Handwritten Character Recognition with Neural Network

黃子睿 ，吳宗諺
電機系
臺北大學
新北市，臺灣
s410987042@gm.ntpu.edu.t
w
s410987001@gm.ntpu.edu.t
w

摘要—這個 Project 中，我們建立一個 CNN(Convolutional Neural Networks)卷積神經網路模型，給予模型大量的英文字母資料庫，讓模型去學習如何偵測及判斷出手寫的英文字母 A~Z，這項功能可以應用在車牌辨識，電子手寫簽名等等。

關鍵字—CNN，字母辨識，機器學習

I. 簡介

CNN 卷積神經網路是由一個或多個卷積層和頂端的全連通層組成，同時也包括關聯權重和池化層。這一結構使得卷積神經網路能夠利用輸入資料的二維結構。因此與其他深度學習結構相比，卷積神經網路在圖像和語音辨識方面能夠給出相比其他種神經網路更好的結果。於是我們將英文字母資料庫丟入建立好的 CNN 後對 CNN 進行訓練，訓練完成後它就可以辨識本身資料庫中的英文字母以及人類手寫的英文字母，這樣一來就可以將這功能應用到車牌辨識或電子手寫簽名等等需要辨識字母的任務。

II. 步驟

Step1:匯入需要的函數庫(圖一)

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
from keras.optimizers import SGD, Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.utils import to_categorical
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import tensorflow as tf
```

(圖一)

Step2:從預設的資料庫中讀取檔案 (圖二)

```
data = pd.read_csv(r"D:\A_Z_Handwritten_Data.csv").astype('float32')
print(data.head(10))
```

(圖二)

讀取電腦中 Handwritten_Data 的 csv 檔案，並印出前 10 筆 data(圖三)

```
     0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  ...  0.639  0.640  0.641  \
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
5  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
6  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
7  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
8  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
9  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0

   0.642  0.643  0.644  0.645  0.646  0.647  0.648
0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3    0.0    0.0    0.0    0.0    0.0    0.0    0.0
4    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5    0.0    0.0    0.0    0.0    0.0    0.0    0.0
6    0.0    0.0    0.0    0.0    0.0    0.0    0.0
7    0.0    0.0    0.0    0.0    0.0    0.0    0.0
8    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9    0.0    0.0    0.0    0.0    0.0    0.0    0.0

[10 rows x 785 columns]
```

(圖三)

Step3:分類資料及 reshape(圖四)

```
train_x, test_x, train_y, test_y = train_test_split(X, y, random_state=1)
train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))
test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))
print("Train data shape: ", train_x.shape)
print("Test data shape: ", test_x.shape)
```

(圖四)

用隨機方式分割將 data 分成 train data 及 test data。因為我們後面要把資料用圖像顯示，所以我們這裡把資料庫 csv 檔案中 data 原本的 784columns 的 pixel data 轉換成 28*28 的 pixels。

Step4:創立字母對應整數的字典(圖五)

```
In [5]: word_dict = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',15:'P'
                     ,16:'Q',17:'R',18:'S',19:'T',20:'U',21:'V',22:'W',23:'X', 24:'Y',25:'Z'}
```

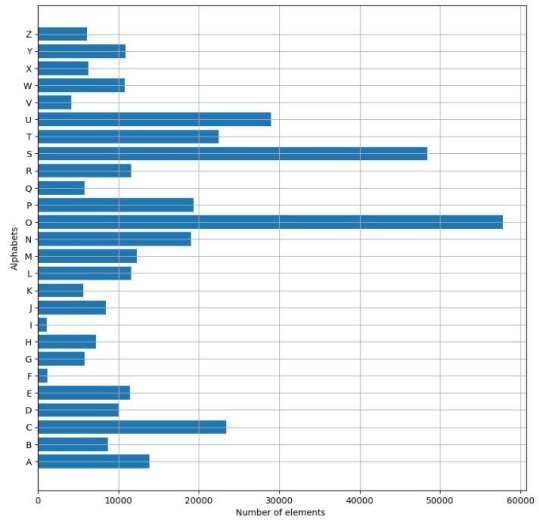
(圖五)

Step5:觀看資料庫中每個字母的數量(圖六)

```
y_int = np.int0(y)
count = np.zeros(26, dtype='int')
for i in y_int:
    count[i] +=1
alphabets = []
for i in word_dict.values():
    alphabets.append(i)
fig, ax = plt.subplots(1,1, figsize=(10,10))
ax.barh(alphabets, count)
plt.xlabel("Number of elements ")
plt.ylabel("Alphabets")
plt.grid()
plt.show()
```

(圖六)

Output:字母數量分布圖(圖七)



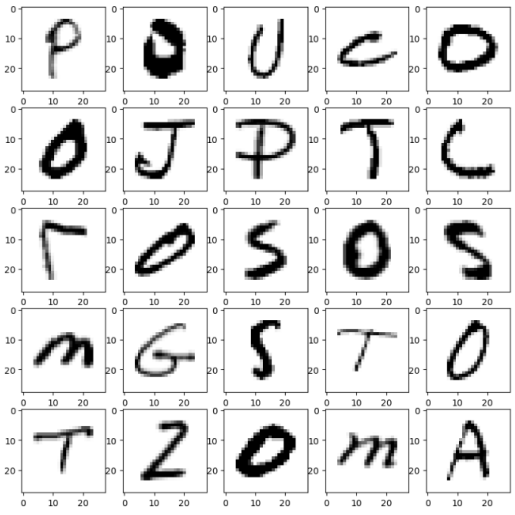
(圖七)

Step6:資料分堆 (圖八)

```
shuff = shuffle(train_x[:100])
fig, ax = plt.subplots(5,5, figsize = (10,10))
axes = ax.flatten()
for i in range(25):
    _, shu = cv2.threshold(shuff[i], 30, 200, cv2.THRESH_BINARY)
    axes[i].imshow(np.reshape(shuff[i], (28,28)), cmap="Greys")
plt.show()
```

(圖八)

將 train data set 進行分堆，隨機挑選 25 個字母顯示 Output(圖九)



(圖九)

Step7:data reshape (圖十)

```
train_X = train_x.reshape(train_x.shape[0],train_x.shape[1],train_x.shape[2],1)
print("New shape of train data: ", train_X.shape)
test_X = test_x.reshape(test_x.shape[0], test_x.shape[1], test_x.shape[2],1)
print("New shape of train data: ", test_X.shape)
```

(圖十)

將 data 轉換成可以輸入 CNN 模型的大小
Output(圖十一)

```
New shape of train data:  (279337, 28, 28, 1)
New shape of train data:  (93113, 28, 28, 1)
```

(圖十一)

Step8:建立 CNN 模型(圖十二)

```
In [10]: model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))
model.add(Dense(32,activation ="relu"))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))
model.add(Dense(64,activation ="relu"))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))
model.add(Dense(128,activation ="relu"))

model.add(Flatten())
model.add(Dense(26,activation ="softmax"))
```

(圖十二)

Step9:訓練模型(圖十三)

```
model.compile(optimizer = Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_X, train_yOHE, epochs=1,  validation_data = (test_X,test_yOHE))
model = tf.keras.models.load_model('model_test1.h5')
```

(圖十三)

這裡對模型進行 compile 以及 fitting。第三行的 load model 是為了讀取訓練完的模型(因為訓練比較費時，將訓練好的模型直接讀取可以省下大量時間)

Step10:存取訓練完的模型(圖十四)

```
model.summary()
model.save(r'model_hand_test1.h5')
```

(圖十四)

訓練完模型後，將模型儲存成.h5 檔案，供日後讀取模型使用

訓練完後的模型(圖十五)

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_6 (MaxPooling  (None, 25, 25, 32)        0
2D)
_________________________________________________________________
dense_8 (Dense)              (None, 25, 25, 32)        1056
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 25, 25, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling  (None, 24, 24, 64)        0
2D)
_________________________________________________________________
dense_9 (Dense)              (None, 24, 24, 64)        4160
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 22, 22, 128)       73856
_________________________________________________________________
max_pooling2d_8 (MaxPooling  (None, 21, 21, 128)       0
2D)
_________________________________________________________________
dense_10 (Dense)             (None, 21, 21, 128)       16512
_________________________________________________________________
flatten_2 (Flatten)          (None, 56448)             0
_________________________________________________________________
dense_11 (Dense)             (None, 26)                1467674
=================================================================
Total params: 1,582,074
Trainable params: 1,582,074
Non-trainable params: 0
```

(圖十五)

Step11:查看準確度(圖十六)

```
print("The validation accuracy is :", history.history['val_accuracy'
print("The training accuracy is :", history.history['accuracy'])
print("The validation loss is :", history.history['val_loss'])
print("The training loss is :", history.history['loss'])
```

（圖十六）

Output:（圖十七）

```
The validation accuracy is : [0.9790254831314087]
The training accuracy is : [0.9667104482650757]
The validation loss is : [0.0755283385515213]
The training loss is : [0.14348392188549042]
```

（圖十七）

Step12:測試模型運作是否正常（圖十八）

```
fig, axes = plt.subplots(10,10, figsize=(15,20))
axes = axes.flatten()
for i,ax in enumerate(axes):
    img = np.reshape(test_X[i], (28,28))
    ax.imshow(img, cmap="Greys")
    pred = word_dict[np.argmax(test_yOHE[i])]
    ax.set_title("Prediction: "+pred)
    ax.grid()
```

（圖十八）

測試 100 個字母是否預測正確

Output:（圖十九）

（圖十九）

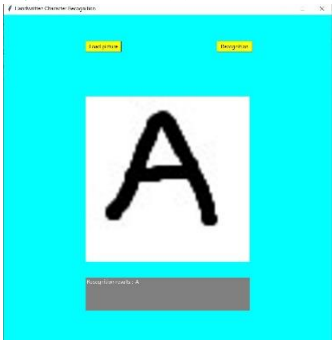
可以看到結果均正確

Step13:建立 UI（圖二十）

```
import tkinter as tk
import tkinter as tk
from PIL import ImageTk, Image
from tkinter import messagebox
from tkinter import filedialog
from PIL import Image,ImageTk
import copy


def show_import_img():
    global path
    path= filedialog.askopenfilename()
    if not path:
        messagebox.showinfo("Info", "Import failed")
    else:
        lb_results.configure(text ='Recognition results :  ',fg='white')
        import_img(path)


def import_img(path):
        img = Image.open(path)
        tk_img = ImageTk.PhotoImage(img)
        img = Image.open(path)
        w, h = img.size
        image_resized=img.resize((500,500))
        tk_img = ImageTk.PhotoImage(image_resized)
        canvas.delete('all') #
        canvas.config(scrollregion=(0,0,w,h))
        canvas.create_image(0, 0, anchor='nw', image=tk_img)
        canvas.tk_img = tk_img

def prediction():
    img = cv2.imread(path)
    img_copy = img.copy()
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (400,440))

    img_copy = cv2.GaussianBlur(img_copy, (7,7), 0)
    img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
    _, img_thresh = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV)
    img_final = cv2.resize(img_thresh, (28,28))
    img_final =np.reshape(img_final, (1,28,28,1))
    img_pred = word_dict[np.argmax(model.predict(img_final))]
    lb_results.configure(text ='Recognition results :  '+img_pred,fg='white')


#window
window = tk.Tk()
window.title('Handwritten Character Recognition')
window.geometry('1000x1000')
window.resizable(False,False)
window.configure(background='aqua')

#canvas
canvas = tk.Canvas(window, width=500, height=500, bg='gray')
canvas.create_text(250, 250, text='請匯入圖片',fill='black')
canvas.pack(padx='250')
canvas.pack(side='left')

#button
bt_import_image=tk.Button(text='Load picture',width=11,command=show_import_img,activebackground='blue'
                          ,activeforeground='green',bg='yellow')
bt_recognition=tk.Button(text='Recognition', width=11,command=prediction,activebackground='blue'
                         ,activeforeground='green',bg='yellow')
#label
lb_results=tk.Label(anchor='nw',text='Recognition results :  ',width=55,bg='gray',fg='white',height=5)
#layout
bt_import_image.place(x=250, y=80)
bt_recognition.place(x=650, y=80)
lb_results.place(anchor='nw',x=250,y=800)

#mainloop
window.mainloop()
```

（圖二十）

Step14:實際輸入手寫資料進行預測（圖二十一）

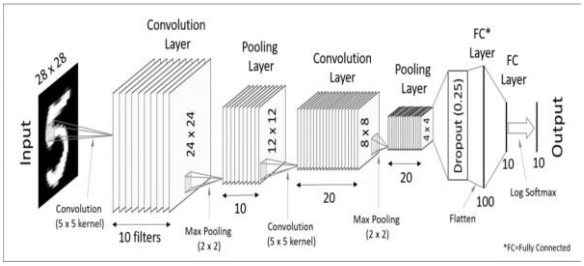


（圖二十一）

III. CNN 架構及參數

CNN 卷積神經網路是由一個或多個卷積層和頂端的全連通層組成，同時也包括關聯權重和池化層。這一結構使得卷積神經網路能夠利用輸入資料的二維結構。因此與其他深度學習結構相比，卷積神經網路在圖像和語音辨識方面能夠給出更好的結果。

A. CNN 架構:(圖二十二)



(圖二十二)

可以看到 CNN 的 input 與 Output 之間隔了許多交替的卷基層及池化層，接著經過扁平層把多維的輸入壓扁一維輸出，最後經過完全連接層輸出。

B. CNN 架構中的層(layer)功用是甚麼

1. 卷積層(convolution layer)

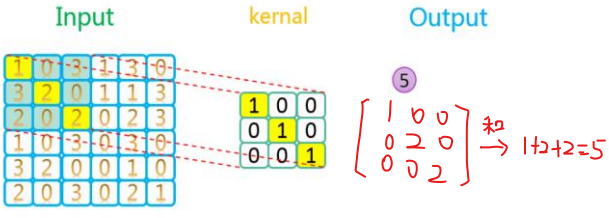
可以產生一組平行的特徵圖（feature map），它通過在輸入圖像上滑動不同的卷積核(kernal)並執行哈達馬達乘積[*]而組成。此外，在每一個滑動的位置上，卷積核與輸入圖像之間會執行一個元素對應乘積並求和的運算以將感受的資訊投影到特徵圖中的一個元素。一張特徵圖中的所有元素都是通過一個卷積核計算得出的，也即一張特徵圖共享了相同的權重和偏置項。用文字敘述有點抽象，下面會展示卷積過程的示意圖，會更容易理解。

[*]哈達馬達乘積:當 A 與 B 矩陣進行哈達馬達乘積，相對位置相同的元素進行相乘，結果如下，然後求這個乘積矩陣所有元素的和，作為卷積的結果。(圖二十三)

$$A * B = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1J}b_{1J} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2J}b_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}b_{I1} & a_{I2}b_{I2} & \cdots & a_{IJ}b_{IJ} \end{bmatrix}.$$

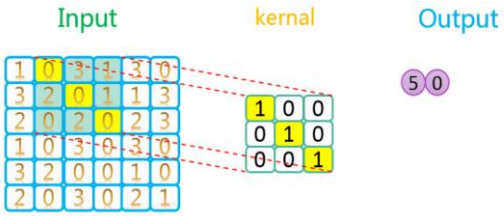
(圖二十三)

卷積過程示意圖(kernel size=3*3, stride=1，按此連結可跳至參數解釋)(圖二十四~圖二十九)

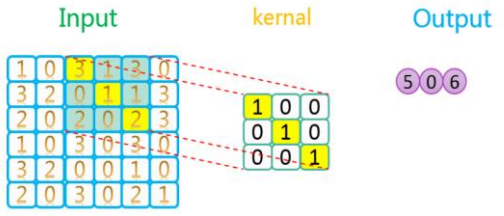


(圖二十四)

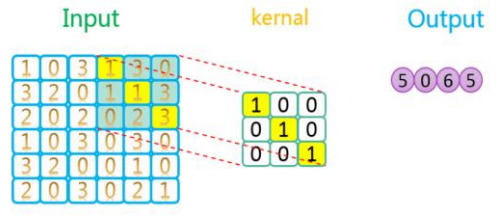
Output=1*1+2*1+2*1=5(兩矩陣進行哈達馬達乘積，然後將結果全部相加即為 Output)



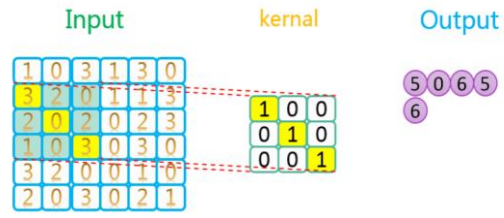
(圖二十五)



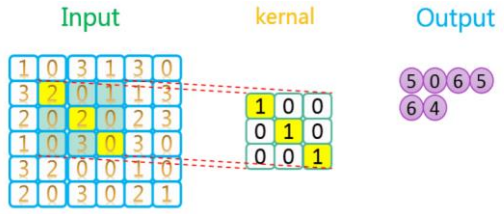
(圖二十六)



(圖二十七)



(圖二十八)

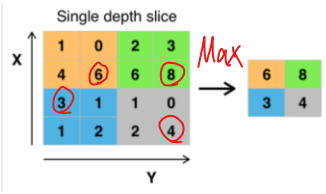


(圖二十九)

…後面步驟以此類推，最後得到 4*4 的 Output

2. 池化層(pooling layer):

池化（Pooling）是卷積神經網路中一個重要的概念，他是一種非線性形式的降採樣。有多種不同形式的非線性池化函式，而其中「最大池化（Max pooling）」是最為常見的(其他還有取平均值，取全局最大值)。它是將輸入的圖像劃分為若干個矩形區域，對每個子區域輸出最大值。池化層每次在一個池化窗口上計算輸出，然後根據步幅移動池化窗口。(圖三十)是目前最常用的池化層。



（圖三十）

步幅為 2，池化窗口為 2*2 的最大池化層，這將會減少 75%的資料量

這種機制能夠有效地原因在於，一個特徵的精確位置遠不及它相對於其他特徵的粗略位置重要。池化層會不斷地減小資料的空間大小，因此參數的數量和計算量也會下降。但是要注意池化也會因此丟失大量的信息，kernel size 可以設定各種尺寸，取較大的 kernel，可抓取較大的特徵，取較小的 kernel，可抓取較小的特徵。

另外，我們通過卷積層可以很容易地發現圖像中的各種邊緣。但是卷積層發現的特徵往往過於精確，通過池化層我們可以降低卷積層對邊緣的敏感性。

3. Dropout 層：主要目的為防止過度擬合，首要參數 rate 為關掉隱藏層節點的比例，利用隨機關掉隱藏層節點與輸入神經元的連結，不更新權重(W)，造成多個結果，再作比較去除極端值，即可達到避免過度擬合的現象。

4. 扁平層(flatten)：把多維的輸入壓扁為一維輸出，常用在從卷積層到全連接層的過渡，無參數

5. 完全連結層(Fully connected (Dense) layer)

最後，在經過幾個卷積和最大池化層之後，神經網路中的進階推理通過完全連接層來完成。就和常規的非卷積人工神經網路中一樣，完全連接層中的神經元與前一層中的所有啟用都有聯絡。因此，它們的啟用可以作為仿射變換來計算，也就是先乘以一個矩陣然後加上一個偏差(bias)偏移量(向量加上一個固定的或者學習來的偏差量)。

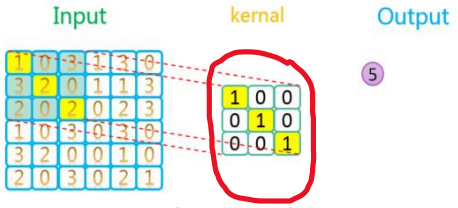
C. Code 參數解釋：（圖三十一）

```
In [10]: model = Sequential()
         model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)))
         model.add(MaxPool2D(pool_size=(2, 2), strides=2))
         model.add(Dense(32,activation ="relu"))

         model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'))
         model.add(MaxPool2D(pool_size=(2, 2), strides=2))
         model.add(Dense(64,activation ="relu"))

         model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'))
         model.add(MaxPool2D(pool_size=(2, 2), strides=2))
         model.add(Dense(128,activation ="relu"))

         model.add(Flatten())
         model.add(Dense(26,activation ="softmax"))
```

（圖三十一）

1. 卷積核(Kernal)

如上面卷積提到的，卷積核是進行卷積時使用的，在 Tensorflow 中，被稱為 filter。（圖三十二）

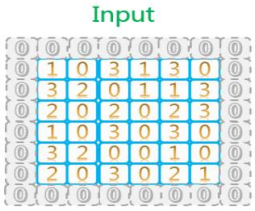


（圖三十二）

2. 填充(Padding)方式（圖三十三）

Valid 表示不做填充，Same 表示進行填充

填充的作用主要是盡可能充分的保留和使用輸入特徵（如果不使用填充，網絡層數越深，所丟失的邊緣數據就越多，到最後可能無特徵可用），同時也應注意填充的尺寸不宜過大，避免引入過多無用的數據



（圖三十三）

Same 類型以 0 進行填充示意圖

3. 滑動步長(Strides)

滑動步長決定了在卷積或者池化的過程中，每次操作移動的步數。

4. pool size

Pool size 大小為池化核的大小，池化核是在進行池化操作時候的 Kernal，池化的作用類似於 PCA，可以有效的對數據降維同時保留關鍵特徵

5. 激活函數(Activation)

(1)Relu(Rectified Linear Units):

使用線性整流 $f(x) = \max(0, x)$ 作為這一層神經的激勵函式，當 x<0 時為 0，當 x>0 時為 x。它可以增強判定函式和整個神經網路的非線性特性，而本身並不會改變卷積層。

(2)Softmax:

可以將 x 轉為機率值，且所有類別的機率總和等於 1，適合多分類，最大值就代表可能性最大

(3)Sigmoid:

這也蠻常用的，但我這裡沒用到順便補充一下。Sigmoid 函數可以將 X 的範圍限制在[0,1]之間，中間只有一小段模糊地帶，適合用於二分法。

6. filter:

該 layer 的層數，層數過小會不夠精確，過大可能會 overfit。

7. Dense:

各層的變數設定，第一個數字為該層的輸出變數

D. 可能遇到的問題

1. 沒有對數據標準化

2. 學習率（learning rate)選用不對

3. 樣本過大

這三者可能會導致模型不收斂，而不收斂會導致模型的準確度大幅下降。

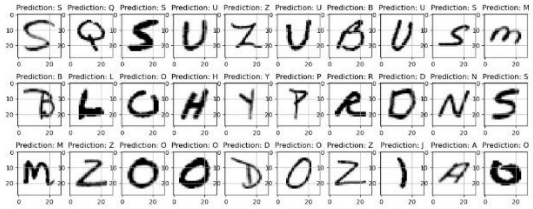
IV. 結果

1. 模型準確度:（圖三十四）

```
The validation accuracy is : [0.9790254831314087]
The training accuracy is : [0.9667104482650757]
The validation loss is : [0.0755283385515213]
The training loss is : [0.14348392188549042]
```

（圖三十四）

2. 資料庫預測:(圖三十五）



（圖三十五）

3. 實際判讀手寫字母:（圖三十六）

（圖三十六）

V. 結論與心得

結論:

在這項 project 中，我們首先將 data 分類以及 reshape，把 data 整理好後，接著建立了 CNN 的模型（其中的架構是將好幾層的卷積層與池化層交錯，接著經過扁平層把資料從多維轉換為一維，最後透過全連接層輸出)。接著將 data 丟入模型訓練，訓練完成後的驗證準確度及訓練準確度都到達了大約 97%的準確度。訓練完後就可以讓模型去測試 test data 是否可以判斷正確，最後我們寫了一個 UI 來測試判斷我們自己手寫的字母。這個功能若資料庫再加上數字的資料庫，也可以做到車牌辨識。若再加上人臉，指紋，虹膜等資料庫，也可以做到人臉辨識，指紋辨識，虹膜辨識等等，但是這幾個比起判斷字母還要更精細許多，所以要想辦法有效率地抓取有用的特徵值，還有提高精準度，並不是那麼容易。

心得:

我在做過及看過許多機器學習的應用及案例後發現大多機器學習的步驟都大同小異。通常步驟如下:

第一步:收集大量資料

這裡我們使用網路上的 data set

第二步:準備數據

這裡我們把 data 進行分堆以及 reshape

第三步:選擇模型

這裡我們選擇神經網路中的 CNN 模型

第四步:訓練機器

第五步:評估分析

第六步:調整參數

這裡我們根據測試後的準確度不斷去調整 CNN 模型的參數及調整架構(例如:卷基層數，池化層數，kernal 大小，stride 等等)

這個 project 中大部分概念老師上課都教過了，比較陌生的部份我覺得是 CNN 及 UI。雖然我之前對 CNN 並不太熟，但是 CNN 在神經網路中算是蠻好理解的一種了，因此查詢並熟讀一些網路上的資料便大致理解了 CNN 整個的運作原理。

而 UI 是因為之前沒寫過，幾乎沒概念要怎麼寫出一個 UI。但是查了後發現在 python 中寫 UI 並沒有想像中那麼難(當然要設計更完整，更好看的 UI 就很困難了)，許多指令都很直觀，一個指令對應一個物件有一點像在寫 html 的感覺。

CNN 架構中原先我們 strides 是設定 1(圖三十七)，準確度如圖三十八，但是後來發現設定 2 會比較好。原因是因為 stride 設定 2 的話(圖三十九)，精確度只會下降一點點(例如 validation accuracy 下降約 0.002 training accuracy 下降約 0.1)，但是訓練的時間可以縮短接近兩倍。於是後來設定 strides=2。

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=(2, 2), strides=1))
model.add(Dense(32,activation ="relu"))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=1))
model.add(Dense(64,activation ="relu"))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=1))
model.add(Dense(128,activation ="relu"))

model.add(Flatten())
model.add(Dense(26,activation ="softmax"))
```

（圖三十七）

Strides=1 準確度

```
The validation accuracy is : [0.9790254831314087]
The training accuracy is : [0.9667104482650757]
The validation loss is : [0.0755283385515213]
The training loss is : [0.14348392188549042]
```

（圖三十八）

Stride=2 準確度

```
The validation accuracy is : [0.977307140827179]
The training accuracy is : [0.9552475810050964]
The validation loss is : [0.08046963065862656]
The training loss is : [0.16546852886676788]
```

（圖三十九）

另外我之前一直搞不清楚 validation 及 test 的差別在哪。我查到一篇文章我覺得解釋得非常貼切。文章中作者把 train set 比喻成解題大全，validation set 比喻成模擬考試，test set 比喻成最後的大考(學測，高考之類的)。從這比喻我們就可知道 validaiton set(模擬考)是讓模型知道哪裡可以再更好去做修正，達到更高的精確度，test set 是模型最後才會跑的正式測驗。若我們讓模型先跑 test set，就相當於給模型最終的標準答案，那模型變成只是在背最終結果的答案，事實上他並沒有學會。因此 train validation test 三個數據集最好是互相無交集的。

我從國高中就不斷的聽到 AI，機器學習，互聯網等等，聽起來很厲害很艱深的詞彙。上了一個學期的機器學習課程，以及透過期中以及期末的報告之後，我對於機器學習有了更深更全面的認識，如果我以後走 AI 或研究深度學習等等領域的話，這學期所學到的知識必定會派上用場，實在是受益良多。

VI. 參考資料

1.https://ithelp.ithome.com.tw/articles/10192028
CNN 模型設計

2. https://flat2010.github.io/2018/06/15/%E6%89%8B%E7%AE%97CNN%E4%B8%AD%E7%9A%84%E5%8F%82%E6%95%B0/ CNN 中的參數解釋及計算

3.https://blog.sciencenet.cn/blog-377709-1187879.html
train & valid & test 解釋

4. https://data-flair.training/blogs/handwritten-character-recognition-neural-network/
手寫字母判斷範例

5. https://zh.wikipedia.org/zh-tw/%E5%8D%B7%E7%A7%AF%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C CNN 維基

6.https://medium.com/jameslearningnote/%E8%B3%87%E6%96%99%E5%88%86%E6%9E%90-%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E7%AC%AC5-1%E8%AC%9B-%E5%8D%B7%E7%A9%8D%E7%A5%9E%E7%B6%93%E7%B6%B2%E7%B5%A1%E4%BB%8B%E7%B4%B9-convolutional-neural-network-4f8249d65d4f 卷積神經網絡介紹

7. https://blog.csdn.net/simongeek/article/details/78200127 train & valid & test 解釋