

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：

```
cnn_v1.h5 test acc: 0.94529, test loss: 0.18100
cnn_v10.h5 test acc: 0.93118, test loss: 0.25295
cnn_v2.h5 test acc: 0.95059, test loss: 0.16178
cnn_v3.h5 test acc: 0.92647, test loss: 0.28473
cnn_v4.h5 test acc: 0.94824, test loss: 0.17894
cnn_v5.h5 test acc: 0.95412, test loss: 0.18110
cnn_v6.h5 test acc: 0.92118, test loss: 0.27892
cnn_v7.h5 test acc: 0.94235, test loss: 0.20074
cnn_v8.h5 test acc: 0.96176, test loss: 0.18704
cnn_v9.h5 test acc: 0.93588, test loss: 0.21232
```

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

Model: "sequential_17"		
Layer (type)	Output Shape	Param #
conv2d_51 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_30 (MaxPooling)	(None, 13, 13, 32)	0
conv2d_52 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_31 (MaxPooling)	(None, 5, 5, 64)	0
dropout_64 (Dropout)	(None, 5, 5, 64)	0
conv2d_53 (Conv2D)	(None, 3, 3, 128)	73856
dropout_65 (Dropout)	(None, 3, 3, 128)	0
conv2d_54 (Conv2D)	(None, 1, 1, 128)	147584
dropout_66 (Dropout)	(None, 1, 1, 128)	0
flatten_17 (Flatten)	(None, 128)	0
dropout_67 (Dropout)	(None, 128)	0
dense_34 (Dense)	(None, 128)	16512
dropout_68 (Dropout)	(None, 128)	0
dense_35 (Dense)	(None, 10)	1290
Total params: 258,058		
Trainable params: 258,058		
Non-trainable params: 0		

Model: "sequential_11"		
Layer (type)	Output Shape	Param #
conv2d_34 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_19 (MaxPooling)	(None, 13, 13, 32)	0
conv2d_35 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_20 (MaxPooling)	(None, 5, 5, 64)	0
dropout_43 (Dropout)	(None, 5, 5, 64)	0
conv2d_36 (Conv2D)	(None, 1, 1, 128)	204928
dropout_44 (Dropout)	(None, 1, 1, 128)	0
flatten_11 (Flatten)	(None, 128)	0
dropout_45 (Dropout)	(None, 128)	0
dense_22 (Dense)	(None, 128)	16512
dropout_46 (Dropout)	(None, 128)	0
dense_23 (Dense)	(None, 10)	1290
Total params: 241,546		
Trainable params: 241,546		
Non-trainable params: 0		

2. 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():
    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
```

→定義方法 version2

kernel_size 7->5->3 version

```
def create_cnn_model_v3():
    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))
```

→定義方法 version3

```

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", inpu
t_shape=(28,28,1)))
    model.add(Conv2D(32, (5,5), activation="relu", padding="same", inpu
t_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():
    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))

```

→定義方法 version8

```

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_proje
ct_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_data
path)

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_d
ata_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'.f
ormat(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, '{}.h
5'.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.
numpy')):
    →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