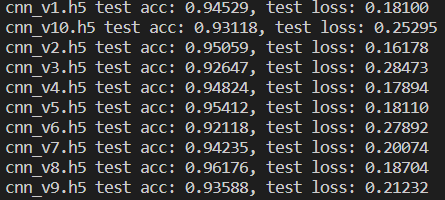
演算法Term\_Project報告 大氣4A 106601015 黃展皇

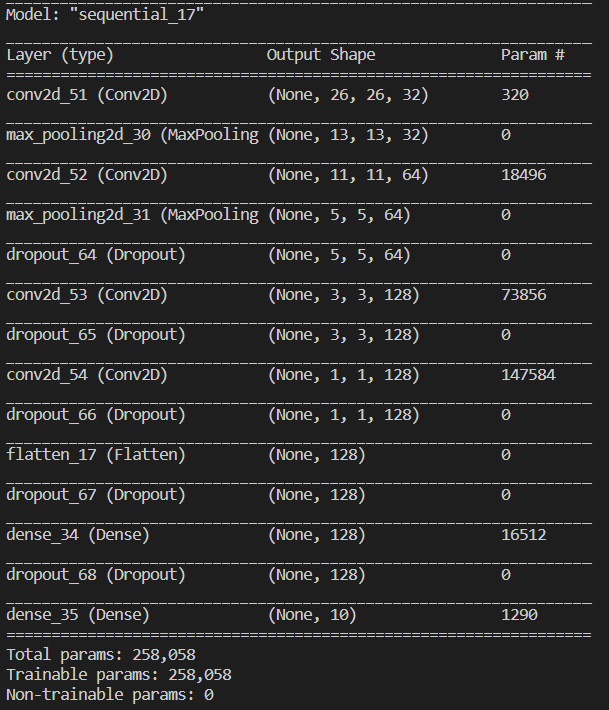
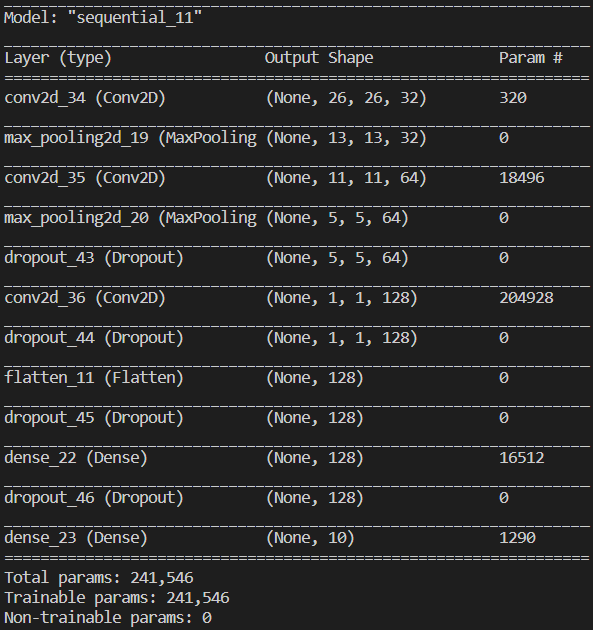
1. 中文手寫辨識準確率(accuracy)，以截圖方式呈現：



如圖，共做了10種版本的CNN模型，分別測試深度、kernel大小及其排列組合、Dropout大小、有無Pooling等等。

其中acc表現最好的是v8(下左圖)，得到0.96176的準確度

loss表現最好的是v2(下右圖) ，loss下降到0.16178

1. Source code之逐行解釋：下方用中文做詳細解釋

# import the following packages

import os

import random

import numpy as np

from PIL import Image

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Flatten

from tensorflow.keras.layers import Conv2D, MaxPooling2D

from tensorflow.keras.models import save\_model, load\_model

from tensorflow.keras.applications import ResNet152V2, ResNet50

from tensorflow.keras.utils import to\_categorical

from matplotlib import pyplot as plt

import datetime

🡪以上import需要用到的套件，版本放在最後附錄

#x, y(label) preprocess

🡪輸入datapath路徑(只會是train\_image/test\_image)，並且os.walk對datapath

內的所有圖片做讀取、nomalize、reshape，並對labels做to\_categorical，最後return data\_x, data\_y分別對應資料以及labels

def data\_x\_y\_preprocess(datapath): 🡪定義方法，輸入資料路徑

    print('data\_x\_y\_preprocess init') 🡪調用到這方法的測試

    img\_row, img\_col = 28, 28 🡪定義圖片大小

    data\_x = np.zeros((img\_row, img\_col)).reshape(1, img\_row, img\_col)

🡪定義空圖片array，形狀是(1, 28, 28)(數量, 高, 寬)

    pic\_counter = 0 🡪初始化圖片數量=0

    data\_y = [] 🡪圖形分類結果

    num\_class = 10 🡪分類結果總數(0~9)

    for root, dirs, files in os.walk(datapath): 🡪走訪datapath

        for f in files: 🡪對所有圖片檔案迴圈

            data\_y.append(int(root.split('\\')[-1]))

🡪f的root尾段即為該數字分類類別

            fullpath = os.path.join(root, f) 🡪更新檔案完整位置

            img = Image.open(fullpath) 🡪用Image套件開圖片

            img = (np.array(img)/255).reshape(1, 28, 28) #nomalize

🡪做nomalize，0~255大小同除255即可，再reshape成data\_x大小

            data\_x = np.vstack((data\_x, img))

🡪在shape[0]拼接data\_x與img

            pic\_counter += 1 🡪圖片數量+1

    data\_x = np.delete(data\_x, 0, axis=0) 🡪刪除原本的空array

    data\_x = data\_x.reshape(pic\_counter, img\_row, img\_col, 1)

🡪擴充shape多一維

    #data\_y = np\_utils.to\_categorical(data\_y, num\_class)

    data\_y = to\_categorical(data\_y, num\_class)

🡪將data\_y list做to\_categorical(做one-hot encoding)

    return data\_x, data\_y 🡪回傳x, y

接下來的create\_cnn\_model\_{1~10}系列皆為回傳一個Sequential模型的方法：

建立model為Sequential模型，並不斷.add堆疊Conv2D、MaxPooling2D、Dropout、Flatten等層做組合排列，最後Dense層用'softmax'方法做分類，完成模型建構，.Summary()觀察模型架構，最後回傳model

# basic version

def create\_cnn\_model\_v1(): 🡪定義方法version1

    model = Sequential()

    model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.1))

    model.add(Flatten())

    model.add(Dropout(0.1))

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.25))

    model.add(Dense(10, activation='softmax'))

    model.summary()

    return model

# deeper version

def create\_cnn\_model\_v2(): 🡪定義方法version2

    model = Sequential()

    model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.1))

    model.add(Conv2D(128, kernel\_size=(5, 5), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Flatten())

    model.add(Dropout(0.1))

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.25))

    model.add(Dense(10, activation='softmax'))

    model.summary()

    return model

# kernel\_size 7->5->3 version

def create\_cnn\_model\_v3(): 🡪定義方法version3

    model = Sequential()

    model.add(Conv2D(32, kernel\_size=(7, 7), activation='relu', input\_shape=(28, 28, 1)))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.1))

    model.add(Conv2D(64, kernel\_size=(5, 5), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.1))

    model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Flatten())

    model.add(Dropout(0.1))

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.25))

    model.add(Dense(10, activation='softmax'))

    model.summary()

    return model

# just test

def create\_cnn\_model\_v4(): 🡪定義方法version4

    model = Sequential()

    model.add(Conv2D(16, kernel\_size=(5, 5), activation='relu', padding='same', input\_shape=(28, 28, 1)))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.1))

    model.add(Conv2D(36, kernel\_size=(5, 5), activation='relu', padding='same'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.1))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.5))

    model.add(Dense(10, activation='softmax'))

    model.summary()

    return model

# just test2

def create\_cnn\_model\_v5(): 🡪定義方法version5

    model = Sequential()

    model.add(Conv2D(32, (5,5), activation="relu", padding="same", input\_shape=(28,28,1)))

    model.add(Conv2D(32, (5,5), activation="relu", padding="same", input\_shape=(28,28,1)))

    model.add(MaxPooling2D(pool\_size=(2,2)))

    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3,3), activation="relu", padding="same"))

    model.add(Conv2D(64, (3,3), activation="relu", padding="same"))

    model.add(MaxPooling2D(pool\_size=(2,2)))

    model.add(Dropout(0.25))

    model.add(Flatten())

    model.add(Dense(256, activation="relu"))

    model.add(Dropout(0.5))

    model.add(Dense(10, activation="softmax"))

    model.summary()

    return model

# just test3

def create\_cnn\_model\_v6(): 🡪定義方法version6

    model = Sequential()

    model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)))

    model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.25))

    model.add(Flatten())

    model.add(Dense(128, activation="relu"))

    model.add(Dropout(0.5))

    model.add(Dense(10, activation="softmax"))

    model.summary()

    return model

# deeper 3 3 3 version

def create\_cnn\_model\_v7(): 🡪定義方法version7

    model = Sequential()

    model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.1))

    model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Flatten())

    model.add(Dropout(0.1))

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.25))

    model.add(Dense(10, activation='softmax'))

    model.summary()

    return model

# more deeper 3 3 3 3 version

def create\_cnn\_model\_v8(): 🡪定義方法version8

    model = Sequential()

    model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.1))

    model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Flatten())

    model.add(Dropout(0.1))

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.25))

    model.add(Dense(10, activation='softmax'))

    model.summary()

    return model

# no MaxPooling2D, more deeper 3 3 3 3 3 version

def create\_cnn\_model\_v9(): 🡪定義方法version9

    model = Sequential()

    model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Flatten())

    model.add(Dropout(0.1))

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.25))

    model.add(Dense(10, activation='softmax'))

    model.summary()

    return model

# more deeper 3->3->5->7 version

def create\_cnn\_model\_v10(): 🡪定義方法version10

    model = Sequential()

    model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Conv2D(128, kernel\_size=(5, 5), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Conv2D(128, kernel\_size=(7, 7), activation='relu'))

    model.add(Dropout(0.1))

    model.add(Flatten())

    model.add(Dropout(0.1))

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.25))

    model.add(Dense(10, activation='softmax'))

    model.summary()

    return model

'''

def create\_ResNet50():

    model = ResNet50(include\_top=False, weights='imagenet', input\_tensor=None, input\_shape=(28, 28, 1)) #[(None, 224, 224, 3)

    #model = ResNet50()

    model.summary()

    return model

'''

def show\_training\_curve(train\_history):

🡪定義接收訓練歷史物件的方法，畫出並儲存loss下降曲線

    plt.plot(train\_history.history['loss']) 🡪畫出loss曲線

    plt.plot(train\_history.history['val\_loss']) 🡪畫出val\_loss曲線

    plt.title('train history') 🡪定義title

    plt.ylabel('loss') 🡪ylabel為loss

    plt.xlabel('epochs') 🡪xlabel為epochs

    plt.legend(['loss', 'val\_loss'], loc='upper left') 🡪上圖標在左上區

    plt.savefig(os.path.join('.', 'term\_project', datetime.datetime.now().strftime("%Y\_%m\_%d\_\_%H\_%M\_%S"))) 🡪用給定名字存在給定路徑

    #plt.show()

def main(operator='train'): 🡪主程式，有train與test兩種動作型態

    print('-----main {} init-----'.format(operator)) 🡪驗證用

    term\_project\_path = os.path.join('C:\\', 'Users', 'user', 'Desktop', 'algorithm', 'term\_project') 🡪定義路徑

    model\_dict = { 🡪用dict.儲存模型

        'cnn\_v1':create\_cnn\_model\_v1(),

        'cnn\_v2':create\_cnn\_model\_v2(),

        'cnn\_v3':create\_cnn\_model\_v3(),

        'cnn\_v4':create\_cnn\_model\_v4(),

        'cnn\_v5':create\_cnn\_model\_v5(),

        'cnn\_v6':create\_cnn\_model\_v6(),

        'cnn\_v7':create\_cnn\_model\_v7(),

        'cnn\_v8':create\_cnn\_model\_v8(), #best

        'cnn\_v9':create\_cnn\_model\_v9(),

        'cnn\_v10':create\_cnn\_model\_v10(),

        #'ResNet50':create\_ResNet50(),

    }

    if operator == 'train': 🡪若為train動作：

        train\_datapath = os.path.join(term\_project\_path, 'train\_image')

🡪定義train\_datapath(一個系統路徑，裡面有多個訓練圖片子資料夾樹)

        # get train data

        if os.path.isfile(os.path.join(term\_project\_path, 'train\_data\_x.npy')):

🡪若.npy檔案存在，則直接np.load進來分train\_data\_x, train\_data\_y

            train\_data\_x, train\_data\_y = np.load(os.path.join(term\_project\_path, 'train\_data\_x.npy')), np.load(os.path.join(term\_project\_path, 'train\_data\_y.npy'))

        else:

🡪若.npy檔案不存在，則調用data\_x\_y\_preprocess取得資料

            train\_data\_x, train\_data\_y = data\_x\_y\_preprocess(train\_datapath)

            np.save(os.path.join(term\_project\_path, 'train\_data\_x.npy'), train\_data\_x) 🡪將資料儲存為.npy檔案(下次就不用重新處理資料)

            np.save(os.path.join(term\_project\_path, 'train\_data\_y.npy'), train\_data\_y) 🡪將資料儲存為.npy檔案(下次就不用重新處理資料)

        print('---> train\_data\_x.shape, train\_data\_y.shape:', train\_data\_x.shape, train\_data\_y.shape) 🡪驗證用

        # loop in model\_dict

        for model\_name, model in model\_dict.items(): 🡪for所有模型迴圈

            if os.path.isfile(os.path.join(term\_project\_path, '{}.h5'.format(model\_name))):

🡪如果.h5(模型網路結果)存在則呈現已訓練過

                print('{} has been train!!!'.format(model\_name))

            else:

🡪如果找不到.h5檔案(還沒訓練)：

                model.compile(

                    loss='categorical\_crossentropy',

                    optimizer='adam',

                    metrics=['accuracy']

                )

🡪 compile模型(loss用類別的crossentropy，優化器用adam，測量方式只放準確度)

                train\_history = model.fit(

                    train\_data\_x, train\_data\_y,

                    batch\_size=32,

                    epochs=30,

                    verbose=1,

                    validation\_split=0.1

                )

🡪 fit模型(當然用train\_x/y, 設定基本的batch\_size=32, epochs=30, 其中設定validation\_split=0.1，會從train資料裡面選0.1比例的資料做驗證)

                show\_training\_curve(train\_history)

🡪繪製訓練時的loss曲線

                save\_model(model, os.path.join(term\_project\_path, '{}.h5'.format(model\_name)))

🡪儲存訓練好的模型

    elif operator == 'test': 🡪若為test動作

        test\_datapath = os.path.join(term\_project\_path, 'test\_image')

🡪定義test\_datapath(一個系統路徑，裡面有多個測試圖片子資料夾樹)

        if os.path.isfile(os.path.join(term\_project\_path, 'test\_data\_x.npy')):

🡪若.npy檔案存在，則直接np.load進來分test\_data\_x, test\_data\_y

            test\_data\_x, test\_data\_y = np.load(os.path.join(term\_project\_path, 'test\_data\_x.npy')), np.load(os.path.join(term\_project\_path, 'test\_data\_y.npy'))

        else:

🡪若.npy檔案不存在，則調用data\_x\_y\_preprocess取得資料

            test\_data\_x, test\_data\_y = data\_x\_y\_preprocess(test\_datapath)

            np.save(os.path.join(term\_project\_path, 'test\_data\_x.npy'), test\_data\_x)

🡪將資料儲存為.npy檔案(下次就不用重新處理資料)

            np.save(os.path.join(term\_project\_path, 'test\_data\_y.npy'), test\_data\_y)

🡪將資料儲存為.npy檔案(下次就不用重新處理資料)

        print('test\_data\_x.shape, test\_data\_y.shape:', test\_data\_x.shape, test\_data\_y.shape)

        #for all h5

        for file\_name in os.listdir(term\_project\_path):🡪for所有.h5模型

            if file\_name.split('.')[-1] == 'h5':

                model = load\_model(os.path.join(term\_project\_path, file\_name)) 🡪讀進model

                score = model.evaluate(test\_data\_x, test\_data\_y, verbose=0) 🡪 用test\_data\_x, test\_data\_y 對model evaluate分數

                print('{} test acc: {:.5f}, test loss: {:.5f}'.format(file\_name, score[1], score[0]))

🡪秀出測試的準確度與loss

    else:

        print('operator error!!!')

if \_\_name\_\_ == '\_\_main\_\_':

    main(operator='train') 🡪對所有model\_dict中的模型做訓練

    main(operator='test') 🡪對所有model\_dict中的模型做測試

補充以上程式所需要的套件及版本：

tensorflow (-gpu)=2.4.0、pillow =7.2.0、numpy =1.19.0、matplotlib=3.2.2

電腦配置：~~垃圾~~windows10、conda env、tensorflow-gpu、GeForce GTX 1050 Ti 4GB、DDR4-2666 8G\*2

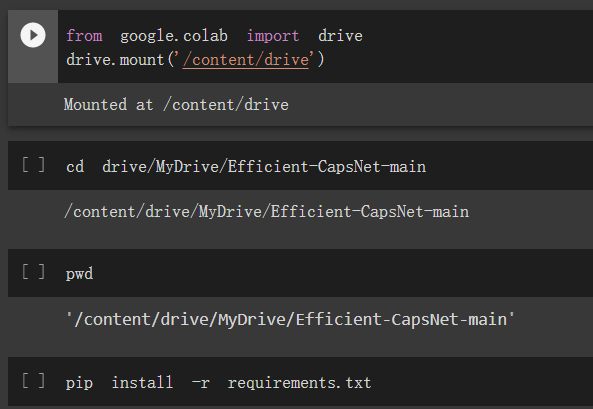
另外補充：

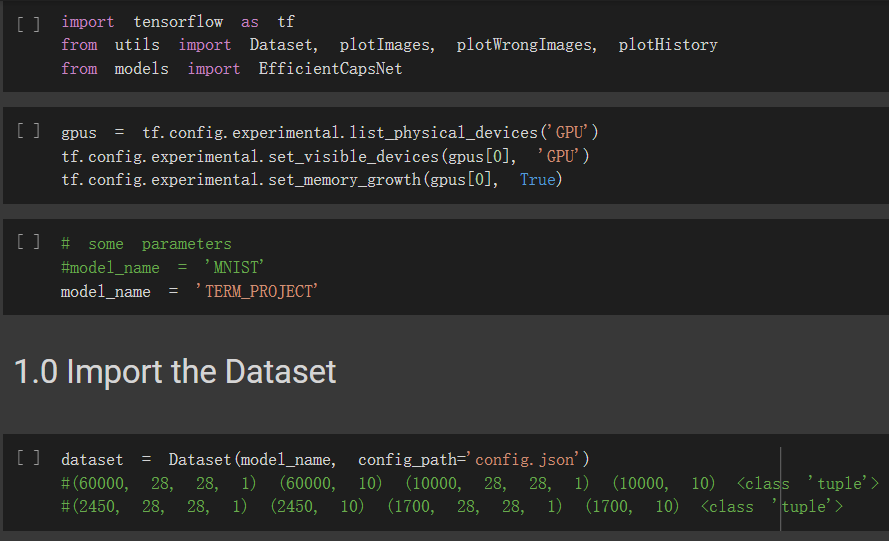
另外自行參考2021年的論文”EFFICIENT-CAPSNET: CAPSULE NETWORK WITH SELF-ATTENTION ROUTING”(https://arxiv.org/abs/2101.12491)

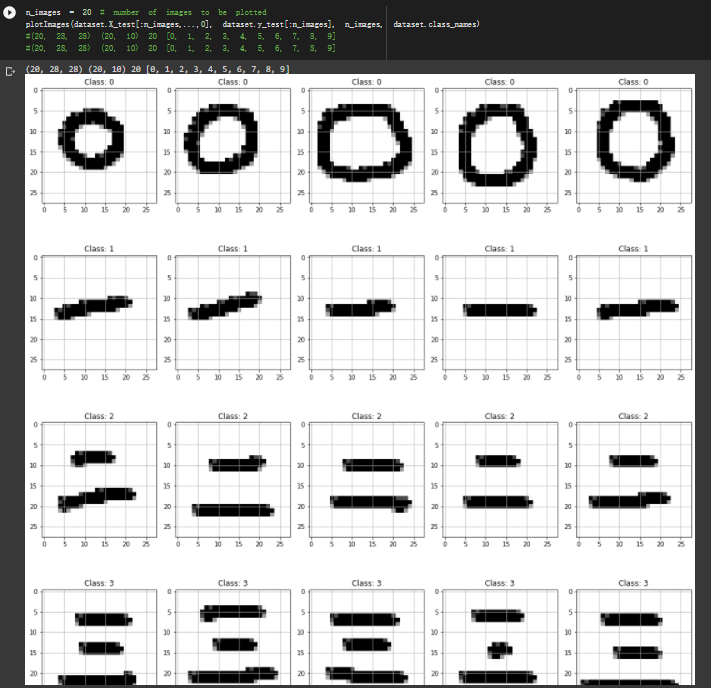
實作Efficient-CapsNet用於本次的中文手寫辨識。

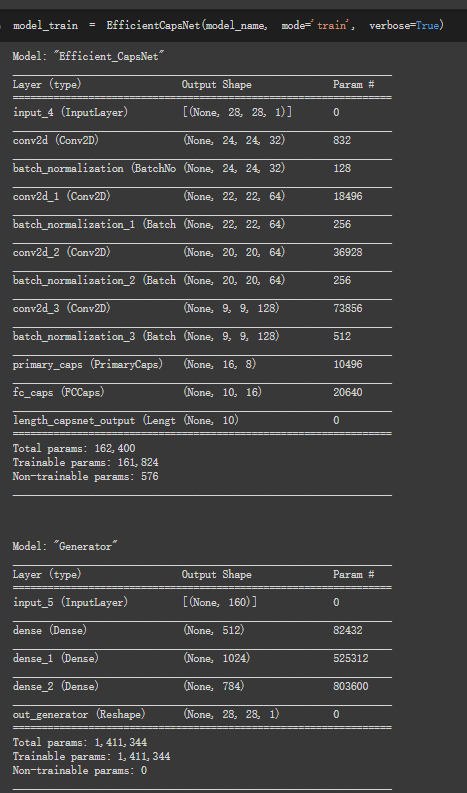
附件中有efficient\_capsnet\_train.ipynb/efficient\_capsnet\_test.ipynb分別用於呼叫寫好的preprocess或model等等方式並加以訓練/測試，詳細程式碼與註解附於ipynb檔案內。

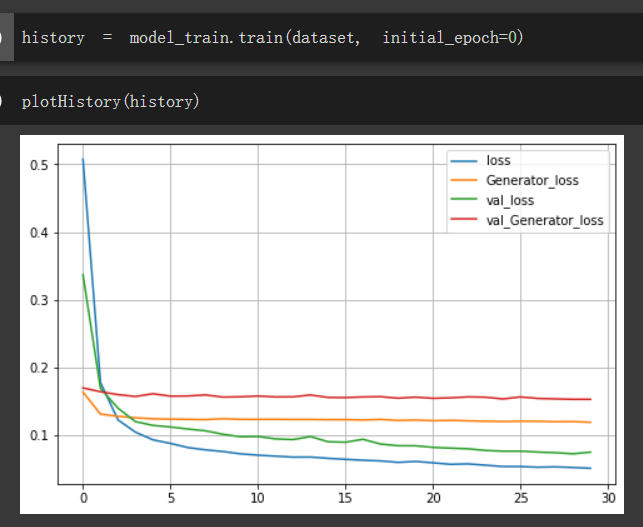
環境採用colab(tf 2.5以上，GPU加速，依賴套件如requirements.txt所示)

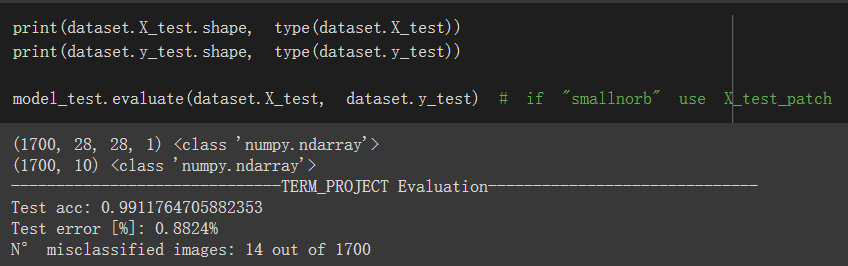
前處理與環境設置

套件與資料集設置

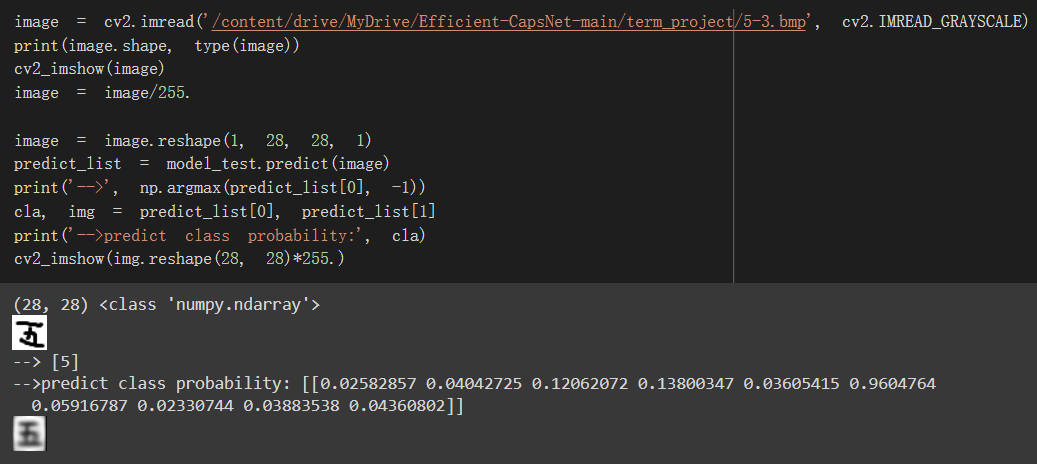
可視化

模型參數

訓練結果與loss曲線



測試結果：準確度99.118%，1700個測試案例中只錯誤14個案例



單一個案測試，預測96%為”五”，生成器產生圖片雖略帶模糊但能看出五