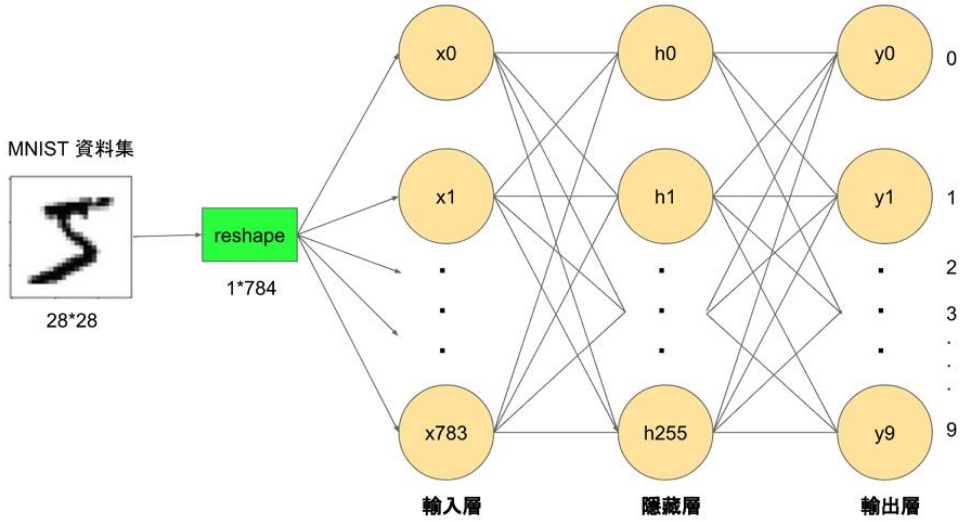
作品名稱：mnist 手寫數字辨識

一、說明

深度學習就是透過各種類神經網路，本專題會使用多層感知器(MLP)、卷積神經網路(CNN)、循環神經網路(RNN)透過 mnist 資料集產出訓練出來的值，將一大堆的數據輸入神經網路中，讓電腦透過大量數據的訓練找出規律自動學習，最後讓電腦能依據自動學習累積的經驗作出預測。

二、論文

多層感知器(MLP)：多層感知器是由多層人工神經元組成的類神經網路, 在 MINST 資料集的手寫數字辨識中要用到的 MLP 為如下具有輸入層, 一個隱藏層, 以及輸出層的類神經網路。

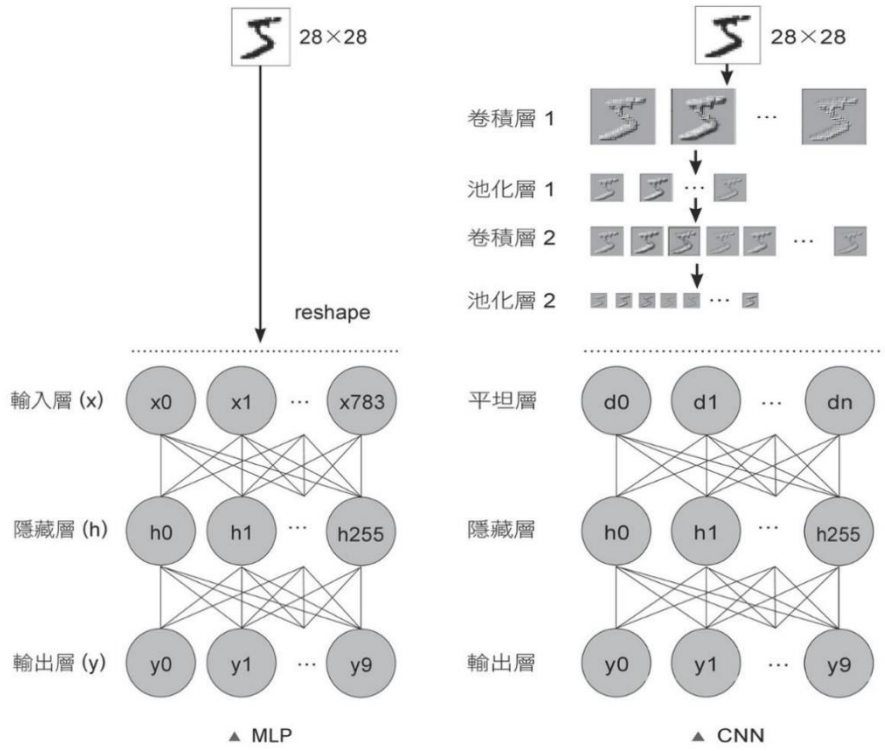


為了提高學習的準確率，神經網路更發展到有一個輸入層、一個或多個隱藏層及一個輸出層的多層感知器(MLP)。

- 輸入層：數字圖片是一張 28*28 的圖片、共有 784 個神經元所組成了神經網路第一層，數值的範圍介於 0~1 之間。灰階 0 代表灰色、1 代表白色，又稱為激勵值，數值越大則該神經元就越亮。
- 輸出層：完成輸入層後先不管其他層的內容，我們來看看他最右方的輸入層，也就是最後判斷的結果，其中有 10 個神經元，各代表數字 0~9，其中也有代表的激勵值。
- 隱藏層：為了方便說明，在這裡我們設計了兩個隱藏層，每層有 16 個神經元。在真實的案例中可依據需求設置調整隱藏層與神經元的數量。

卷積神經網路(CNN)：它是目前深度神經網路(Deep Neural Network)領域發展的主力，在圖片辨別上甚至可以做到比人類還精準之程度。

- 結構圖，和多層感知器相比較，卷積神經網路增加卷積層 1、池化層 1、卷積層 2、池化層 2，提取特徵後再以平坦層將特徵輸入神經網路中。以下使用 MNIST 資料及進行說明：
-

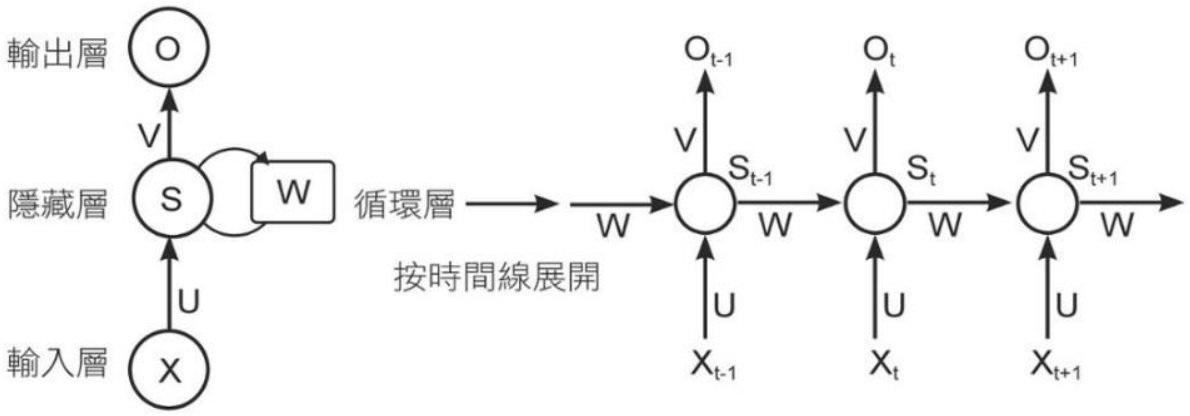


圖片中最上方有卷積層 1、池化層 1、卷積層 2、池化層 2，將原始的圖片以卷積、池化處理後產生更多的特徵小圖片，作為輸入的神經元。

- 卷積層：是將原始圖片與特定的濾鏡(Feature Detector)進行卷積運算，你也可以將卷積運算看成是原始圖片濾鏡特效的處理，filters 可以設定濾鏡數目，kernel_size 可以設定濾鏡(filter)大小，每一個濾鏡都會以亂數處理的方式產生不同的卷積運算，因此可以得到不同的濾鏡特效效果，增加圖片數量。
- 池化層：是採用 Max Pooling，指挑出矩陣當中的最大值，相當於只挑出圖片局部最明顯的特徵，這樣就可以縮減卷積層產生的卷積運算圖片數量。

循環神經網路(RNN)：它是「自然語言處理」領域最常使用的神經網路模型，LSTM 因為 RNN 前面的輸入和後面的輸入具有關連性，即可以建立回饋迴路。也可以用於語言翻譯、情緒分析、氣象預測及股票交易等。

- 結構圖：循環神經網路中主要有三種模型，分別是 SimpleRNN、LSTM 和 GRU。因為 SimpleRNN 超簡單，效果不夠好，記不住長期的事情，所以又發展出長短記憶網路(LSTM)，然後 LSTM 又被簡化為閘式循環網路 GRU。

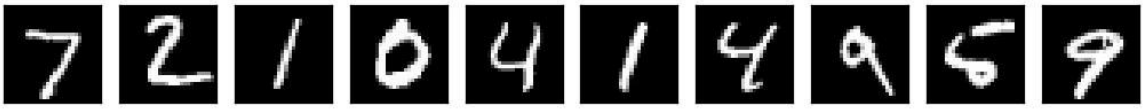


如圖共有三個時間點依序是 t-1、t、t+1，在 t 的時間點：

- X1 是神經網路 t 時間點的輸入，Ot 是神經網路 t 時間點的輸出。
- (U,V,W)都是神經網路共用的參數，W 參數是神經網路 t-1 時間點輸出，並且也作為神經網路 t 時間點的輸入。
- S1 是隱藏狀態，代表神經網路上的記憶，是神經網路目前時間點的輸入 X1 加上上個時間點的狀態 St-1，再加上 U 與 W 的參數，共同評估之結果：
  St = f(U * Xt + W * St-1)
  簡單來說就是前面的狀態會影響現在的狀態，現在的狀態也會影響以後的狀態。

三、實作

MNIST 資料集是由紐約大學 Yann Le Cun 教授蒐集整理很多 0~9 的手寫數字圖片所形成的資料集，這是一個大型手寫數字資料庫，對於機器學習學者來說是初學者，圖片每張大小為 28*28、皆為灰階影像，每個像素為 0~255 之數值、資料庫當中包含了 60000 筆的訓練資料、10000 筆的測試資料。在 MNIST 資料集中，每一筆資料都是由下載好的 mnist 的資料實作出成果。



1. 使用多層感知(MLP)進行辨識訓練：

(1) 建立模型與資料結構：

```
#導入相關套件
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
from keras.datasets import mnist # keras.datasets：載入 MNIST 資料集
from keras.models import Sequential # Keras：建立訓練模型
from keras.layers import Dense, Activation, Dropout
from keras.utils import np_utils
import numpy as np # Numpy：矩陣運算
%matplotlib inline # matplotlib.pyplot 將資料視覺化，可以圖表呈現結果
```

```
import matplotlib.pyplot as plt
(x_train_image, y_train_label), (x_test_image, y_test_label) =
mnist.load_data() # 呼叫 load_data() 載入 MNIST 資料集
nb_classes = 10 # 類別的數目
x_train_image = x_train_image.reshape(60000, 784).astype('float32')
x_test_image = x_test_image.reshape(10000, 784).astype('float32')
# 壓縮圖片顏色至 0 ~ 1
x_train_image /= 255
x_test_image /= 255
#依分類數量將圖片標籤轉換格式的陣列
y_train_cat = np_utils.to_categorical(y_train_label, nb_classes)
y_test_cat = np_utils.to_categorical(y_test_label, nb_classes)
model = Sequential()
model.add(Dense(50, input_shape=(784,)))
model.add(Dense(units=nb_classes))
model.add(Activation('softmax'))
# 定義定義損失函數、優化函數及成效衡量指標
model.compile(loss='categorical_crossentropy', optimizer='SGD',
metrics=['accuracy'])
model.summary()
```

執行結果：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 50)                39250
_________________________________________________________________
dense_2 (Dense)              (None, 10)                510
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 39,760
Trainable params: 39,760
Non-trainable params: 0
_________________________________________________________________
```

(2)開始訓練：

```
epochs = 10
history = model.fit(x_train_image, y_train_cat, epochs=epochs,
batch_size=128,verbose=1)
```

執行結果：

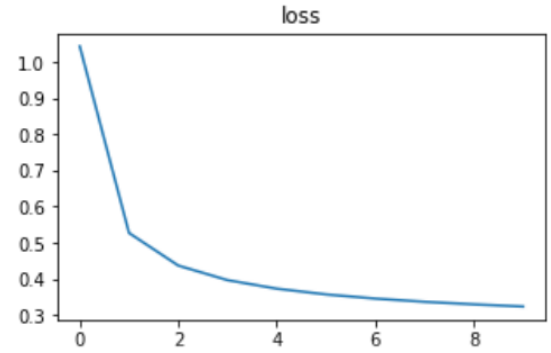
```
Epoch 1/10
60000/60000 [==============================] - 0s 8us/step - loss: 1.0253 - acc: 0.7352
Epoch 2/10
60000/60000 [==============================] - 0s 6us/step - loss: 0.5316 - acc: 0.8648
Epoch 3/10
60000/60000 [==============================] - 0s 6us/step - loss: 0.4414 - acc: 0.8815
Epoch 4/10
60000/60000 [==============================] - 0s 6us/step - loss: 0.4007 - acc: 0.8902
Epoch 5/10
60000/60000 [==============================] - 0s 6us/step - loss: 0.3766 - acc: 0.8963
Epoch 6/10
60000/60000 [==============================] - 0s 6us/step - loss: 0.3605 - acc: 0.9000
Epoch 7/10
60000/60000 [==============================] - 0s 6us/step - loss: 0.3485 - acc: 0.9032
Epoch 8/10
60000/60000 [==============================] - 0s 6us/step - loss: 0.3394 - acc: 0.9051
Epoch 9/10
60000/60000 [==============================] - 0s 6us/step - loss: 0.3320 - acc: 0.9067
Epoch 10/10
60000/60000 [==============================] - 0s 6us/step - loss: 0.3260 - acc: 0.9082
```

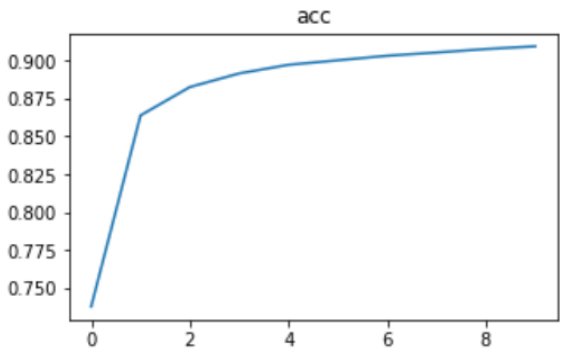
(3)執行後結果：

```
#訓練完成後設定周期數與相關設定
plt.figure(figsize=(8,6))
plt.plot(history.epoch,history.history['loss'])
plt.title('loss')
plt.figure(figsize=(8,6))
plt.plot(history.epoch,history.history['acc'])
plt.title('acc')
#測試資料後的評估模型準確率
scores = model.evaluate(x_test_image, y_test_cat, verbose=2)
print("accuracy = {:2.2f}%".format(scores[1]*100.0))
```

執行結果：

```
accuracy = 91.22%
```





2. 使用卷積神經網路(CNN)進行辨識訓練：

(1)建立模型與資料結構：

```
# 導入相關套件
import os
```

```
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np # Numpy：矩陣運算
import pandas as pd
from keras.utils import np_utils
from keras.datasets import mnist # keras.datasets：載入 MNIST 資料集
from keras.models import Sequential # Keras：建立訓練模型
from keras.layers import Dense,Dropout,Flatten,Conv2D,MaxPooling2D
(x_Train, y_Train), (x_Test, y_Test) = mnist.load_data()
# 影像特徵值轉換為 4 維矩陣
x_Train4D = x_Train.reshape(x_Train.shape[0], 28, 28,
1).astype('float32')
x_Test4D = x_Test.reshape(x_Test.shape[0], 28, 28, 1).astype('float32')
# 壓縮圖片顏色至 0 ~ 1
x_Train4D_normalize = x_Train4D / 255
x_Test4D_normalize = x_Test4D / 255
#依分類數量將圖片標籤轉換格式的陣列
y_TrainOne = np_utils.to_categorical(y_Train)
y_TestOne = np_utils.to_categorical(y_Test)
model = Sequential()
# 新增卷積層
model.add(Conv2D(filters=16, kernel_size=(5,5), padding='same',
input_shape=(28, 28,1), activation='relu'))
# 新增池化層
model.add(MaxPooling2D(pool_size=(2,2)))
# 新增卷積層
model.add(Conv2D(filters=16, kernel_size=(5,5), padding='same',
activation='relu'))
# 新增池化層
model.add(MaxPooling2D(pool_size=(2,2)))
# 防止過擬合
model.add(Dropout(0.25))
# 全連接層局部
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.summary()
```

執行結果：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 16)        416
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 16)        6416
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 16)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 7, 7, 16)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               100480
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
=================================================================
Total params: 108,602
Trainable params: 108,602
Non-trainable params: 0
_________________________________________________________________
```

(2)開始訓練：

```
# 定義定義損失函數、優化函數及成效衡量指標
model.compile(loss='categorical_crossentropy',
optimizer='adam',metrics=['acc'])
train_history = model.fit(x=x_Train4D_normalize,y=y_TrainOne,
validation_split=0.2,epochs=10, batch_size=300, verbose=1)
```

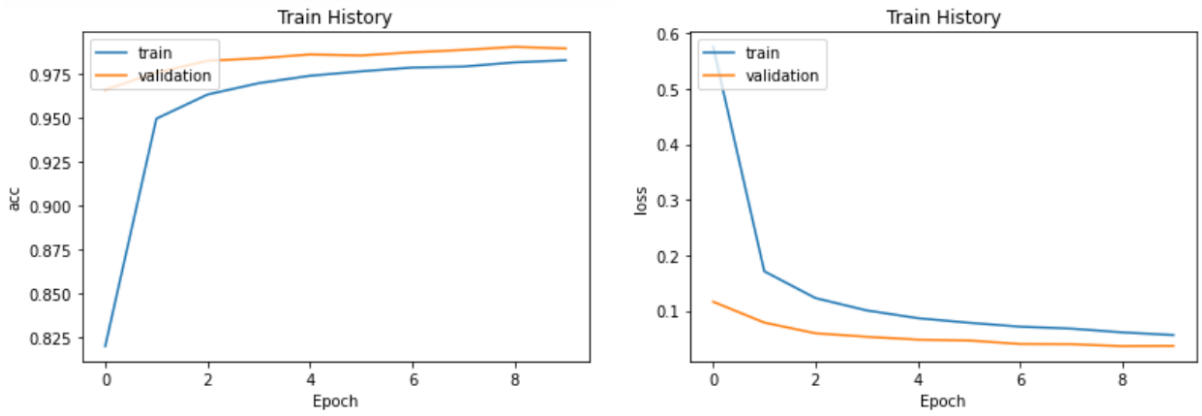
執行結果：

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 1539s 32ms/step - loss: 0.5754 - acc: 0.8200 - val_loss: 0.1162 - val_acc: 0.965
9
Epoch 2/10
48000/48000 [==============================] - 1524s 32ms/step - loss: 0.1710 - acc: 0.9495 - val_loss: 0.0785 - val_acc: 0.975
7
Epoch 3/10
48000/48000 [==============================] - 1530s 32ms/step - loss: 0.1228 - acc: 0.9632 - val_loss: 0.0593 - val_acc: 0.982
4
Epoch 4/10
48000/48000 [==============================] - 1524s 32ms/step - loss: 0.1005 - acc: 0.9697 - val_loss: 0.0530 - val_acc: 0.983
8
Epoch 5/10
48000/48000 [==============================] - 1531s 32ms/step - loss: 0.0865 - acc: 0.9739 - val_loss: 0.0479 - val_acc: 0.986
0
Epoch 6/10
48000/48000 [==============================] - 1532s 32ms/step - loss: 0.0783 - acc: 0.9764 - val_loss: 0.0463 - val_acc: 0.985
4
Epoch 7/10
48000/48000 [==============================] - 1529s 32ms/step - loss: 0.0712 - acc: 0.9786 - val_loss: 0.0400 - val_acc: 0.987
3
Epoch 8/10
48000/48000 [==============================] - 1532s 32ms/step - loss: 0.0677 - acc: 0.9791 - val_loss: 0.0397 - val_acc: 0.988
6
Epoch 9/10
48000/48000 [==============================] - 1529s 32ms/step - loss: 0.0609 - acc: 0.9815 - val_loss: 0.0362 - val_acc: 0.990
3
Epoch 10/10
48000/48000 [==============================] - 1531s 32ms/step - loss: 0.0562 - acc: 0.9827 - val_loss: 0.0366 - val_acc: 0.989
4
```

(3)訓練後結果：

```
def show_train_history(train_history, train, validation):
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.title('Train History')
    plt.ylabel(train)
    plt.xlabel('Epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
import matplotlib.pyplot as plt
show_train_history(train_history,'acc','val_acc')
show_train_history(train_history, 'loss', 'val_loss')
```

執行結果：



(4)預測值及混淆矩陣：

```
loss, acc = model.evaluate(x_Test4D_normalize, y_TestOne)
```

```
print("\nLoss: %.2f, Accuracy: %.2f%%" %(loss, acc* 100))
import pandas as pd
prediction=model.predict_classes(x_Test4D_normalize)
pd.crosstab(y_Test,prediction,rownames=['label'], colnames=['predict'])
```

執行結果：

```
10000/10000 [==============================] - 1s 144us/step

Loss: 0.03, Accuracy: 99.02%
```

predict	0	1	2	3	4	5	6	7	8	9
label										
0	975	0	0	0	0	0	2	1	2	0
1	0	1132	2	0	0	0	1	0	0	0
2	1	0	1028	0	0	0	0	3	0	0
3	0	0	2	999	0	5	0	2	2	0
4	0	0	1	0	976	0	0	0	0	5
5	1	0	0	2	0	886	2	1	0	0
6	4	2	0	0	1	1	949	0	1	0
7	0	1	7	1	0	1	0	1012	1	5
8	5	0	3	1	1	1	1	2	957	3
9	1	3	1	0	4	4	0	6	2	988

(5)建立模型與資料結構(調整模式)：

```
(x_Train , y_Train),(x_Test , y_Test) = mnist.load_data()# 呼叫
load_data() 載入 MNIST 資料集
# 影像特徵值轉換為 4 維矩陣
x_Train4D=x_Train.reshape(x_Train.shape[0],28,28,1).astype('float32')
x_Test4D=x_Test.reshape(x_Test.shape[0],28,28,1).astype('float32')
# 壓縮圖片顏色至 0 ~ 1
x_Train4D_normalize = x_Train4D / 255
x_Test4D_normalize = x_Test4D / 255
# 依分類數量將圖片標籤轉換格式的陣列
y_TrainOneHot = np_utils.to_categorical(y_Train)
y_TestOneHot = np_utils.to_categorical(y_Test)
model = Sequential()
# 新增卷積層
model.add(Conv2D(filters=16, kernel_size=(5,5), padding='same',
input_shape=(28,28,1), activation='relu'))
```

```
# 新增池化層
model.add(MaxPooling2D(pool_size=(2,2)))
# 新增卷積層
model.add(Conv2D( filters=36, kernel_size=(5,5), padding='same',
activation='relu'))
# 新增池化層
model.add(MaxPooling2D(pool_size=(2,2)))
# 防止過擬合
model.add(Dropout(0.25))
# 全連接層局部
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.summary()
```

執行結果：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 16)        416
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 36)        14436
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 36)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 7, 7, 36)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1764)              0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               225920
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
=================================================================
Total params: 242,062
Trainable params: 242,062
Non-trainable params: 0
_________________________________________________________________
```

(6)再次開始訓練：

```
# 定義定義損失函數、優化函數及成效衡量指標
model.compile(loss='categorical_crossentropy',
optimizer='adam',metrics=['accuracy'])
```

```
# 開始訓練
train_history=model.fit(x=x_Train4D_normalize,
y=y_TrainOneHot,validation_split=0.2, epochs=10,
batch_size=300,verbose=2)
```

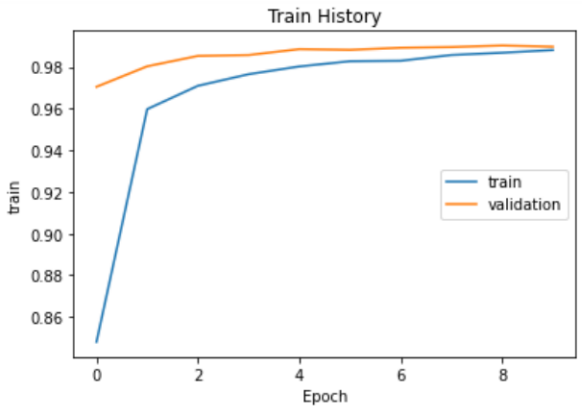
執行結果：

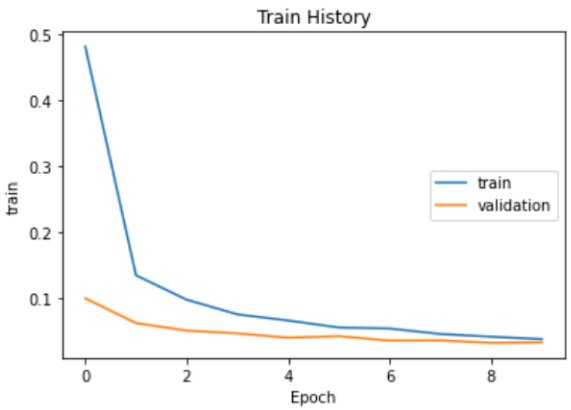
```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
 - 1520s - loss: 0.4809 - acc: 0.8480 - val_loss: 0.1001 - val_acc: 0.9705
Epoch 2/10
 - 1520s - loss: 0.1349 - acc: 0.9598 - val_loss: 0.0626 - val_acc: 0.9803
Epoch 3/10
 - 1521s - loss: 0.0979 - acc: 0.9710 - val_loss: 0.0512 - val_acc: 0.9853
Epoch 4/10
 - 1521s - loss: 0.0758 - acc: 0.9765 - val_loss: 0.0470 - val_acc: 0.9858
Epoch 5/10
 - 1522s - loss: 0.0665 - acc: 0.9803 - val_loss: 0.0404 - val_acc: 0.9886
Epoch 6/10
 - 1523s - loss: 0.0559 - acc: 0.9828 - val_loss: 0.0428 - val_acc: 0.9883
Epoch 7/10
 - 1517s - loss: 0.0544 - acc: 0.9830 - val_loss: 0.0361 - val_acc: 0.9893
Epoch 8/10
 - 1516s - loss: 0.0461 - acc: 0.9858 - val_loss: 0.0363 - val_acc: 0.9896
Epoch 9/10
 - 1523s - loss: 0.0420 - acc: 0.9869 - val_loss: 0.0327 - val_acc: 0.9903
Epoch 10/10
 - 1521s - loss: 0.0383 - acc: 0.9881 - val_loss: 0.0335 - val_acc: 0.9898
```

(7).訓練後結果(準確率變化)：

```
def show_train_history(train_history, train, validation):
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.title('Train History')
    plt.ylabel('train')
    plt.xlabel('Epoch')
    plt.legend(['train', 'validation'], loc='center right')
    plt.show()
import matplotlib.pyplot as plt
show_train_history(train_history, 'acc','val_acc')
show_train_history(train_history, 'loss','val_loss')
```

執行結果：





(8)混淆矩陣(改變過程)：

```
#訓練後結果
loss, accuracy = model.evaluate(x_Test4D_normalize , y_TestOneHot)
print( "\nLoss: %.2f, Accuracy: %.2f%%" % (loss, accuracy* 100 ))
#混淆矩陣
import pandas as pd
prediction=model.predict_classes(x_Test4D_normalize)
pd.crosstab(y_Test,prediction,rownames=['label'],
colnames=['predict'])
```

執行結果：

```
10000/10000 [==============================] - 2s 167us/step

Loss: 0.02, Accuracy: 99.18%
```

predict label	0	1	2	3	4	5	6	7	8	9
0	977	0	1	0	0	1	0	1	0	0
1	0	1134	1	0	0	0	0	0	0	0
2	1	0	1030	0	0	0	0	1	0	0
3	0	0	1	1006	0	1	0	1	1	0
4	0	1	0	0	977	0	0	1	0	3
5	1	0	0	11	0	877	2	0	0	1
6	5	2	1	0	1	3	946	0	0	0
7	0	2	7	1	0	0	0	1018	0	0
8	2	0	2	3	0	1	0	1	961	4
9	1	4	0	0	3	2	0	5	2	992

3. 使用循環神經網路(RNN)進行辨識訓練：

(1)建立模型與資料結構：

```
#導入相關套件
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
from keras.datasets import mnist # keras.datasets：載入 MNIST 資料集
from keras.models import Sequential # Keras：建立訓練模型
from keras.layers import Dense, Activation, Dropout
from keras.layers.recurrent import SimpleRNN, LSTM, GRU
from keras.utils import np_utils
%matplotlib inline
# matplotlib.pyplot 將資料視覺化，可以圖表呈現結果
import numpy as np # Numpy：矩陣運算
import matplotlib.pyplot as plt
(x_train , y_train),(x_test , y_test) = mnist.load_data() # 呼叫
load_data() 載入 MNIST 資料集
nb_classes = 10 # 類別的數目
img_rows, img_cols = 28, 28 # 圖片的長與寬
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# 壓縮圖片顏色至 0 ~ 1
x_train /= 255
x_test /= 255
# 依分類數量將圖片標籤轉換格式的陣列
y_train = np_utils.to_categorical(y_train, nb_classes)
y_test = np_utils.to_categorical(y_test, nb_classes)
nb_units = 50 # 隱藏層節點數值
model = Sequential() # 建立簡單的線性執行模型
model.add(LSTM(nb_units, input_shape=(img_rows, img_cols))) # 二個維度
model.add(Dense(units=nb_classes))
model.add(Activation('softmax'))
# 定義定義損失函數、優化函數及成效衡量指標
model.compile(loss='categorical_crossentropy', optimizer='SGD',
metrics=['accuracy'])
model.summary()
```

執行結果：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 50)                15800
_________________________________________________________________
dense_2 (Dense)              (None, 10)                510
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 16,310
Trainable params: 16,310
Non-trainable params: 0
_________________________________________________________________
```

(2)開始訓練：

```
epochs = 10
history = model.fit(x_train, y_train, epochs=epochs, batch_size=128, verbose=1)
```

執行結果：

```
Epoch 1/10
60000/60000 [==============================] - 6s 102us/step - loss: 2.2746 - acc: 0.2060
Epoch 2/10
60000/60000 [==============================] - 5s 88us/step - loss: 2.1689 - acc: 0.3270
Epoch 3/10
60000/60000 [==============================] - 5s 91us/step - loss: 1.9442 - acc: 0.3556
Epoch 4/10
60000/60000 [==============================] - 5s 92us/step - loss: 1.6969 - acc: 0.4666
Epoch 5/10
60000/60000 [==============================] - 5s 90us/step - loss: 1.4425 - acc: 0.5607
Epoch 6/10
60000/60000 [==============================] - 6s 94us/step - loss: 1.1935 - acc: 0.6274
Epoch 7/10
60000/60000 [==============================] - 6s 99us/step - loss: 0.9810 - acc: 0.6919
Epoch 8/10
60000/60000 [==============================] - 6s 97us/step - loss: 0.8103 - acc: 0.7493
Epoch 9/10
60000/60000 [==============================] - 6s 93us/step - loss: 0.6774 - acc: 0.7952
Epoch 10/10
60000/60000 [==============================] - 5s 91us/step - loss: 0.5737 - acc: 0.8280
```

(3)訓練後結果：

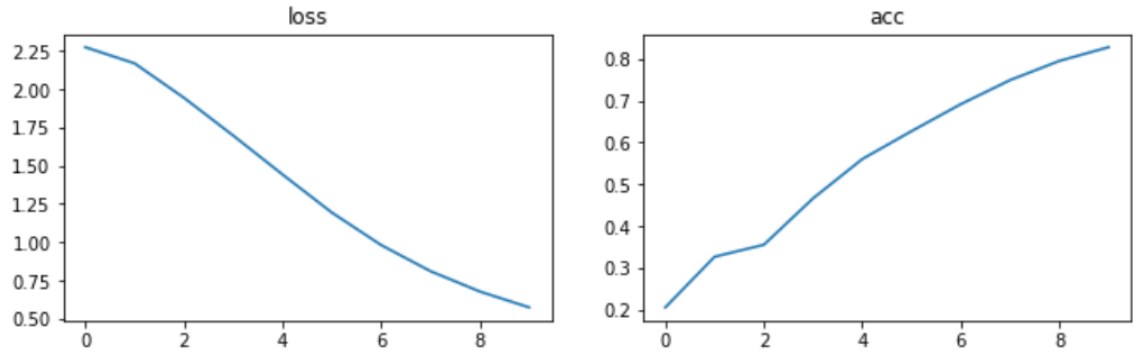
```
#訓練完成後設定周期數設定
plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['loss'])
plt.title('loss')

plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['acc'])
plt.title('acc');
#測試資料後的評估模型準確率
```

```
scores = model.evaluate(x_test, y_test, verbose=2)
print("accuracy = {:2.2f}%".format(scores[1]*100.0))
```

執行結果：結果偏低需要再次提高數值後訓練

```
accuracy = 82.71%
```



(4)建立模型與資料結構(調整模式)：

```
(x_train , y_train),(x_test , y_test) = mnist.load_data() # 呼叫
load_data() 載入 MNIST 資料集
nb_classes = 10 # 類別的數目
img_rows, img_cols = 28, 28 # 圖片的長與寬
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
#壓縮圖片顏色至 0 ~ 1
x_train /= 255
x_test /= 255
# 依分類數量將圖片標籤轉換格式的陣列
y_train = np_utils.to_categorical(y_train, nb_classes)
y_test = np_utils.to_categorical(y_test, nb_classes)
nb_units = 128 # 調整隱藏層節點數值
model = Sequential() # 建立簡單的線性執行模型
model.add(LSTM(nb_units, input_shape=(img_rows, img_cols))) # 新增 LSTM
維度
model.add(Dense(units=nb_classes))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy']) #優化函數改成 adam
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 128)               80384
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 81,674
Trainable params: 81,674
Non-trainable params: 0
_________________________________________________________________
```

(5)再次開始訓練：

```
epochs = 10
history = model.fit(x_train, y_train, epochs=epochs, batch_size=128, verbose=1)
```

執行結果：

```
Epoch 1/10
60000/60000 [==============================] - 9s 154us/step - loss: 0.5296 - acc: 0.8290
Epoch 2/10
60000/60000 [==============================] - 8s 141us/step - loss: 0.1543 - acc: 0.9533
Epoch 3/10
60000/60000 [==============================] - 8s 140us/step - loss: 0.1044 - acc: 0.9692
Epoch 4/10
60000/60000 [==============================] - 9s 143us/step - loss: 0.0811 - acc: 0.9756
Epoch 5/10
60000/60000 [==============================] - 9s 142us/step - loss: 0.0650 - acc: 0.9800
Epoch 6/10
60000/60000 [==============================] - 9s 144us/step - loss: 0.0525 - acc: 0.9842
Epoch 7/10
60000/60000 [==============================] - 9s 142us/step - loss: 0.0456 - acc: 0.9865
Epoch 8/10
60000/60000 [==============================] - 9s 144us/step - loss: 0.0416 - acc: 0.9877
Epoch 9/10
60000/60000 [==============================] - 9s 144us/step - loss: 0.0347 - acc: 0.9890
Epoch 10/10
60000/60000 [==============================] - 9s 151us/step - loss: 0.0319 - acc: 0.9902
```

(6)訓練後結果(準確率提高)：

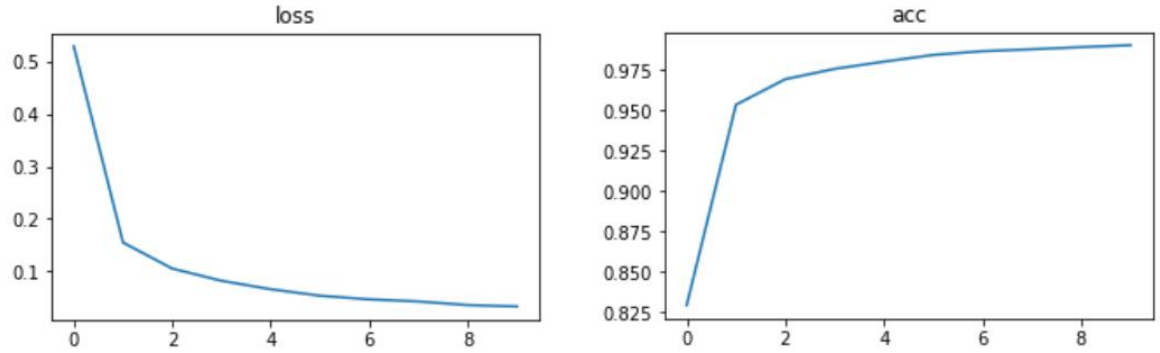
```
#訓練完成後設定周期數設定
plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['loss'])
plt.title('loss')

plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['acc'])
plt.title('acc');
#測試資料後的評估模型準確率
```

```
scores = model.evaluate(x_test, y_test, verbose=2)
print("accuracy = {:2.2f}%".format(scores[1]*100.0))
```

執行結果：

```
accuracy = 98.38%
```



四、結論

每一種的類神經網路都會有不同的執行結果，MLP 辨識出來的結果為 91.22%，使用了三大層面就能夠很快地辨識出結果出來，運算結構較為初始化。CNN 執行後會很就，必須要透過龐大的運算，對結構圖是分析的非常清楚，再由卷積層、池化層做出精緻的運算，就像是大張圖片濃縮成小張圖片，最後依據混淆矩陣來判斷出辨識出了的結果且數值為 99.18%。RNN 執行速度跟 MLP 差不多，需要調整隱藏層節點數值及 LSTM 維度數值後重新執行，結果就會提高至 98.38%。所以最好的辨識為 CNN。

五、參考

巨匠電腦 python 機器學習開發
巨匠電腦 python 深度學習開發
碁峰 Python 機器學習與深度學習特訓班