演算法 Term Project

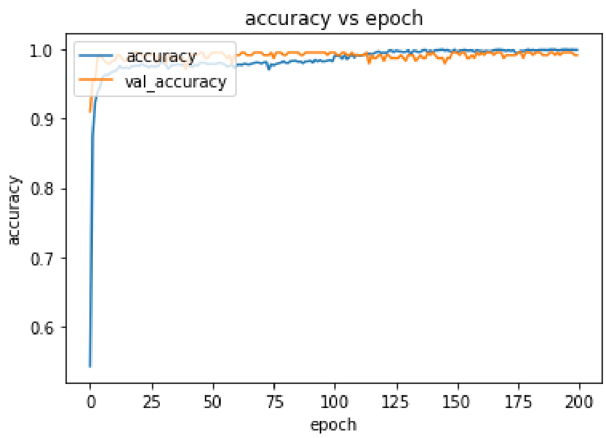
106503008 通訊四 蔡嘉倫

1.

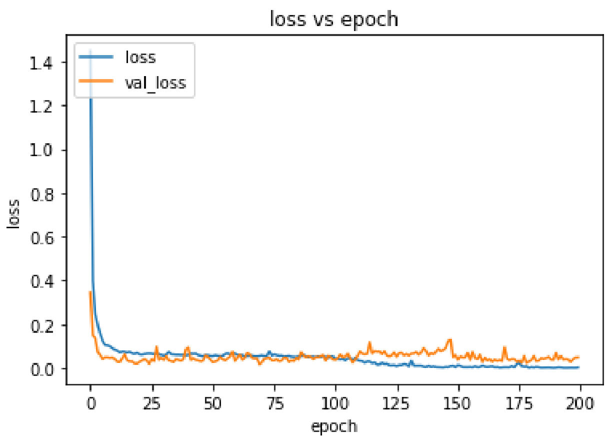
(1)損失函數值與準確率值

```
test loss 0.18521706759929657
test accuracy 0.9823529124259949
```

(2)準確度與執行次數比較圖



(3)損失函數值與執行次數比較圖



(4)Model

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_23 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 11, 11, 32)        9248
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 5, 5, 32)          0
_________________________________________________________________
dropout_30 (Dropout)         (None, 5, 5, 32)          0
_________________________________________________________________
flatten_11 (Flatten)         (None, 800)               0
_________________________________________________________________
dropout_31 (Dropout)         (None, 800)               0
_________________________________________________________________
dense_22 (Dense)             (None, 128)               102528
_________________________________________________________________
dropout_32 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_23 (Dense)             (None, 10)                1290
=================================================================
Total params: 113,386
Trainable params: 113,386
Non-trainable params: 0
```

(5)training process

```
Epoch 67/200
69/69 [==============================] - 1s 11ms/step - loss: 0.0569 - accuracy: 0.9787 - val_loss: 0.0341 - val_accuracy: 0.9959
Epoch 68/200
69/69 [==============================] - 1s 12ms/step - loss: 0.0552 - accuracy: 0.9810 - val_loss: 0.0470 - val_accuracy: 0.9918
Epoch 69/200
69/69 [==============================] - 1s 11ms/step - loss: 0.0608 - accuracy: 0.9791 - val_loss: 0.0381 - val_accuracy: 0.9959
Epoch 70/200
69/69 [==============================] - 1s 12ms/step - loss: 0.0571 - accuracy: 0.9782 - val_loss: 0.0210 - val_accuracy: 0.9918
Epoch 71/200
69/69 [==============================] - 1s 12ms/step - loss: 0.0551 - accuracy: 0.9805 - val_loss: 0.0299 - val_accuracy: 0.9959
Epoch 72/200
69/69 [==============================] - 1s 12ms/step - loss: 0.0543 - accuracy: 0.9819 - val_loss: 0.0302 - val_accuracy: 0.9959
Epoch 73/200
69/69 [==============================] - 1s 12ms/step - loss: 0.0530 - accuracy: 0.9814 - val_loss: 0.0423 - val_accuracy: 0.9959
Epoch 74/200
69/69 [==============================] - 1s 12ms/step - loss: 0.0643 - accuracy: 0.9760 - val_loss: 0.0346 - val_accuracy: 0.9959
Epoch 75/200
69/69 [==============================] - 1s 11ms/step - loss: 0.0637 - accuracy: 0.9800 - val_loss: 0.0497 - val_accuracy: 0.9878
```

2. Source code 之逐行解釋

(1)引入函式庫

```python
import os                # os模組，用於達成建立文件，刪除文件，查詢文件，而在此處作為讀手寫數字相片以及獲得路徑
from PIL import Image  # PIL是Python強大的圖像處理庫，能提供處理image的方法
import numpy as np       # numpy可用來存儲和處理大型矩陣，由於影像是二維陣列，因此可以處理圖片的二維運算
import keras.utils       # 來做one-hot encoding，one-hot encoding :將類別拆成多個行(column)，每個列中
                         # 的數值由1、0替代，當某一列的資料存在的該行的類別則顯示1，反則顯示0。

# 用keras模組引入Sequential()，
# Sequential Model (順序式模型)：就是一種簡單的模型，單一輸入、單一輸出，按順序一層(Dense)一層的由上往下執行
from keras.models import Sequential
# 卷積層 、池化層 、平坦層 、隱藏層、輸出層：
# Dense就是我們所用的全連接層，將上一層的神經元全數連結，實現特徵的非線性組合
from keras.layers import Dense
# Dropout是神經網絡和深度學習模型的一種簡單而有效的正則化方式，為避免overfit
from keras.layers import Dropout
# 平坦層使用多層感知器來穩定判斷結果。所以再接入多層感知器前，先必須將矩陣打平成一維的陣列作為輸入
from keras.layers import Flatten
# 進行卷積層的處理，提取特徵值，而這裡因為是二維振烈，故引用2D
from keras.layers import Conv2D
# 用於在池化層，找出最大值
from keras.layers import MaxPooling2D
import matplotlib.pyplot as plt  # 為了畫出loss vs epoch以及accuracy vs epoch的比較
```

(2)資料預處理

```python
# 把data_x和data_y(Label)做資料的預處理
def preprocessing(datapath):
    row, col = 28, 28                             # 定義圖片大小為28x28
    data_path = datapath                                  # 當前目錄的路徑
    data_x = np.zeros((28, 28)).reshape(1, 28, 28)  # #np_zero創建第一組28X28的零陣列,reshape
    counter = 0                                           # 用於記錄圖片張數的計數器
    data_y = []                                           # 紀錄label
    classnum = 10                                         # 數字有10種類別(0~9)，之後給utils進行one hot encoding
    #遞迴找出所有子目錄與檔案，即讀取handwrite資料夾內所有圖片
    for root, dirs, images in os.walk(data_path):
        for f in images:
            label = int(root.split("\\")[2])  # 方法即是用split去儲存handwriye資料夾的名稱作為label
            data_y.append(label)
            fullpath = os.path.join(root, f)  # 透過os.path.join取得圖片路徑
            img = Image.open(fullpath)        # 開啟圖片
            #灰階圖（channel=1）、尺寸(28x28)、正規化（值從0~255變成0~1）
            img = (np.array(img) / 255).reshape(1, 28, 28)
            data_x = np.vstack((data_x, img)) # 陣列堆疊至下一組
            counter += 1                      # 此時處理好一張相片，counter+1
    data_x = np.delete(data_x, [0], 0)        # 刪除一開始宣告的np.zero的全零陣列
    # 調整資料格式(圖片張數,row,col,色彩通道=1(灰階處理))
    data_x = data_x.reshape(counter, row, col, 1)
    # 將類別向量轉換為二進制（只有0和1）的矩陣類型表示，將label轉為one-hot encoding
    data_y = keras.utils.to_categorical(data_y, classnum)
    # 此function用於train.test picture的預處理
    return data_x, data_y
```

(3)找出圖片路徑

```
# 從路徑中，把train和test的手寫照片讀出
path = os.getcwd()                                    # 獲得當前路徑，由於此檔案是放在與train.test資料夾相同位置
                                                      # 因此只須加上train image/test image即可找到這二者位置
trainpath = os.path.join(path, 'train_image')  # 當前路徑加上train_image
trainpath = trainpath.replace('\\', '/')        # 在windows下會印出\\，但在預處理資料會出問題，所以改為/
train_x, train_y= preprocessing(trainpath)  # 讀train資料
testpath = os.path.join(path, 'test_image')  # 當前路徑加上test_image
testpath = testpath.replace('\\', '/')        # 在windows下會印出\\，但在預處理資料會出問題，所以改為/
test_x, test_y = preprocessing(testpath)  # 讀test資料
```

(4)Build Model

```
# 建立簡單的線性執行模型
model = Sequential()
# 建立第一個卷積層
model.add(
    Conv2D(filters=32,                # filter：建立32個filter去進行convolution
           kernel_size=(3, 3),        # kernel_size:每一個kernel為3x3，依此大小進行convolution
           input_shape=(28, 28, 1),   # input_shape：輸入影像的大小為28x28，並且以灰階呈現
           activation='relu'))        # activation：激活函數，用relu，使之非線性
# 建立池化層，池化大小=2x2，並取最大值
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(
    Conv2D(filters=32,                 # filter：建立32個filter去進行convolution
           kernel_size=(3, 3),         # kernel_size:每一個kernel為3x3，依此大小進行convolution
           activation='relu'))         # activation：激活函數，用relu，使之非線性
# 第二個池化層，池化大小=2x2，取最大值
model.add(MaxPooling2D(pool_size=(2, 2)))
# 建立Dropout，用來隨機斷開輸入神經元，用於防止overfitting，斷開比例0.2(即保留0.8)
model.add(Dropout(0.2))
# 建立Flatten層，以reshape轉換為1的向量
model.add(Flatten())
# 建立Dropout，用來隨機斷開輸入神經元，用於防止過度擬和，斷開比例0.2
model.add(Dropout(0.2))
# 建立全連接層：128個output
model.add(Dense(128, activation='relu'))
# 建立Dropout，用來隨機斷開輸入神經元，用於防止過度擬和，斷開比例0.2
model.add(Dropout(0.2))
# activation function使用softmax，由於softmax介於0.1之間是和分類，將結果分類(units=10)，代表分為10類
model.add(Dense(units=10, activation='softmax'))
# 輸出整個模型的摘要資訊，包含簡單的結構表與參數統計
model.summary()
```

(5)Model Compile

```
# 將model進行編譯: 選擇損失函數、優化方法及成效衡量方式
# loss：設定損失函數，在深度學習使用cross_entropy，訓練效果通常較好
# categorical_crossentropy:當預測值與實際值愈相近，損失函數就愈小，反之差距很大，就會更影響損失函數的值
model.compile(loss='categorical_crossentropy',
              #每一次的學習率都會有個確定的範圍，會使參數的更新更加穩定，因此Adam是常用的optimizer
              optimizer="adam",
              #能用metrics來判斷樣本準確率
              metrics=['accuracy'])
```

(6)Model Training

```
# 把model進行training，訓練過程會存在 train_history變數中
train_history = model.fit(train_x, train_y,     # 輸入訓練資料
                          epochs=200,           # epochs為訓練次數
                          batch_size=32,        # 每一次訓練32筆資料
                          verbose=1,            # 顯示模式:
                                                # verbose=0:不會顯示任何內容（無聲模式）
                                                # verbose=1:顯示一個動畫進度條
                                                # verbose=2:只顯示Epoch的樣式，另如:150/200
                          validation_split=0.1)# 設定訓練與驗證資料比例，在訓練之前keras會自動將資料分成：
                                                # 90%為訓練資料，10%作為驗證資料。
```

(7)輸出 loss 與準確度

```
loss, accuracy = model.evaluate(test_x, test_y, verbose=0)#用於評估模型，輸出為準確度
print("test loss", loss)                                  #打印出loss的值
print("test accuracy", accuracy)                          #打印出accuracy的值
```

(8)畫出比較圖

```
# 畫accuracy隨著epoch次數的變化圖
plt.figure(1)                                  # 圖一
#history中儲存了'acc', 'loss', 'val_acc', 'val_loss'4種數據
plt.plot(train_history.history['accuracy'])    # 取得模型在訓練集上準確率
plt.plot(train_history.history['val_accuracy'])  # 取得模型在驗證集上準確率
plt.title('accuracy vs epoch')                 # 設圖一的title為accuracy vs epoch
plt.ylabel('accuracy')                         # 設y座標的label為accuracy
plt.xlabel('epoch')                            # 設x座標的label為epoch
plt.legend(['accuracy', 'val_accuracy'], loc='upper left')  # 設圖例
plt.show()                                     # 顯示圖一

# 畫loss隨epoch次數的變化圖
plt.figure(2)                                  # 圖二
plt.plot(train_history.history['loss'])        # 取得模型在訓練集上的loss值
plt.plot(train_history.history['val_loss'])    # 取得模型在驗證集上的loss值
plt.title('loss vs epoch')                     # 設圖二的title
plt.ylabel('loss')                             # 設y座標的label
plt.xlabel('epoch')                            # 設x座標的label
plt.legend(['loss', 'val_loss'], loc='upper left')  # 設圖例
plt.show()                                     # 顯示圖二
```