演算法期末報告　機械 4A 106303013 陳日昇

一、題目:深度學習中文手寫數字辨識

輸入: 中文手寫數字資料集(dataset)

輸出: 中文手寫數字辨識準確率(accuracy)

二、程式碼

1. trainmidel.py : 訓練模型程式
    - 引入函式庫

```python
from PIL import Image
from keras.utils import np_utils
import numpy as np
import matplotlib.pyplot as plt
import os
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
```

- 副函式
    - Read_FilePath(file_path): 讀取檔案路徑及標籤

```python
def Read_FilePath(file_path):  #讀取bmp型別的檔案
    filepaths = []  # 儲存圖片路徑
    labels = []     # 儲存標籤
    for root, _, files in os.walk(file_path): # 走過file_path下所有資料夾
        for file in files:
            if os.path.splitext(file)[1] == '.bmp': # 當遇到.bmp檔時
                filepaths.append(os.path.join(root, file)) # 保存圖片路徑
                labels.append(root[-1]) # 紀錄圖片資料夾，以作為標籤
    return filepaths, labels # 回傳圖片路徑及標籤
```

- Write_ImageData(filepaths): 製作圖片資料流

```python
def Write_ImageData(filepaths): # 製作圖片資料
    imgdata = [] # 儲存圖片資料
    for filepath in filepaths:
        image = np.array(Image.open(filepath)) # 開圖片檔
        imgdata.append(image) # 圖片檔案保存
    return np.array(imgdata) # 回傳圖片資料
```

- Parameter_ser(model): 神經網路參數設定

```python
def Parameter_set(model):
    #建立卷積層1
    # 輸入數字影像 28x28 的大小，執行一次卷積運算，產生 16 個影像，卷積運算不會改變影像大小，結果還是 28x28
    model.add(  Conv2D(filters = 16,      # 建立 16 個 filter weight
                kernel_size = (5,5),      # 每一個濾鏡大小為 5x5
                padding = 'same',         # 讓卷積運算產生的影像大小不變
                input_shape = (28,28,1), # 第1, 2 維，是輸入的影像形狀 28x28，第 3 維，因為是單色灰階影像，所以是 1
                activation='relu'))       # 設定 ReLU 激活函數
    #建立池化層
    model.add(MaxPooling2D(pool_size = (2, 2)))  # 執行第一次縮減取樣，將16個 28x28 影像，縮小為16個 14x14 的影像

    # 建立卷積層2
    # 執行第二次卷積運算，將原本的 16 個影像，轉換為 36 個影像，卷積運算不會改變影像大小，結果還是 14x14
    model.add(  Conv2D(filters = 36,      # 建立 36 個 filter weight
                kernel_size = (5,5),      # 每一個濾鏡大小為 5x5
                padding = 'same',         # 讓卷積運算產生的影像大小不變
                activation = 'relu'))     # 設定 ReLU 激活函數

    # 建立池化層2，並加入Dropout 避免 overfitting
    model.add(MaxPooling2D(pool_size = (2, 2))) # 將 36 個 14x14 的影像，縮小為 36 個 7x7 的影像
    model.add(Dropout(0.25))

    # 建立神經網路 (平坦層, 隱藏層, 輸出層)
    model.add(Flatten()) # 建立平坦層
    model.add(Dense(128, activation = 'relu')) # 建立隱藏層，共有128個神經元
    model.add(Dropout(0.5)) # 每次訓練迭代時，會隨機在神經網路中，放棄50%的神經元，以避免overfitting
    # 建立輸出層
    # 共 10 個神經元，對應 0~9 共 10 個數字，並使用 softmax 激活函數進行轉換
    model.add(Dense(10, activation = 'softmax')) # 可將神經元的輸出，轉換為預測每一個數字的機率
    print(model.summary())
```

- Train_Model(model): 訓練模型

```python
def Train_Model(model):
    ## 訓練模型：輸入 features, label，執行 100 次訓練週期
    model.compile(  loss = 'categorical_crossentropy',
                    optimizer = 'adam',
                    metrics = ['accuracy'])

    train_history = model.fit(x = x_TrainData,
                              y = y_TrainData,
                              validation_split = 0.4,   # 60%為訓練資料, 40%驗證資料
                              epochs = 100,             # 訓練100次
                              batch_size = 500,         # 每一批次500筆資料
                              verbose = 2)              # 顯示訓練過程

    plot_train_history(train_history, 'accuracy', 'val_accuracy')
    plot_train_history(train_history, 'loss', 'val_loss')
```

- Save_Model(model): 保存模型

```python
def Save_Model(model):
    print("------正在保存模型------")
    model.save('model.h5')
```

- plot_train_history(train, train_acc, test_acc): 畫出精度歷程圖

```python
def plot_train_history(train_history, train_acc, test_acc):
    plt.clf() # 清除圖片
    plt.plot(train_history.history[train_acc]) # 畫出訓練之準度
    plt.plot(train_history.history[test_acc])  # 畫出測試之準度
    plt.title('Train History') # 標題
    plt.ylabel('Accuracy')     # y軸標題
    plt.xlabel('Epoch')        # x軸標題
    plt.legend(['train', 'test'], loc='upper left')
    plt.show() # 畫圖
```

- 主函式

```python
if __name__ == '__main__':
    ## Step 1: 讀取圖片資料
    file_path  = "train_image"
    tf_file_name = "data.tfrecords"
    filepaths, labels = Read_FilePath(file_path)   #將讀到的圖案路徑存起來
    imgdata = Write_ImageData(filepaths)

    ## Step 2: 圖片資料預處理，產生 feature 及 label
    x_TrainData = imgdata.reshape(imgdata.shape[0],28,28,1).astype('float32') # 將資料改成4維陣列
    x_TrainData_normalize = x_TrainData / 255      # 將 features 標準化
    y_TrainData = np_utils.to_categorical(labels)  # 以 Onehot Encoding 轉換 label

    ## Step 3: 建立模型
    # 線性堆疊模型
    model = Sequential() # 建立模型
    Parameter_set(model) # 設定參數
    Train_Model(model)   # 訓練模型
    Save_Model(model)    # 保存模型
```

2. testmodel.py 測式模型程式
   - 程式

```python
from keras.models import load_model
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

if __name__ == '__main__':
    model = load_model('model.h5') # 讀取儲存之模型
    imagepath = "./test_image/104201529/5/5(1).bmp" # 測試圖片路徑
    # imagepath = "2.bmp"
    image = Image.open(imagepath) # 開檔案
    pic = image # 額外儲存圖檔
    x_TrainData = np.array(image).reshape(1,28,28,1).astype('float32') # 將圖檔資料改成4維陣列
    prob = model.predict(x_TrainData) # 模型預測機率
    predict = int(model.predict_classes(x_TrainData)) # 模型預測類別

    plt.text(0, 2, "Predict: {}\nProbability: {}%".format(predict, prob[0][np.argmax(prob)] * 100), color = 'red') # 在圖上顯示文字
    plt.imshow(pic) # 畫圖
    plt.show() # 顯示圖片
```

三、結果

- 模型架構

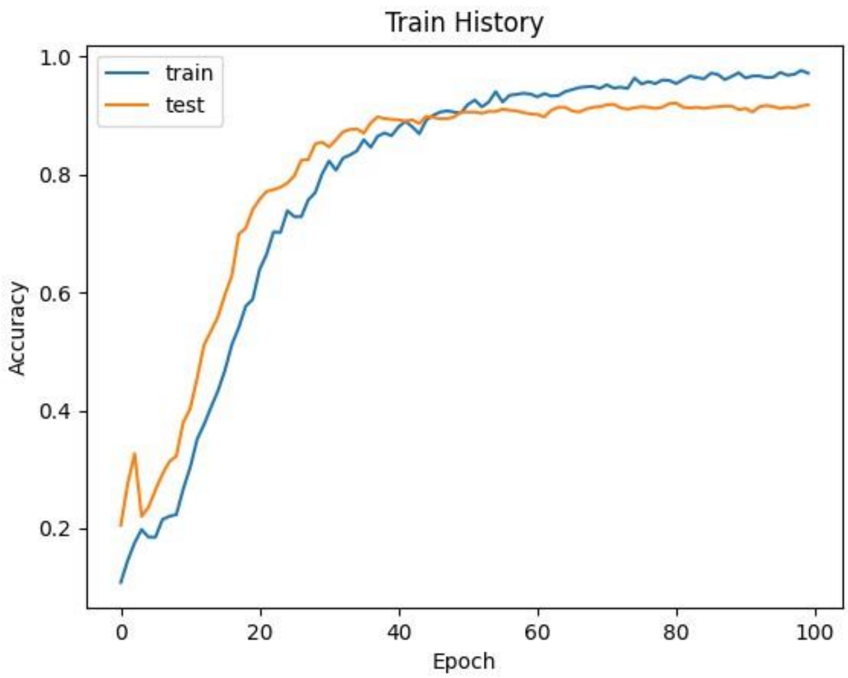
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 16)        416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 36)        14436
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 36)          0
_________________________________________________________________
dropout (Dropout)            (None, 7, 7, 36)          0
_________________________________________________________________
flatten (Flatten)            (None, 1764)              0
_________________________________________________________________
dense (Dense)                (None, 128)               225920
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 242,062
Trainable params: 242,062
Non-trainable params: 0
```

- 訓練精度

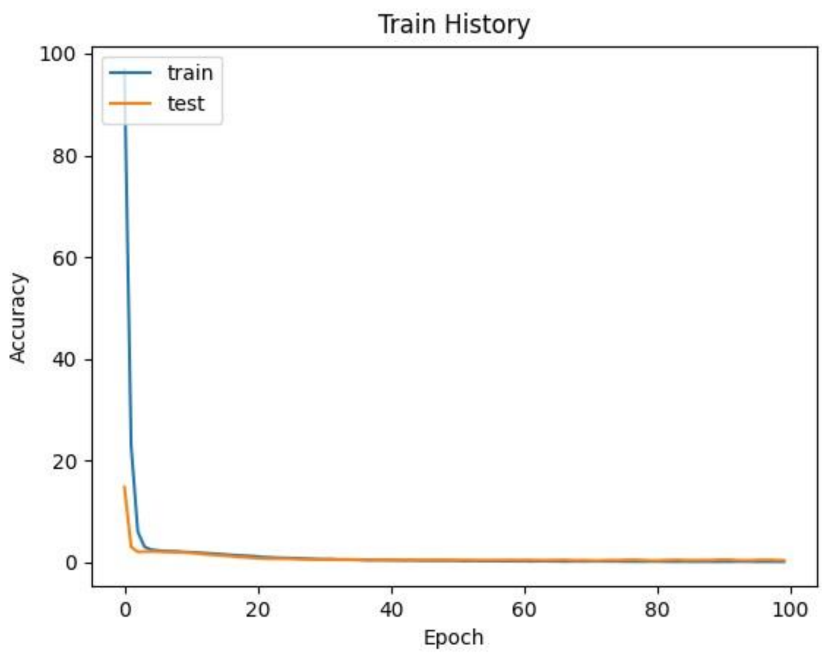
```
Epoch 92/100
3/3 - 0s - loss: 0.1048 - accuracy: 0.9673 - val_loss: 0.4801 - val_accuracy: 0.9061
Epoch 93/100
3/3 - 0s - loss: 0.1361 - accuracy: 0.9673 - val_loss: 0.3627 - val_accuracy: 0.9153
Epoch 94/100
3/3 - 0s - loss: 0.1335 - accuracy: 0.9646 - val_loss: 0.3576 - val_accuracy: 0.9173
Epoch 95/100
3/3 - 0s - loss: 0.1223 - accuracy: 0.9653 - val_loss: 0.3492 - val_accuracy: 0.9153
Epoch 96/100
3/3 - 0s - loss: 0.0934 - accuracy: 0.9735 - val_loss: 0.4241 - val_accuracy: 0.9122
Epoch 97/100
3/3 - 0s - loss: 0.1083 - accuracy: 0.9687 - val_loss: 0.4380 - val_accuracy: 0.9143
Epoch 98/100
3/3 - 0s - loss: 0.1049 - accuracy: 0.9701 - val_loss: 0.4459 - val_accuracy: 0.9133
Epoch 99/100
3/3 - 0s - loss: 0.0959 - accuracy: 0.9769 - val_loss: 0.3796 - val_accuracy: 0.9163
Epoch 100/100
3/3 - 0s - loss: 0.0950 - accuracy: 0.9721 - val_loss: 0.3689 - val_accuracy: 0.9184
```

訓練精準度為 97.21%，測試精準度為 91.84%

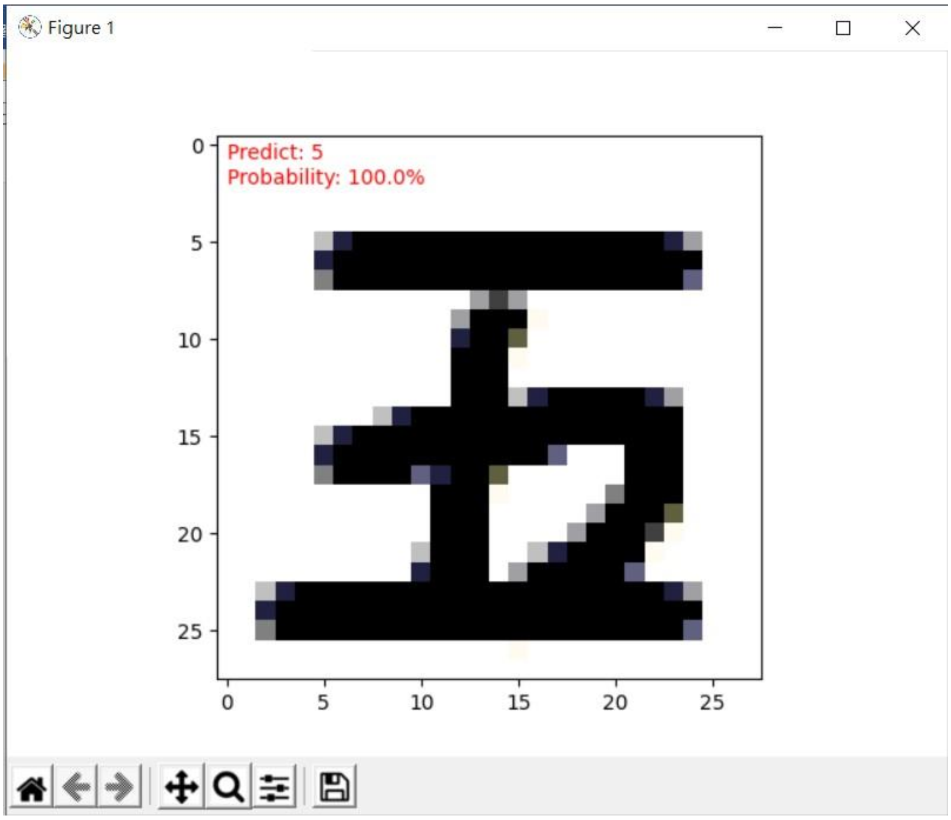
- 精度曲線


Train History
train
test
1.0
0.8
0.6
0.4
0.2
Accuracy
0
20
40
60
80
100
Epoch

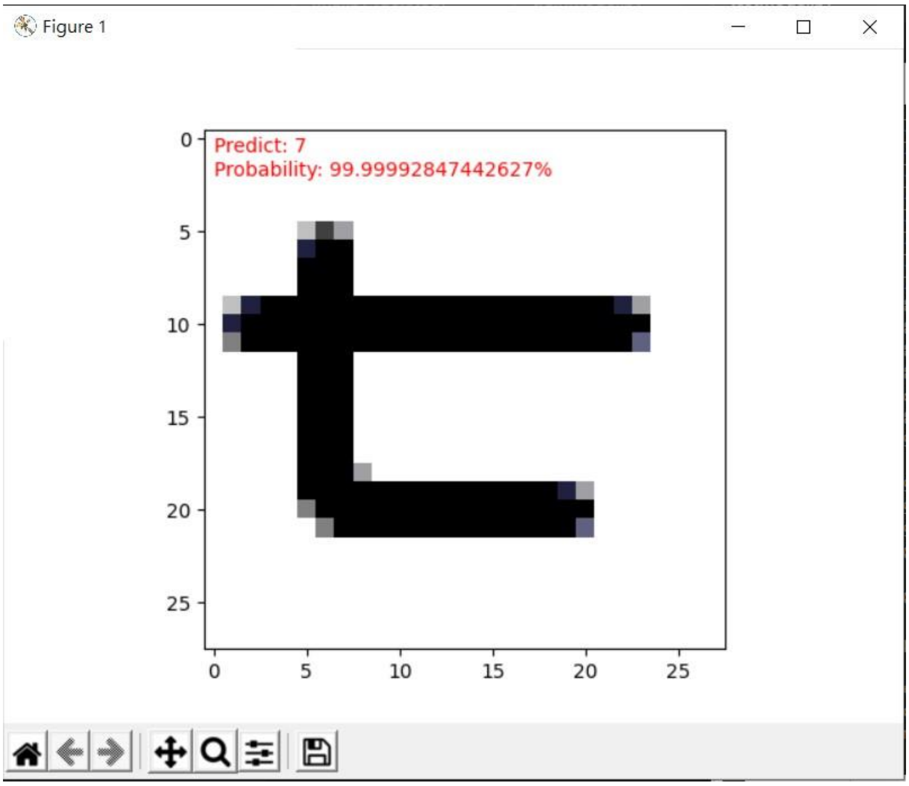
- 錯誤曲線


Train History
train
test
100
80
60
40
20
0
Accuracy
0
20
40
60
80
100
Epoch

- 模型測試(使用不同檔案做預判)
    - ./test_image/104201529/5/5(1).bmp



    - ./test_image/103503522/7/image3.bmp



- ./test_image/104503530/3/3_1.bmp