人工智慧專書報告

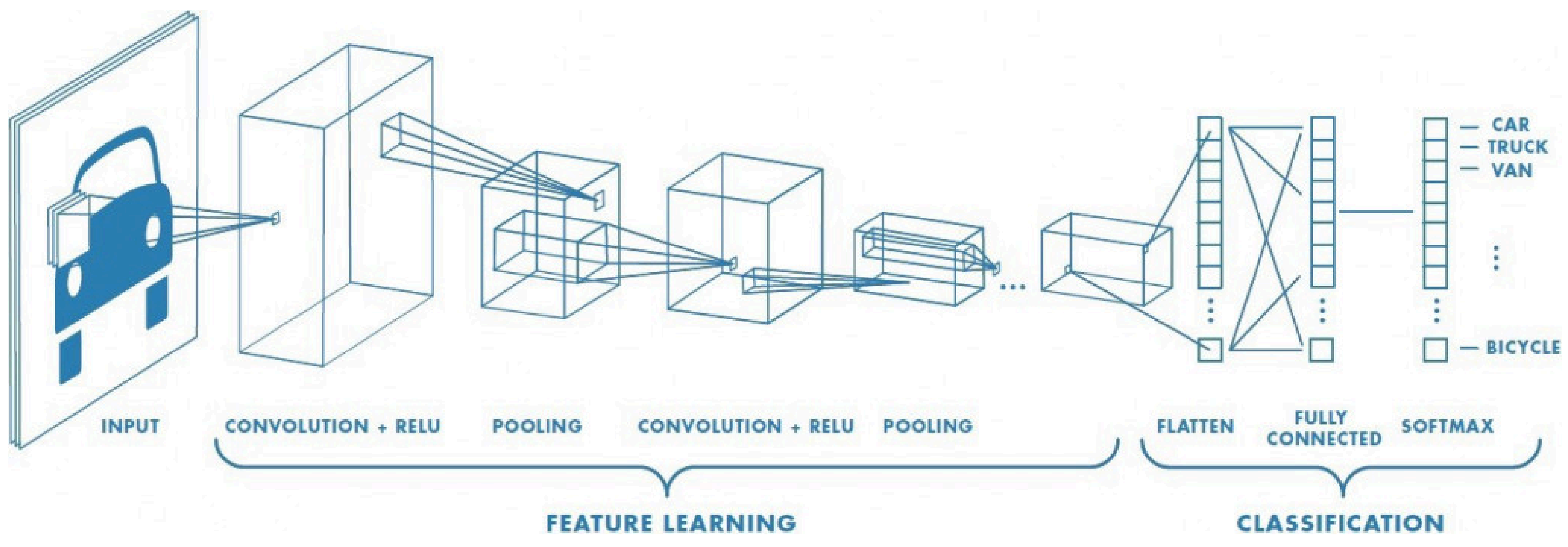
TensorFlow+Keras 深度學習人工智慧實務應用

第十九章.TensorFlow卷積神經網路CNN辨識手寫數字

第四組 ｜陳鈺昕 賴兆信 黃子晏 宋苡瑄 黃天芸

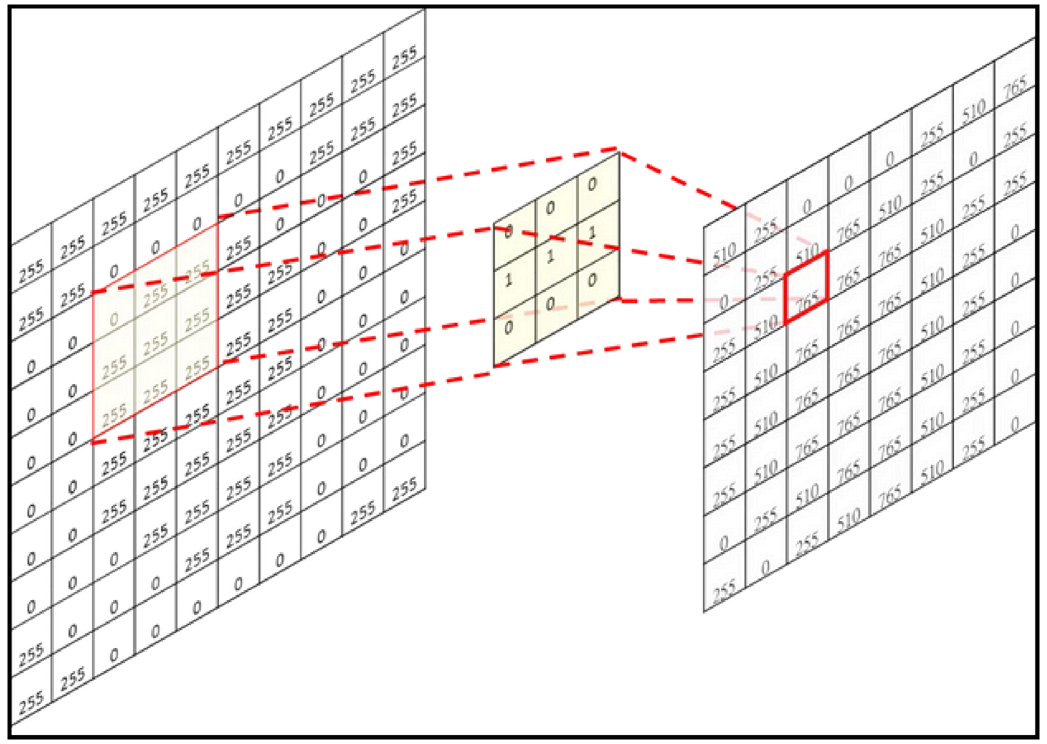
CNN

卷積神經網路（Convolutional Neural Network, CNN）是一種深度學習模型，在圖像辨識、物體偵測、影像處理等電腦視覺領域表現特別出色，同時也應用於自然語言處理、語音辨識等其他領域。它的設計靈感來自於人類視覺皮層的運作方式，能夠自動從輸入資料中學習並提取有用的特徵。

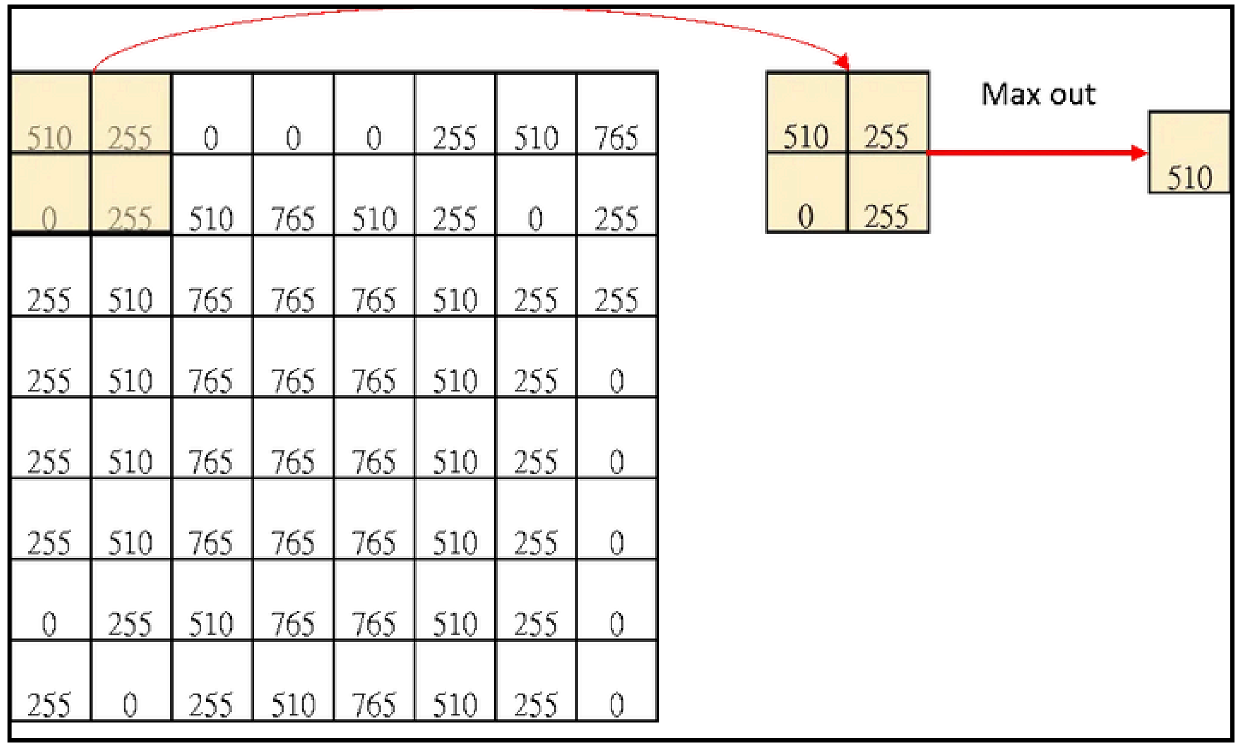


CNN

捲積



池化


510 255 0 0 0 255 510 765
0 255 510 765 510 255 0 255
255 510 765 765 765 510 255 255
255 510 765 765 765 510 255 0
255 510 765 765 765 510 255 0
255 510 765 765 765 510 255 0
0 255 510 765 765 510 255 0
255 0 255 510 765 510 255 0
510 255
0 255
Max out
510

MLP vs CNN 模型原理比較

項目	多元感知器 MLP	卷積神經網路 CNN
結構特性	完全連接，每層神經元與下一層全部連結	包含卷積層、池化層、全連接層
資料輸入需求	需將影像展平成一維向量（如 28x28 → 784）	保持影像的空間結構（2D）
空間特徵保留	❌ 無，空間資訊會被破壞	✔ 可擷取局部空間特徵
參數量	多（因為全部神經元互連）	少（參數共享）
訓練時間	通常較快，但效果有限	略慢但表現更好

流程

1.
導入庫和設置

2.
載入並預處理
MNIST 數據集

3.
構建 CNN 模型

4.
訓練模型

5.
進行預測

6.
評估性能

資料集說明（MNIST）

來源：TensorFlow Keras 內建 MNIST

訓練集：60,000 張手寫數字圖
測試集：10,000 張

每張圖：28x28 像素，灰階，數值範圍 0~255

資料處理

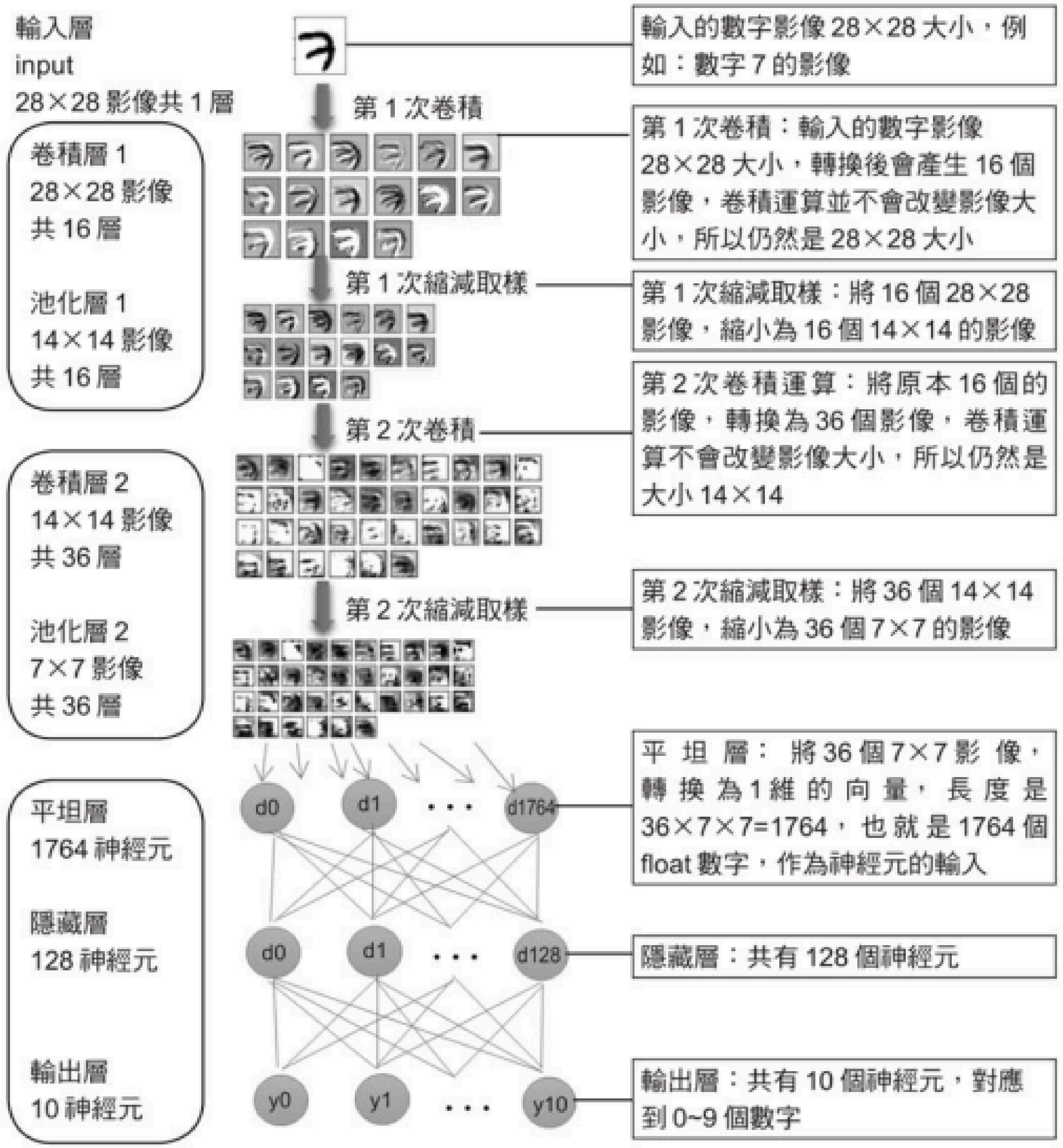
```python
# 正規化數據 (0~255 -> 0~1)
x_train = x_train / 255.0
x_test = x_test / 255.0

# 展平 (Flatten) 28x28 -> 784
# 為了與 TensorFlow 1.x 的占位符 (placeholder) 輸入格式匹配
x_train = x_train.reshape(-1, 784)
x_test = x_test.reshape(-1, 784)

# one-hot 編碼
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# 拆分訓練資料與驗證資料 (80% 訓練、20% 驗證)
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(
        x_train, y_train, test_size=0.2, random_state=42)

print("訓練資料形狀: ", x_train.shape, y_train.shape)
print("驗證資料形狀: ", x_val.shape, y_val.shape)
print("測試資料形狀: ", x_test.shape, y_test.shape)
```

正規化：
提高數值穩定性和加速梯度優化

展平：
適應模型輸入

one-hot 編碼：
適用於多類分類的 softmax 和交叉熵損失

模型架構


輸入層
input
28×28 影像共 1 層
卷積層 1
28×28 影像
共 16 層
池化層 1
14×14 影像
共 16 層
卷積層 2
14×14 影像
共 36 層
池化層 2
7×7 影像
共 36 層
平坦層
1764 神經元
隱藏層
128 神經元
輸出層
10 神經元
第 1 次卷積
第 1 次縮減取樣
第 2 次卷積
第 2 次縮減取樣
d0
d1
d1764
d128
y0
y1
y10
輸入的數字影像 28×28 大小，例如：數字 7 的影像
第 1 次卷積：輸入的數字影像 28×28 大小，轉換後會產生 16 個影像，卷積運算並不會改變影像大小，所以仍然是 28×28 大小
第 1 次縮減取樣：將 16 個 28×28 影像，縮小為 16 個 14×14 的影像
第 2 次卷積運算：將原本 16 個的影像，轉換為 36 個影像，卷積運算不會改變影像大小，所以仍然是大小 14×14
第 2 次縮減取樣：將 36 個 14×14 影像，縮小為 36 個 7×7 的影像
平坦層：將 36 個 7×7 影像，轉換為 1 維的向量，長度是 36×7×7=1764，也就是 1764 個 float 數字，作為神經元的輸入
隱藏層：共有 128 個神經元
輸出層：共有 10 個神經元，對應到 0~9 個數字

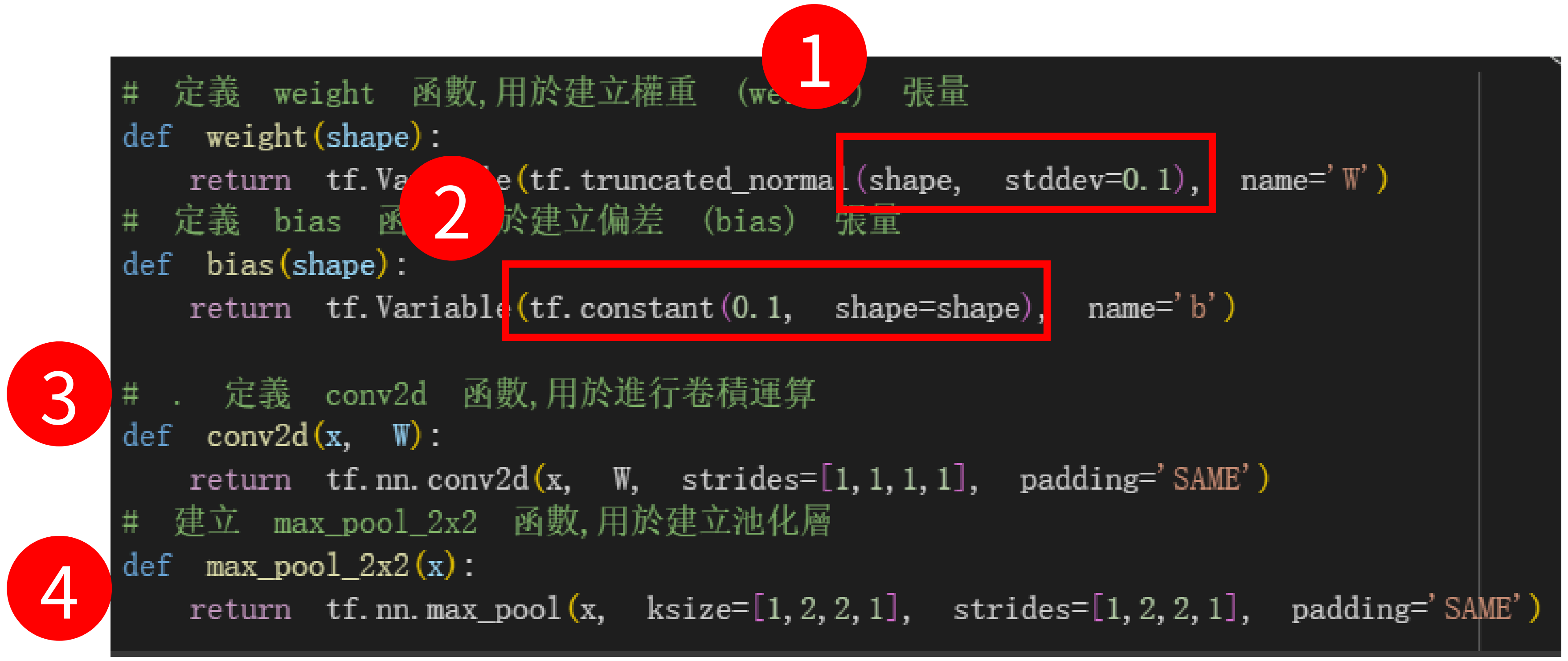
模型訓練設定

1.使用截斷正態分佈（標準差 = 0.1）初始化，小隨機值避免梯度消失或爆炸，標準差控制初始化範圍。

2.小的正偏置防止 ReLU 激活後神經元初始不活躍。

3.卷積提取空間特徵（如邊緣、紋理），對圖像識別至關重要

4.池化減少計算量，增強特徵魯棒性，通過降採樣防止過擬合。




```
# 定義 weight 函數,用於建立權重 (weight) 張量
def weight(shape):
    return tf.Variable(tf.truncated_normal(shape, stddev=0.1), name='W')
# 定義 bias 函數,用於建立偏差 (bias) 張量
def bias(shape):
    return tf.Variable(tf.constant(0.1, shape=shape), name='b')

# . 定義 conv2d 函數,用於進行卷積運算
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1,1,1,1], padding='SAME')
# 建立 max_pool_2x2 函數,用於建立池化層
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
```

模型訓練設定

```
#  輸入層(Input  Layer)
with  tf.name_scope('Input_Layer'):
    x  =  tf.placeholder("float",  shape=[None,  784],  name="x")
    x_image  =  tf.reshape(x,  [-1,  28,  28,  1])

#建立卷積層1
with  tf.name_scope('C1_Conv'):
    W1  =  weight([5,  5,  1,  16])
    b1  =  bias([16])
    Conv1  =  conv2d(x_image,  W1)  +  b1
    C1_Conv  =  tf.nn.relu(Conv1)

#  建立池化層1
with  tf.name_scope('C1_Pool'):
    C1_Pool  =  max_pool_2x2(C1_Conv)

#  卷積層2
with  tf.name_scope('C2_Conv'):
    W2  =  weight([5,  5,  16,  36])
    b2  =  bias([36])
    Conv2  =  conv2d(C1_Pool,  W2)  +  b2
    C2_Conv  =  tf.nn.relu(Conv2)

#  建立池化層2
with  tf.name_scope('C2_Pool'):
    C2_Pool  =  max_pool_2x2(C2_Conv)
```

```
#  建立隱藏層
with  tf.name_scope('D_Hidden_Layer'):
    W3  =  weight([1764,  128])
    b3  =  bias([128])
    D_Hidden  =  tf.nn.relu(tf.matmul(D_Flat,  W3)  +  b3)
    keep_prob  =  tf.placeholder(tf.float32)
    D_Hidden_Dropout  =  tf.nn.dropout(D_Hidden,  keep_prob)

#  建立輸出層(Output_Layer  )
with  tf.name_scope('Output_Layer'):
    W4  =  weight([128,  10])
    b4  =  bias([10])
    y_predict  =  tf.nn.softmax(tf.matmul(D_Hidden_Dropout,  W4)  +  b4)
```

模型訓練

```python
trainEpochs = 10
batchSize = 100
totalBatchs = int(len(x_train) / batchSize)

epoch_list, accuracy_list, loss_list = [], [], []
val_loss_list, val_accuracy_list = [], []

sess = tf.Session()
sess.run(tf.global_variables_initializer())

startTime = time()
for epoch in range(trainEpochs):
    for i in range(totalBatchs):
        batch_x = x_train[i*batchSize:(i+1)*batchSize]
        batch_y = y_train[i*batchSize:(i+1)*batchSize]
        sess.run(optimizer, feed_dict={x: batch_x, y_label: batch_y, keep_prob: 0.8})

    # 訓練集表現 (loss + acc)
    train_loss, train_acc = sess.run([loss_function, accuracy],
        feed_dict={x: x_train, y_label: y_train, keep_prob: 1.0})
    loss_list.append(train_loss)
    accuracy_list.append(train_acc)

    # 驗證集表現 (val_loss + val_acc)
    val_loss, val_acc = sess.run([loss_function, accuracy],
        feed_dict={x: x_val, y_label: y_val, keep_prob: 1.0})
    val_loss_list.append(val_loss)
    val_accuracy_list.append(val_acc)

    epoch_list.append(epoch)

    print("Train Epoch:", '%02d' % (epoch+1),
          "Train Loss=", "{:.9f}".format(train_loss),
          "Train Accuracy=", train_acc,
          "Val Loss=", "{:.9f}".format(val_loss),
          "Val Accuracy=", val_acc)

    # Early stopping 檢查條件
    if epoch > 3 and val_loss > val_loss_list[-2]:
            print("Validation loss increasing, early stop!")
            break

duration = time() - startTime
print("Train Finished takes:", duration)
```

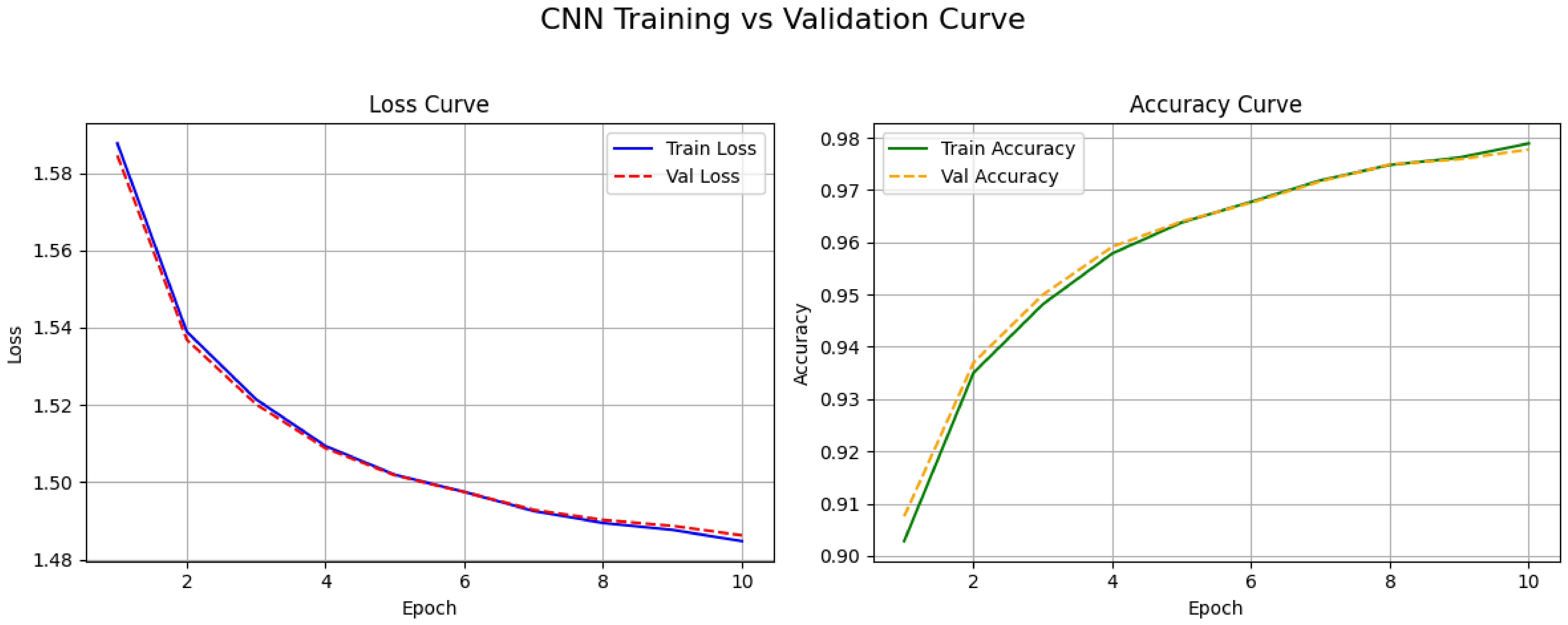
```
Train Epoch: 01 Train Loss= 1.587630153 Train Accuracy= 0.90283334 Val Loss= 1.584499478 Val Accuracy= 0.90758336
Train Epoch: 02 Train Loss= 1.538931966 Train Accuracy= 0.93504167 Val Loss= 1.536863089 Val Accuracy= 0.937
Train Epoch: 03 Train Loss= 1.521410465 Train Accuracy= 0.94816667 Val Loss= 1.520141006 Val Accuracy= 0.95
Train Epoch: 04 Train Loss= 1.509340763 Train Accuracy= 0.957875 Val Loss= 1.508755803 Val Accuracy= 0.95916665
Train Epoch: 05 Train Loss= 1.501906872 Train Accuracy= 0.96379167 Val Loss= 1.501762986 Val Accuracy= 0.964
Train Epoch: 06 Train Loss= 1.497470737 Train Accuracy= 0.96775 Val Loss= 1.497425675 Val Accuracy= 0.96758336
Train Epoch: 07 Train Loss= 1.492524624 Train Accuracy= 0.97185415 Val Loss= 1.492866039 Val Accuracy= 0.9716667
Train Epoch: 08 Train Loss= 1.489446998 Train Accuracy= 0.97479165 Val Loss= 1.490227580 Val Accuracy= 0.97491664
Train Epoch: 09 Train Loss= 1.487636447 Train Accuracy= 0.9762083 Val Loss= 1.488702774 Val Accuracy= 0.9759167
Train Epoch: 10 Train Loss= 1.484732151 Train Accuracy= 0.97891665 Val Loss= 1.486229539 Val Accuracy= 0.97775
Train Finished takes: 22.106186151504517
```

訓練數據的迭代10次

每個批次包含 100 個樣本

每個 epoch 的批次數480

訓練過程結果圖


CNN Training vs Validation Curve
Loss Curve
Train Loss
Val Loss
Loss
Epoch
1.58
1.56
1.54
1.52
1.50
1.48
2
4
6
8
10
Accuracy Curve
Train Accuracy
Val Accuracy
Accuracy
Epoch
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.90

模型預測與準確度

[整體]

準確率: 0.9783

精確率: 0.9783439673661289

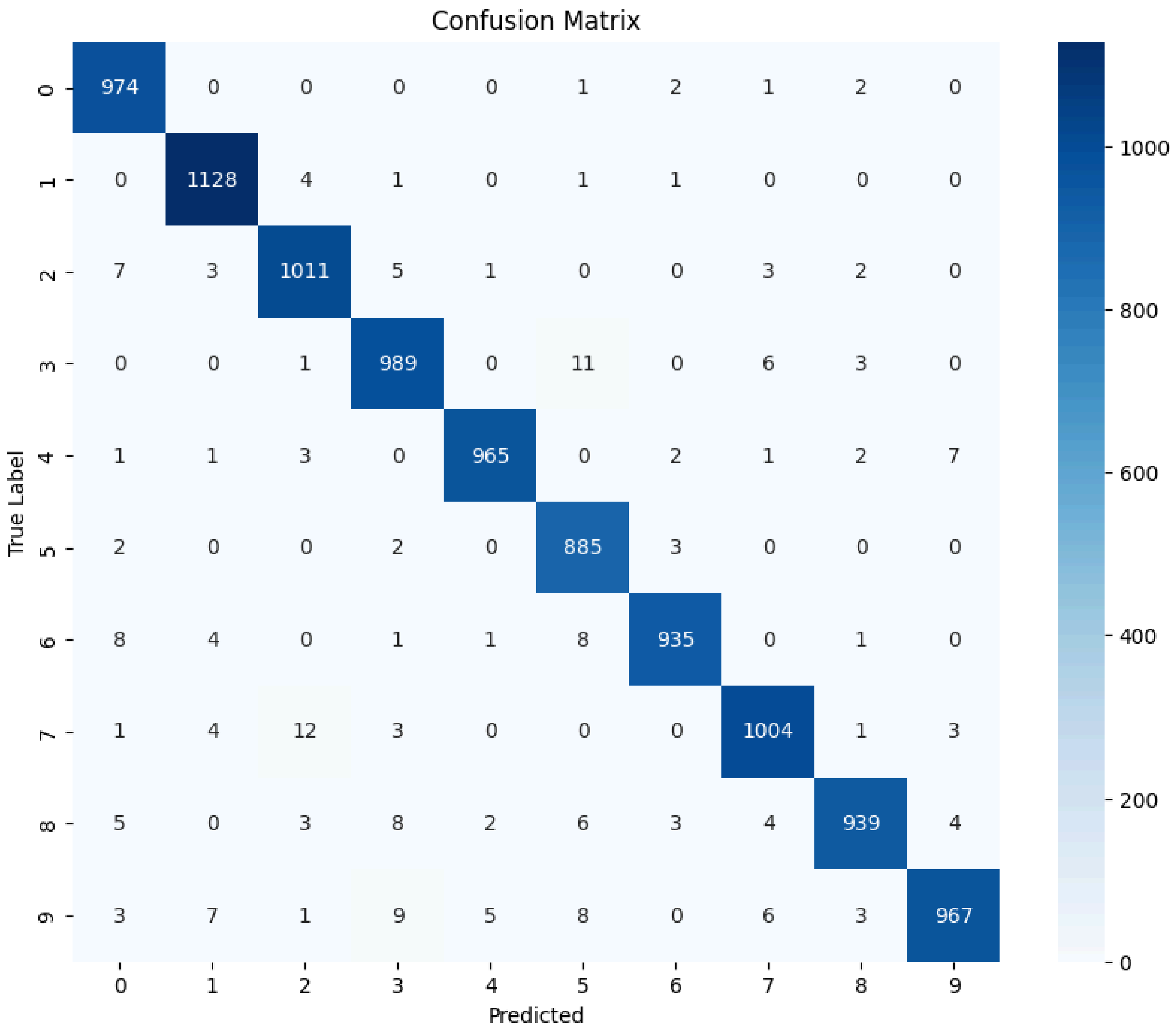
召回率: 0.9782713465427223

F1 Score: 0.9782570735544514

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.99	0.99	0.99	1135
2	0.98	0.97	0.97	1032
3	0.97	0.98	0.97	1010
4	0.99	0.98	0.99	982
5	0.99	0.98	0.98	892
6	0.98	0.98	0.98	958
7	0.98	0.97	0.97	1028
8	0.96	0.98	0.97	974
9	0.99	0.96	0.97	1009
accuracy			0.98	10000

混淆矩陣分析



[3]，被誤判為 "5" (11 次)

這是比較明顯的一個混淆，數字 3 和 5 在下半部寫法比較相似

[7]，被誤判為 "2" (12 次)

這是另一個比較顯著的混淆點，可能是因為書寫位置比較類似

結論與延伸

CNN 成功學習手寫數字圖像特徵
準確率高，能有效應用於分類任務

延伸方向：

- 測試 Fashion MNIST 或更複雜資料
- 調整模型參數、增加卷積層數
- 使用 TensorFlow 2.x Keras 架構實

主要參考資料

《TensorFlow+Keras 深度學習人工智慧實務應用》

其他參考資料

https://en.wikipedia.org/wiki/MNIST_database

https://arxiv.org/abs/2008.10400

https://en.wikipedia.org/wiki/Convolutional_neural_network

https://www.reddit.com/r/tensorflow/comments/gbmvjm/tfkeras_yielding_lower_accuracy_than_keras/

https://arxiv.org/abs/1003.0358

https://ithelp.ithome.com.tw/m/articles/10304942

https://chih-sheng-huang821.medium.com/

結束