\_tra[ind,:]])

labels\_batch = np.vstack([labels\_batch,y\_t[ind,:]])

#print(images\_batch.shape)

#print(labels\_batch.shape)

sess.run(train\_step,feed\_dict={xs:images\_batch,ys:labels\_batch, keep\_prob: 0.5})

if i%display\_step==0 or i==num\_steps:

train\_accuracy = compute\_accuracy(img\_val,y\_v)

print("step %d, training accuracy %g"%(i, train\_accuracy))

#print(batch\_xs)

save\_path = saver.save(sess, model\_path)

print("Model saved in file: %s" % save\_path)

tEnd = time.time()

print("It cost %f sec" % (tEnd - tStart))