# 機器學習系統設計實務與應用

# HW2

組別 ： 為什麼要醬組

學號 : B103012002

姓名 ： 林凡皓

# 目錄

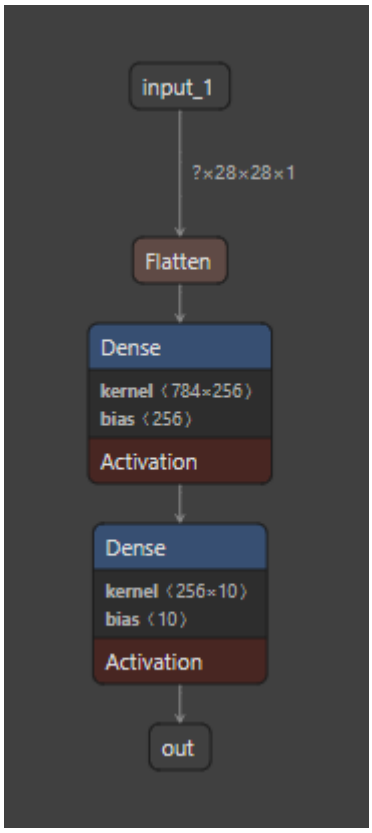
# I. MNIST dataset

## 一、NN

### 1. Model 1：

- 網路架構：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
fc1 (Dense)                  (None, 256)               200960
_________________________________________________________________
out (Dense)                  (None, 10)                2570
=================================================================
Total params: 203,530
Trainable params: 203,530
Non-trainable params: 0
```




建立一個一層 hidden layer 以及一層 flatten layer 的神經網路，hidden layer 的 activation 使用 relu，output layer 的 activation 使用 softmax，optimizer 使用 adam，loss 為 categorical

crossentropy。

- 訓練、測試準確度 ：

  關於資料的處理，我將 MNIST data set 先分成 test data 和 train data，其大小如下圖。

```
train_data.shape: (55000, 28, 28)
train_label.shape: (55000, 10)
test_data.shape: (10000, 28, 28)
test_label.shape: (10000, 10)
```

接著再將 train data 中的 1/2 設定為真正的 train data，1/5 為 validation data。除此之外，為了加速訓練將 batch size = 256，training epochs 為 30，訓練過程的準確度如下圖

```
Epoch 1/30
22000/22000 [==============================] - 1s 29us/step - loss: 0.6516 - acc: 0.8217 - val_loss: 0.3498 - val_acc: 0.9002
Epoch 2/30
22000/22000 [==============================] - 0s 22us/step - loss: 0.3020 - acc: 0.9126 - val_loss: 0.2731 - val_acc: 0.9231
Epoch 3/30
22000/22000 [==============================] - 0s 22us/step - loss: 0.2342 - acc: 0.9324 - val_loss: 0.2302 - val_acc: 0.9340
Epoch 4/30
22000/22000 [==============================] - 0s 22us/step - loss: 0.1897 - acc: 0.9455 - val_loss: 0.2077 - val_acc: 0.9373
Epoch 5/30
22000/22000 [==============================] - 0s 21us/step - loss: 0.1582 - acc: 0.9549 - val_loss: 0.1846 - val_acc: 0.9433
Epoch 6/30
22000/22000 [==============================] - 0s 21us/step - loss: 0.1327 - acc: 0.9618 - val_loss: 0.1717 - val_acc: 0.9478
Epoch 7/30
22000/22000 [==============================] - 1s 31us/step - loss: 0.1095 - acc: 0.9699 - val_loss: 0.1569 - val_acc: 0.9524
Epoch 8/30
22000/22000 [==============================] - 0s 21us/step - loss: 0.0942 - acc: 0.9748 - val_loss: 0.1512 - val_acc: 0.9533
Epoch 9/30
22000/22000 [==============================] - 1s 44us/step - loss: 0.0807 - acc: 0.9784 - val_loss: 0.1435 - val_acc: 0.9522
Epoch 10/30
22000/22000 [==============================] - 1s 62us/step - loss: 0.0705 - acc: 0.9815 - val_loss: 0.1401 - val_acc: 0.9571
Epoch 11/30
22000/22000 [==============================] - 1s 58us/step - loss: 0.0594 - acc: 0.9860 - val_loss: 0.1365 - val_acc: 0.9571
Epoch 12/30
22000/22000 [==============================] - 1s 59us/step - loss: 0.0511 - acc: 0.9882 - val_loss: 0.1293 - val_acc: 0.9595
Epoch 13/30
22000/22000 [==============================] - 1s 59us/step - loss: 0.0443 - acc: 0.9901 - val_loss: 0.1313 - val_acc: 0.9584
Epoch 14/30
22000/22000 [==============================] - 1s 61us/step - loss: 0.0386 - acc: 0.9923 - val_loss: 0.1291 - val_acc: 0.9595
Epoch 15/30
22000/22000 [==============================] - 1s 61us/step - loss: 0.0340 - acc: 0.9930 - val_loss: 0.1282 - val_acc: 0.9604
Epoch 16/30
22000/22000 [==============================] - 1s 61us/step - loss: 0.0288 - acc: 0.9950 - val_loss: 0.1274 - val_acc: 0.9620
Epoch 17/30
22000/22000 [==============================] - 1s 63us/step - loss: 0.0252 - acc: 0.9965 - val_loss: 0.1290 - val_acc: 0.9602
Epoch 18/30
22000/22000 [==============================] - 1s 61us/step - loss: 0.0223 - acc: 0.9967 - val_loss: 0.1245 - val_acc: 0.9602
Epoch 19/30
22000/22000 [==============================] - 1s 63us/step - loss: 0.0194 - acc: 0.9976 - val_loss: 0.1263 - val_acc: 0.9613
Epoch 20/30
22000/22000 [==============================] - 1s 58us/step - loss: 0.0169 - acc: 0.9983 - val_loss: 0.1270 - val_acc: 0.9602
Epoch 21/30
22000/22000 [==============================] - 1s 57us/step - loss: 0.0151 - acc: 0.9987 - val_loss: 0.1284 - val_acc: 0.9616
Epoch 22/30
22000/22000 [==============================] - 1s 66us/step - loss: 0.0139 - acc: 0.9989 - val_loss: 0.1266 - val_acc: 0.9622
Epoch 23/30
22000/22000 [==============================] - 1s 65us/step - loss: 0.0118 - acc: 0.9995 - val_loss: 0.1259 - val_acc: 0.9631
Epoch 24/30
22000/22000 [==============================] - 1s 61us/step - loss: 0.0109 - acc: 0.9994 - val_loss: 0.1304 - val_acc: 0.9615
Epoch 25/30
22000/22000 [==============================] - 1s 62us/step - loss: 0.0094 - acc: 0.9997 - val_loss: 0.1281 - val_acc: 0.9624
Epoch 26/30
22000/22000 [==============================] - 1s 63us/step - loss: 0.0084 - acc: 0.9998 - val_loss: 0.1281 - val_acc: 0.9616
Epoch 27/30
22000/22000 [==============================] - 1s 57us/step - loss: 0.0075 - acc: 0.9998 - val_loss: 0.1320 - val_acc: 0.9629
Epoch 28/30
22000/22000 [==============================] - 2s 68us/step - loss: 0.0070 - acc: 0.9999 - val_loss: 0.1293 - val_acc: 0.9640
Epoch 29/30
22000/22000 [==============================] - 1s 62us/step - loss: 0.0062 - acc: 0.9999 - val_loss: 0.1297 - val_acc: 0.9633
Epoch 30/30
22000/22000 [==============================] - 1s 60us/step - loss: 0.0055 - acc: 0.9999 - val_loss: 0.1323 - val_acc: 0.9635
```

Test accuracy 可以達到 99.99%的準確度，validation accuracy 可

以達到 96.3%左右的準確度。

訓練完成後，對一開始創建的 test data 做測試，結果如下 ：

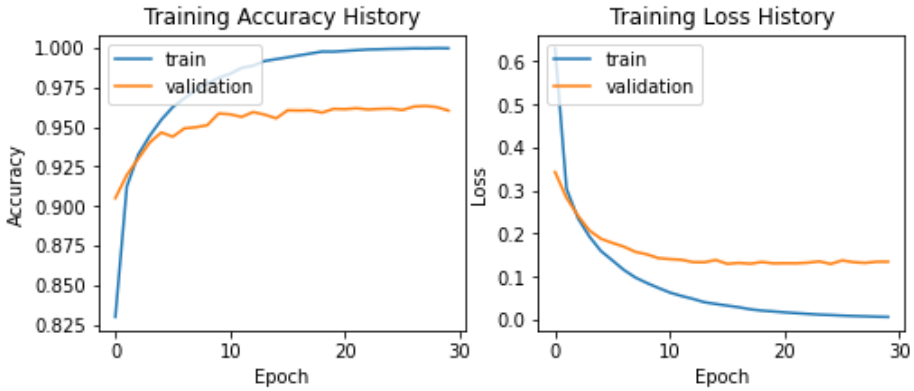
```
10000/10000 [==============================] - 1s 142us/step
Test loss :   0.11302  Test accuracy :  0.96850
```

準確度可以到達 96.85 %。

- 討論 ：

(1) Overfitting :

下圖為將訓練過程的 loss 與 accuracy 視覺化後的結果




由上圖可以看出，最一開始不論是 training 或是 validation 的 loss 與 accuracy 接迅速的變好，但是到後來整體優化會趨緩，代表說訓練已達飽和了，甚至有 overfitting 的情況出現。為了確認是否有 overfitting 的問題，將 training epochs 設成 20 觀察結果。訓練過程如下圖

```
Epoch 1/20
5500/5500 [==============================] - 1s 135us/step - loss: 0.0605 - acc: 0.9911 - val_loss: 0.2550 - val_acc: 0.9207
Epoch 2/20
5500/5500 [==============================] - 1s 134us/step - loss: 0.0546 - acc: 0.9925 - val_loss: 0.2532 - val_acc: 0.9258
Epoch 3/20
5500/5500 [==============================] - 1s 126us/step - loss: 0.0491 - acc: 0.9938 - val_loss: 0.2561 - val_acc: 0.9251
Epoch 4/20
5500/5500 [==============================] - 1s 153us/step - loss: 0.0452 - acc: 0.9940 - val_loss: 0.2514 - val_acc: 0.9251
Epoch 5/20
5500/5500 [==============================] - 1s 129us/step - loss: 0.0413 - acc: 0.9962 - val_loss: 0.2528 - val_acc: 0.9236
Epoch 6/20
5500/5500 [==============================] - 1s 149us/step - loss: 0.0379 - acc: 0.9965 - val_loss: 0.2503 - val_acc: 0.9295
Epoch 7/20
5500/5500 [==============================] - 1s 121us/step - loss: 0.0350 - acc: 0.9973 - val_loss: 0.2502 - val_acc: 0.9280
Epoch 8/20
5500/5500 [==============================] - 1s 140us/step - loss: 0.0317 - acc: 0.9976 - val_loss: 0.2520 - val_acc: 0.9280
Epoch 9/20
5500/5500 [==============================] - 1s 158us/step - loss: 0.0297 - acc: 0.9980 - val_loss: 0.2498 - val_acc: 0.9265
Epoch 10/20
5500/5500 [==============================] - 1s 134us/step - loss: 0.0266 - acc: 0.9980 - val_loss: 0.2508 - val_acc: 0.9280
Epoch 11/20
5500/5500 [==============================] - 1s 133us/step - loss: 0.0244 - acc: 0.9989 - val_loss: 0.2528 - val_acc: 0.9302
Epoch 12/20
5500/5500 [==============================] - 1s 122us/step - loss: 0.0229 - acc: 0.9987 - val_loss: 0.2538 - val_acc: 0.9287
Epoch 13/20
5500/5500 [==============================] - 1s 135us/step - loss: 0.0213 - acc: 0.9995 - val_loss: 0.2545 - val_acc: 0.9295
Epoch 14/20
5500/5500 [==============================] - 1s 127us/step - loss: 0.0201 - acc: 0.9995 - val_loss: 0.2542 - val_acc: 0.9295
Epoch 15/20
5500/5500 [==============================] - 1s 148us/step - loss: 0.0182 - acc: 0.9998 - val_loss: 0.2542 - val_acc: 0.9287
Epoch 16/20
5500/5500 [==============================] - 1s 127us/step - loss: 0.0172 - acc: 0.9996 - val_loss: 0.2551 - val_acc: 0.9324
Epoch 17/20
5500/5500 [==============================] - 1s 142us/step - loss: 0.0156 - acc: 1.0000 - val_loss: 0.2548 - val_acc: 0.9316
Epoch 18/20
5500/5500 [==============================] - 1s 132us/step - loss: 0.0149 - acc: 1.0000 - val_loss: 0.2576 - val_acc: 0.9309
Epoch 19/20
5500/5500 [==============================] - 1s 137us/step - loss: 0.0138 - acc: 0.9998 - val_loss: 0.2589 - val_acc: 0.9287
Epoch 20/20
5500/5500 [==============================] - 1s 146us/step - loss: 0.0130 - acc: 1.0000 - val_loss: 0.2645 - val_acc: 0.9295
```

測試結果如下圖

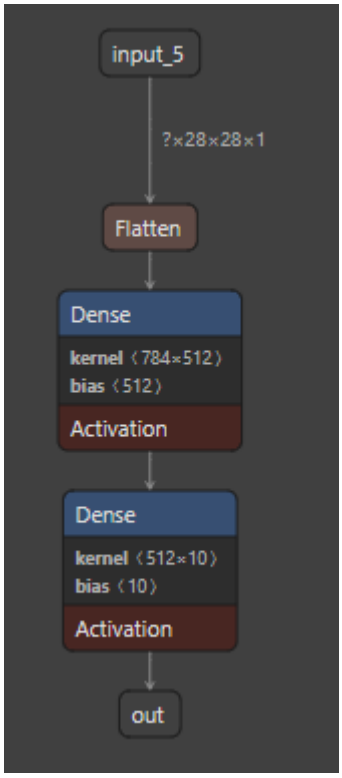
```
10000/10000 [==============================] - 1s 142us/step
Test loss :   0.23303  Test accuracy :  0.92860
```

經由比較會發現，降低 training epochs 後準確度下降了，代表說第一次的訓練 overfitting 並沒有很嚴重，多次訓練對於準確度仍有提升。

2. Model 2 :

➢ 網路架構 :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 784)               0
_________________________________________________________________
fc1 (Dense)                  (None, 512)               401920
_________________________________________________________________
out (Dense)                  (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
```




建立一個一層 hidden layer 以及一層 flatten layer 的神經網路，hidden layer 的 activation 使用 relu，output layer 的 activation 使用 softmax，optimizer 使用 adam，loss 為 categorical crossentropy。和 model 1 的差別在於 hidden layer 的神經元數目由 256 改為 512。

- 訓練、測試準確度 ：

關於資料的處理，我將 MNIST data set 先分成 test data 和 train data，其大小如下圖。

```
train_data.shape: (55000, 28, 28)
train_label.shape: (55000, 10)
test_data.shape: (10000, 28, 28)
test_label.shape: (10000, 10)
```

接著再將 train data 中的 1/2 設定為真正的 train data，1/5 為 validation data。除此之外，為了加速訓練將 batch size = 256，training epochs 為 30，訓練過程的準確度如下圖

```
Train on 2750 samples, validate on 688 samples
Epoch 1/30
2750/2750 [==============================] - 2s 824us/step - loss: 1.5079 - acc: 0.6222 - val_loss: 0.8334 - val_acc: 0.7951
Epoch 2/30
2750/2750 [==============================] - 0s 170us/step - loss: 0.6279 - acc: 0.8335 - val_loss: 0.5272 - val_acc: 0.8401
Epoch 3/30
2750/2750 [==============================] - 1s 182us/step - loss: 0.4419 - acc: 0.8811 - val_loss: 0.4598 - val_acc: 0.8576
Epoch 4/30
2750/2750 [==============================] - 1s 195us/step - loss: 0.3601 - acc: 0.9051 - val_loss: 0.4249 - val_acc: 0.8648
Epoch 5/30
2750/2750 [==============================] - 1s 214us/step - loss: 0.3035 - acc: 0.9218 - val_loss: 0.3987 - val_acc: 0.8765
Epoch 6/30
2750/2750 [==============================] - 1s 189us/step - loss: 0.2670 - acc: 0.9320 - val_loss: 0.3907 - val_acc: 0.8779
Epoch 7/30
2750/2750 [==============================] - 1s 195us/step - loss: 0.2352 - acc: 0.9404 - val_loss: 0.3734 - val_acc: 0.8808
Epoch 8/30
2750/2750 [==============================] - 1s 197us/step - loss: 0.2077 - acc: 0.9491 - val_loss: 0.3562 - val_acc: 0.8866
Epoch 9/30
2750/2750 [==============================] - 0s 181us/step - loss: 0.1838 - acc: 0.9531 - val_loss: 0.3505 - val_acc: 0.8852
Epoch 10/30
2750/2750 [==============================] - 0s 179us/step - loss: 0.1625 - acc: 0.9604 - val_loss: 0.3402 - val_acc: 0.8866
Epoch 11/30
2750/2750 [==============================] - 1s 184us/step - loss: 0.1446 - acc: 0.9709 - val_loss: 0.3355 - val_acc: 0.8939
Epoch 12/30
2750/2750 [==============================] - 1s 198us/step - loss: 0.1311 - acc: 0.9745 - val_loss: 0.3283 - val_acc: 0.8881
Epoch 13/30
2750/2750 [==============================] - 1s 203us/step - loss: 0.1132 - acc: 0.9822 - val_loss: 0.3213 - val_acc: 0.8939
Epoch 14/30
2750/2750 [==============================] - 1s 184us/step - loss: 0.1023 - acc: 0.9862 - val_loss: 0.3214 - val_acc: 0.8939
Epoch 15/30
2750/2750 [==============================] - 1s 207us/step - loss: 0.0918 - acc: 0.9873 - val_loss: 0.3209 - val_acc: 0.8939
Epoch 16/30
2750/2750 [==============================] - 1s 214us/step - loss: 0.0815 - acc: 0.9880 - val_loss: 0.3112 - val_acc: 0.8939
Epoch 17/30
2750/2750 [==============================] - 1s 197us/step - loss: 0.0716 - acc: 0.9924 - val_loss: 0.3060 - val_acc: 0.8997
Epoch 18/30
2750/2750 [==============================] - 1s 182us/step - loss: 0.0632 - acc: 0.9931 - val_loss: 0.3046 - val_acc: 0.9026
Epoch 19/30
2750/2750 [==============================] - 0s 174us/step - loss: 0.0568 - acc: 0.9942 - val_loss: 0.3050 - val_acc: 0.8968
Epoch 20/30
2750/2750 [==============================] - 1s 191us/step - loss: 0.0508 - acc: 0.9953 - val_loss: 0.3015 - val_acc: 0.9012
Epoch 21/30
2750/2750 [==============================] - 1s 194us/step - loss: 0.0458 - acc: 0.9971 - val_loss: 0.3124 - val_acc: 0.9012
Epoch 22/30
2750/2750 [==============================] - 1s 186us/step - loss: 0.0425 - acc: 0.9975 - val_loss: 0.3023 - val_acc: 0.9012
Epoch 23/30
2750/2750 [==============================] - 1s 196us/step - loss: 0.0377 - acc: 0.9982 - val_loss: 0.2966 - val_acc: 0.9026
Epoch 24/30
2750/2750 [==============================] - 1s 210us/step - loss: 0.0340 - acc: 0.9985 - val_loss: 0.3017 - val_acc: 0.9026
Epoch 25/30
2750/2750 [==============================] - 1s 200us/step - loss: 0.0309 - acc: 0.9985 - val_loss: 0.2977 - val_acc: 0.9055
Epoch 26/30
2750/2750 [==============================] - 0s 179us/step - loss: 0.0281 - acc: 0.9985 - val_loss: 0.2975 - val_acc: 0.9012
Epoch 27/30
2750/2750 [==============================] - 1s 189us/step - loss: 0.0256 - acc: 0.9993 - val_loss: 0.2974 - val_acc: 0.9070
Epoch 28/30
2750/2750 [==============================] - 1s 185us/step - loss: 0.0235 - acc: 0.9996 - val_loss: 0.2981 - val_acc: 0.9084
Epoch 29/30
2750/2750 [==============================] - 1s 184us/step - loss: 0.0213 - acc: 0.9996 - val_loss: 0.2979 - val_acc: 0.9041
Epoch 30/30
2750/2750 [==============================] - 0s 181us/step - loss: 0.0197 - acc: 1.0000 - val_loss: 0.2989 - val_acc: 0.9070
```

Test accuracy 可以達到 100%的準確度，validation accuracy 可以達到 90%左右的準確度。

訓練完成後，對一開始創建的 test data 做測試，結果如下 ：

```
10000/10000 [==============================] - 2s 183us/step
Test loss :   0.28968  Test accuracy :  0.91630
```

準確度可以到達 91.63 %。

- 討論 ：

(1) 與 model 1 比較 ：model 2 將模型擴大後我們會發現到，test accuracy 和 validation accuracy 下降了，但是 test accuracy 達到了 100%準確度。這代表說此模型過度擬合了。由這次經驗可以得出一個結論，模型並不是越大越好，而是要針對不同情況去調整模型大小來避免

overfitting。

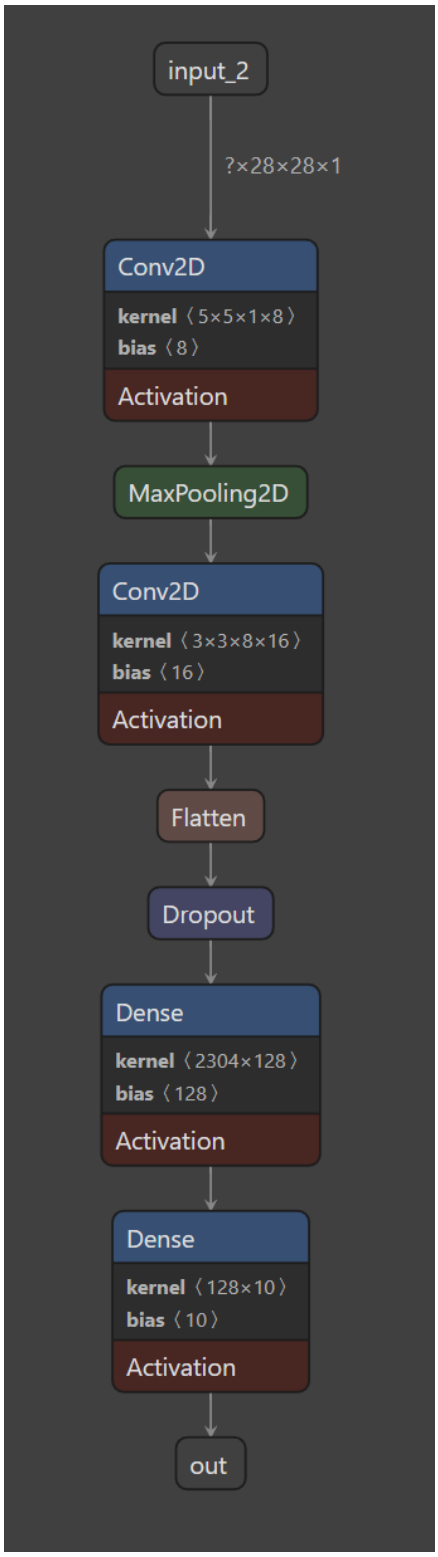
二、CNN

1. Model 1：

➢ 網路架構：

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_2 (InputLayer) | (None, 28, 28, 1) | 0 |
| conv1 (Conv2D) | (None, 28, 28, 8) | 208 |
| max_pool1_W1 (MaxPooling2D) | (None, 14, 14, 8) | 0 |
| conv2 (Conv2D) | (None, 12, 12, 16) | 1168 |
| flatten_2 (Flatten) | (None, 2304) | 0 |
| dropout_2 (Dropout) | (None, 2304) | 0 |
| fc2 (Dense) | (None, 128) | 295040 |
| out (Dense) | (None, 10) | 1290 |

Total params: 297,706
Trainable params: 297,706
Non-trainable params: 0




建立一個兩個 convolutional layer、一個 Dense layer(不包含 output 的 Dense layer，我會稱 output 的 Dense layer 為 output layer)、一個 max pooling layer 和一層 dropout 的 CNN。Hidden layer 的 activation 皆為 relu，output layer 的 activation 為

softmax。Max pooling 的 filter size 為(2, 2)，增加此層的目的在於讓 CNN 能夠更加有效的提取到關鍵特徵。此外，也透過添加 dropout 來避免 overfitting 的問題，將輸入單元被設為 0 的機率設為 0.25。

- 訓練、測試準確度 ：

關於資料的處理，我將 MNIST data set 先分成 test data 和 train data，其大小如下圖。

```
train_data.shape: (55000, 28, 28)
train_label.shape: (55000, 10)
test_data.shape: (10000, 28, 28)
test_label.shape: (10000, 10)
```

接著再將 train data 中的 1/2 設定為真正的 train data，1/5 為 validation data。除此之外，為了加速訓練將 batch size = 256，training epochs 為 50，訓練過程的準確度如下圖

```
Epoch 1/50
22000/22000 [==============================] - 8s 351us/step - loss: 0.6070 - acc: 0.8226 - val_loss: 0.2400 - val_acc: 0.9298
Epoch 2/50
22000/22000 [==============================] - 7s 320us/step - loss: 0.1896 - acc: 0.9438 - val_loss: 0.1408 - val_acc: 0.9580
Epoch 3/50
22000/22000 [==============================] - 7s 331us/step - loss: 0.1253 - acc: 0.9627 - val_loss: 0.1113 - val_acc: 0.9645
Epoch 4/50
22000/22000 [==============================] - 8s 355us/step - loss: 0.0968 - acc: 0.9701 - val_loss: 0.1014 - val_acc: 0.9687
Epoch 5/50
22000/22000 [==============================] - 8s 352us/step - loss: 0.0831 - acc: 0.9740 - val_loss: 0.0871 - val_acc: 0.9720
Epoch 6/50
22000/22000 [==============================] - 8s 343us/step - loss: 0.0689 - acc: 0.9782 - val_loss: 0.0911 - val_acc: 0.9727
Epoch 7/50
22000/22000 [==============================] - 8s 348us/step - loss: 0.0613 - acc: 0.9808 - val_loss: 0.0755 - val_acc: 0.9749
Epoch 8/50
22000/22000 [==============================] - 8s 352us/step - loss: 0.0534 - acc: 0.9826 - val_loss: 0.0790 - val_acc: 0.9745
Epoch 9/50
22000/22000 [==============================] - 8s 356us/step - loss: 0.0495 - acc: 0.9840 - val_loss: 0.0846 - val_acc: 0.9756
Epoch 10/50
22000/22000 [==============================] - 8s 359us/step - loss: 0.0431 - acc: 0.9870 - val_loss: 0.0720 - val_acc: 0.9773
Epoch 11/50
22000/22000 [==============================] - 8s 343us/step - loss: 0.0378 - acc: 0.9876 - val_loss: 0.0731 - val_acc: 0.9780
Epoch 12/50
22000/22000 [==============================] - 8s 348us/step - loss: 0.0353 - acc: 0.9881 - val_loss: 0.0731 - val_acc: 0.9789
Epoch 13/50
22000/22000 [==============================] - 8s 350us/step - loss: 0.0328 - acc: 0.9896 - val_loss: 0.0679 - val_acc: 0.9789
Epoch 14/50
22000/22000 [==============================] - 8s 345us/step - loss: 0.0278 - acc: 0.9910 - val_loss: 0.0762 - val_acc: 0.9765
Epoch 15/50
22000/22000 [==============================] - 7s 340us/step - loss: 0.0267 - acc: 0.9915 - val_loss: 0.0743 - val_acc: 0.9784
Epoch 16/50
22000/22000 [==============================] - 8s 343us/step - loss: 0.0243 - acc: 0.9920 - val_loss: 0.0809 - val_acc: 0.9769
Epoch 17/50
22000/22000 [==============================] - 7s 318us/step - loss: 0.0244 - acc: 0.9923 - val_loss: 0.0739 - val_acc: 0.9796
Epoch 18/50
22000/22000 [==============================] - 6s 260us/step - loss: 0.0209 - acc: 0.9934 - val_loss: 0.0723 - val_acc: 0.9809
Epoch 19/50
22000/22000 [==============================] - 7s 323us/step - loss: 0.0188 - acc: 0.9936 - val_loss: 0.0810 - val_acc: 0.9769
Epoch 20/50
22000/22000 [==============================] - 7s 320us/step - loss: 0.0190 - acc: 0.9937 - val_loss: 0.0705 - val_acc: 0.9807
Epoch 21/50
22000/22000 [==============================] - 7s 304us/step - loss: 0.0161 - acc: 0.9951 - val_loss: 0.0705 - val_acc: 0.9811
Epoch 22/50
22000/22000 [==============================] - 8s 360us/step - loss: 0.0142 - acc: 0.9955 - val_loss: 0.0765 - val_acc: 0.9807
Epoch 23/50
22000/22000 [==============================] - 8s 382us/step - loss: 0.0138 - acc: 0.9954 - val_loss: 0.0747 - val_acc: 0.9815
Epoch 24/50
22000/22000 [==============================] - 9s 388us/step - loss: 0.0123 - acc: 0.9964 - val_loss: 0.0787 - val_acc: 0.9807
Epoch 25/50
22000/22000 [==============================] - 9s 393us/step - loss: 0.0123 - acc: 0.9965 - val_loss: 0.0791 - val_acc: 0.9815
Epoch 26/50
22000/22000 [==============================] - 9s 400us/step - loss: 0.0123 - acc: 0.9962 - val_loss: 0.0783 - val_acc: 0.9811
Epoch 27/50
22000/22000 [==============================] - 9s 389us/step - loss: 0.0090 - acc: 0.9972 - val_loss: 0.0787 - val_acc: 0.9807
Epoch 28/50
22000/22000 [==============================] - 9s 393us/step - loss: 0.0108 - acc: 0.9965 - val_loss: 0.0841 - val_acc: 0.9805
Epoch 29/50
22000/22000 [==============================] - 8s 379us/step - loss: 0.0101 - acc: 0.9964 - val_loss: 0.0752 - val_acc: 0.9816
Epoch 30/50
22000/22000 [==============================] - 8s 385us/step - loss: 0.0090 - acc: 0.9972 - val_loss: 0.0897 - val_acc: 0.9800
```

```
Epoch 31/50
22000/22000 [==============================] - 8s 368us/step - loss: 0.0087 - acc: 0.9969 - val_loss: 0.0777 - val_acc: 0.9807
Epoch 32/50
22000/22000 [==============================] - 9s 389us/step - loss: 0.0078 - acc: 0.9974 - val_loss: 0.0805 - val_acc: 0.9811
Epoch 33/50
22000/22000 [==============================] - 8s 385us/step - loss: 0.0076 - acc: 0.9972 - val_loss: 0.0823 - val_acc: 0.9809
Epoch 34/50
22000/22000 [==============================] - 8s 367us/step - loss: 0.0076 - acc: 0.9974 - val_loss: 0.0866 - val_acc: 0.9811
Epoch 35/50
22000/22000 [==============================] - 8s 356us/step - loss: 0.0080 - acc: 0.9973 - val_loss: 0.0894 - val_acc: 0.9815
Epoch 36/50
22000/22000 [==============================] - 7s 337us/step - loss: 0.0088 - acc: 0.9969 - val_loss: 0.0812 - val_acc: 0.9813
Epoch 37/50
22000/22000 [==============================] - 8s 352us/step - loss: 0.0091 - acc: 0.9970 - val_loss: 0.0843 - val_acc: 0.9802
Epoch 38/50
22000/22000 [==============================] - 8s 360us/step - loss: 0.0085 - acc: 0.9970 - val_loss: 0.0976 - val_acc: 0.9815
Epoch 39/50
22000/22000 [==============================] - 8s 371us/step - loss: 0.0085 - acc: 0.9975 - val_loss: 0.0798 - val_acc: 0.9827
Epoch 40/50
22000/22000 [==============================] - 7s 334us/step - loss: 0.0069 - acc: 0.9975 - val_loss: 0.0830 - val_acc: 0.9822
Epoch 41/50
22000/22000 [==============================] - 7s 311us/step - loss: 0.0062 - acc: 0.9978 - val_loss: 0.0876 - val_acc: 0.9815
Epoch 42/50
22000/22000 [==============================] - 7s 317us/step - loss: 0.0059 - acc: 0.9982 - val_loss: 0.0826 - val_acc: 0.9820
Epoch 43/50
22000/22000 [==============================] - 7s 319us/step - loss: 0.0047 - acc: 0.9984 - val_loss: 0.0847 - val_acc: 0.9822
Epoch 44/50
22000/22000 [==============================] - 8s 352us/step - loss: 0.0069 - acc: 0.9975 - val_loss: 0.0957 - val_acc: 0.9813
Epoch 45/50
22000/22000 [==============================] - 7s 337us/step - loss: 0.0059 - acc: 0.9980 - val_loss: 0.0907 - val_acc: 0.9815
Epoch 46/50
22000/22000 [==============================] - 8s 353us/step - loss: 0.0045 - acc: 0.9983 - val_loss: 0.0824 - val_acc: 0.9824
Epoch 47/50
22000/22000 [==============================] - 8s 349us/step - loss: 0.0029 - acc: 0.9992 - val_loss: 0.0879 - val_acc: 0.9820
Epoch 48/50
22000/22000 [==============================] - 7s 337us/step - loss: 0.0058 - acc: 0.9981 - val_loss: 0.0955 - val_acc: 0.9805
Epoch 49/50
22000/22000 [==============================] - 7s 326us/step - loss: 0.0043 - acc: 0.9987 - val_loss: 0.0941 - val_acc: 0.9820
Epoch 50/50
22000/22000 [==============================] - 8s 345us/step - loss: 0.0031 - acc: 0.9991 - val_loss: 0.0975 - val_acc: 0.9800
```

Training accuracy 可以來到 99.9%，validation accuracy 可以達到 98.3 % 左右的準確度。

對於 testing set，測試結果如下

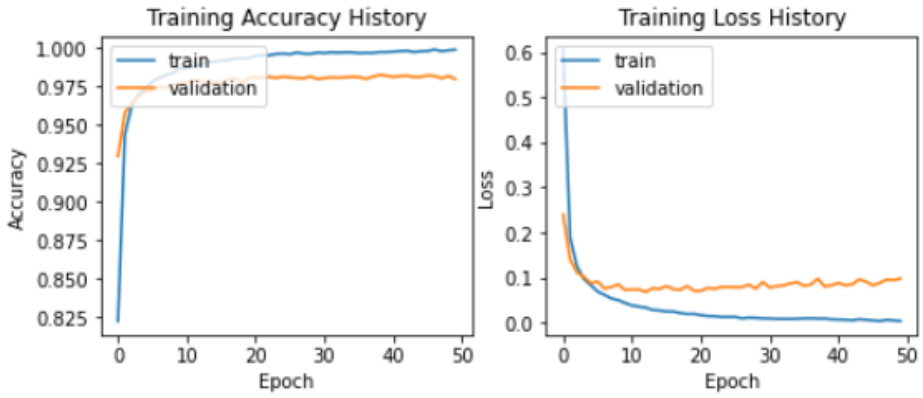
```
Test loss :   0.06075  Test accuracy :  0.98610
```

準確度可以達到 98.6%。

- 討論 ：

(1) Overfitting :

將訓練與驗證準確度視覺化，如下圖




雖然說 training accuracy 和 validation accuracy 在訓練的最後都趨近飽和，可能會發生 overfitting 的問題，但是從訓練結果來看，testing accuracy 與 training accuracy 只差了 1.3%，因此我認為 overfitting 的情況並不是很嚴重，為了驗證我的猜測，將 training epochs 降為 15 再做一次訓

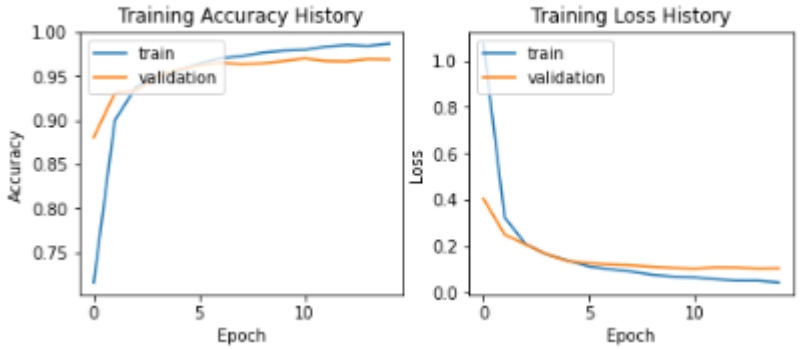
練，結果如下

```
Epoch 1/15
11000/11000 [==============================] - 5s 448us/step - loss: 1.0735 - acc: 0.7155 - val_loss: 0.4033 - val_acc: 0.8804
Epoch 2/15
11000/11000 [==============================] - 4s 371us/step - loss: 0.3233 - acc: 0.8995 - val_loss: 0.2495 - val_acc: 0.9305
Epoch 3/15
11000/11000 [==============================] - 4s 336us/step - loss: 0.2076 - acc: 0.9374 - val_loss: 0.2064 - val_acc: 0.9335
Epoch 4/15
11000/11000 [==============================] - 4s 345us/step - loss: 0.1632 - acc: 0.9508 - val_loss: 0.1637 - val_acc: 0.9498
Epoch 5/15
11000/11000 [==============================] - 4s 336us/step - loss: 0.1369 - acc: 0.9555 - val_loss: 0.1346 - val_acc: 0.9571
Epoch 6/15
11000/11000 [==============================] - 4s 355us/step - loss: 0.1115 - acc: 0.9643 - val_loss: 0.1251 - val_acc: 0.9618
Epoch 7/15
11000/11000 [==============================] - 4s 358us/step - loss: 0.0996 - acc: 0.9699 - val_loss: 0.1199 - val_acc: 0.9651
Epoch 8/15
11000/11000 [==============================] - 4s 339us/step - loss: 0.0894 - acc: 0.9725 - val_loss: 0.1163 - val_acc: 0.9636
Epoch 9/15
11000/11000 [==============================] - 4s 351us/step - loss: 0.0757 - acc: 0.9764 - val_loss: 0.1101 - val_acc: 0.9644
Epoch 10/15
11000/11000 [==============================] - 4s 352us/step - loss: 0.0679 - acc: 0.9786 - val_loss: 0.1056 - val_acc: 0.9669
Epoch 11/15
11000/11000 [==============================] - 4s 354us/step - loss: 0.0643 - acc: 0.9796 - val_loss: 0.1026 - val_acc: 0.9698
Epoch 12/15
11000/11000 [==============================] - 3s 315us/step - loss: 0.0563 - acc: 0.9834 - val_loss: 0.1076 - val_acc: 0.9669
Epoch 13/15
11000/11000 [==============================] - 3s 292us/step - loss: 0.0502 - acc: 0.9852 - val_loss: 0.1067 - val_acc: 0.9665
Epoch 14/15
11000/11000 [==============================] - 4s 354us/step - loss: 0.0500 - acc: 0.9841 - val_loss: 0.1031 - val_acc: 0.9691
Epoch 15/15
11000/11000 [==============================] - 4s 385us/step - loss: 0.0422 - acc: 0.9866 - val_loss: 0.1042 - val_acc: 0.9687
```

Training accuracy 為 98.66%，validation accuracy 為 96.87%。

對 testing set 做測試，結果如下

```
Test loss :  0.07275  Test accuracy :  0.97620
```




比起 epochs=50，test accuracy 下降了一些，也驗證了我的說法。

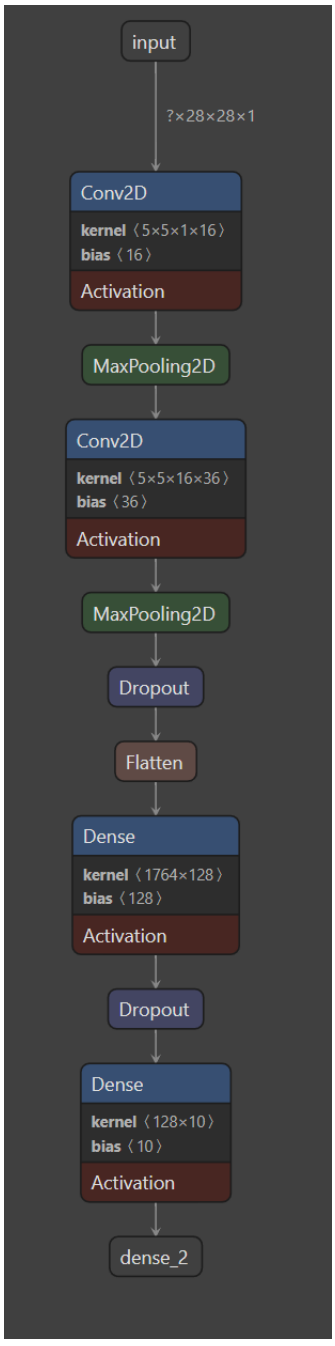
2. Model 2 :

➢ 網路架構 :

這個模型主要是增加一層 max pooling layer 來幫助模型更有效的抓取到重要特徵，並且增加一層 drop out layer (drop out 機率也提升到 0.5)來讓模型有更好的泛化能力。整體架構如下圖

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_1 (Conv2D) | (None, 28, 28, 16) | 416 |
| max_pooling2d_1 (MaxPooling2 | (None, 14, 14, 16) | 0 |
| conv2d_2 (Conv2D) | (None, 14, 14, 36) | 14436 |
| max_pooling2d_2 (MaxPooling2 | (None, 7, 7, 36) | 0 |
| dropout_1 (Dropout) | (None, 7, 7, 36) | 0 |
| flatten_1 (Flatten) | (None, 1764) | 0 |
| dense_1 (Dense) | (None, 128) | 225920 |
| dropout_2 (Dropout) | (None, 128) | 0 |
| dense_2 (Dense) | (None, 10) | 1290 |

Total params: 242,062
Trainable params: 242,062
Non-trainable params: 0



- 訓練、測試準確度 ：

  這一次訓練沒有使用 batch，而是一次將所有 training data 輸入進去做訓練。會這麼做的原因在於對於 model 2 來說，使用 batch 來做訓練會讓結果變差，因此我就不使用 batch 來做訓練。由於這次沒有使用 batch，因此可以將 training epochs 降低，這是因為每一次的訓練都考慮所有樣本，對於準確度的提昇會有很大的幫助，因此不需要訓練那麼多次就可以達到很好的效果，但是每一次迭代的時間也會提升(整體所花費的時間與前一個 model 所花費的時間差不多)。訓練過程如下

```
Epoch 1/10
48000/48000 [==============================] - 30s 620us/step - loss: 0.4897 -
acc: 0.8475 - val_loss: 0.0966 - val_acc: 0.9721
Epoch 2/10
48000/48000 [==============================] - 42s 869us/step - loss: 0.1417 -
acc: 0.9580 - val_loss: 0.0625 - val_acc: 0.9806
Epoch 3/10
48000/48000 [==============================] - 40s 829us/step - loss: 0.1028 -
acc: 0.9692 - val_loss: 0.0520 - val_acc: 0.9839
Epoch 4/10
48000/48000 [==============================] - 40s 842us/step - loss: 0.0833 -
acc: 0.9755 - val_loss: 0.0446 - val_acc: 0.9861
Epoch 5/10
48000/48000 [==============================] - 40s 841us/step - loss: 0.0718 -
acc: 0.9780 - val_loss: 0.0399 - val_acc: 0.9875
Epoch 6/10
48000/48000 [==============================] - 41s 857us/step - loss: 0.0634 -
acc: 0.9811 - val_loss: 0.0392 - val_acc: 0.9884
Epoch 7/10
48000/48000 [==============================] - 42s 880us/step - loss: 0.0563 -
acc: 0.9830 - val_loss: 0.0421 - val_acc: 0.9878
Epoch 8/10
48000/48000 [==============================] - 42s 882us/step - loss: 0.0508 -
acc: 0.9844 - val_loss: 0.0340 - val_acc: 0.9899
Epoch 9/10
48000/48000 [==============================] - 43s 893us/step - loss: 0.0446 -
acc: 0.9868 - val_loss: 0.0342 - val_acc: 0.9898
Epoch 10/10
48000/48000 [==============================] - 43s 887us/step - loss: 0.0422 -
acc: 0.9870 - val_loss: 0.0336 - val_acc: 0.9902
```

  Training accuracy 可以達到 98.7 %，validation accuracy 可達到 99%。

  利用 testing set 做測試，結果如下

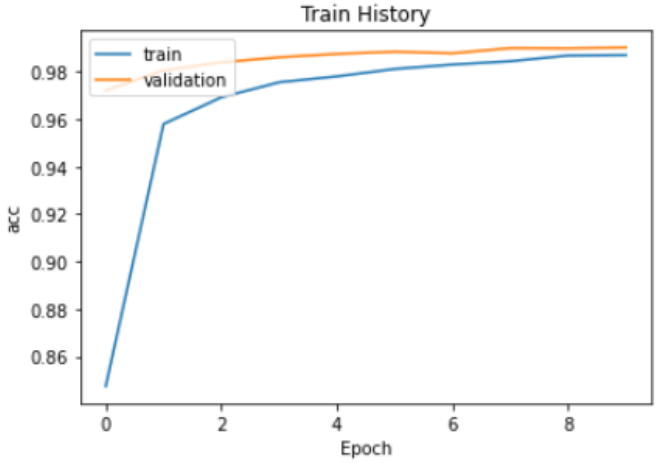
```
10000/10000 [==============================] - 3s 277us/step

        [Info] Accuracy of testing data = 99.1%
```

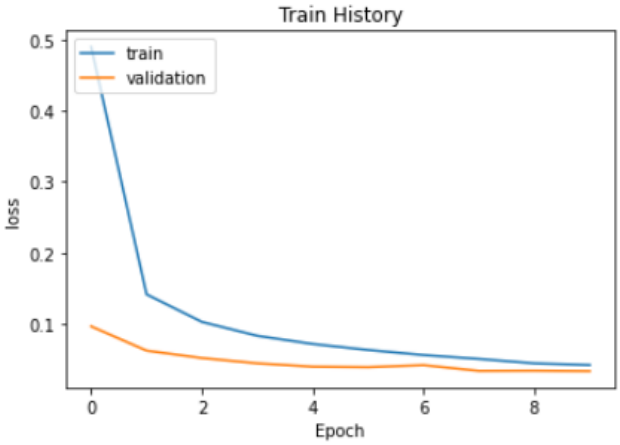
  對於測試資料，此模型有 99.1%的準確度。

- 討論 ：

  (1) Model 1 v.s. model 2： 與前一個 CNN 架構比較，新的 CNN 架構沒有 overfitting 的問題，這主要是因為添加一層 drop out layer。將訓練過程的準確度與 loss 視覺化，結果

如下




由上面兩張圖與前一個模型的結果比較可以看出，這一次的訓練到達尾聲時，不論是 validation accuracy 或是 validation loss 都有持續在做優化，這便是沒有發生 overfitting 的證據。此外，比較 training accuracy 與 testing accuracy 之後甚至可以發現 testing accuracy 比 training accuracy 來的高，表示這個模型有很好的泛化能力。

(2) NN v.s CNN :

不論是第一個 CNN 模型或是第二個 CNN 模型，其結果都比 NN 來的更好，主要原因如下 :

- CNN 使用的卷積層會讓 CNN 更容易去捕捉圖象空間中的關係，此外，CNN 中的 max pooling layer 也會使 CNN 更容易去抓取重要的特徵。
- CNN 比 NN 更不容易 overfit。這主要是因為 drop out layer 的緣故。對於 CNN 來說，只要有效的利用 drop

out layer 來降低 overfitting 發生的機率，我們就可以建構更大的神經網路來抓取一些更複雜的特徵，這對於提升 CNN 的極限有很大的幫助。但是對於一般 NN 來說，太大的模型很容易造成 overfit，這便限制了 NN 的極限表現。

(3) CNN 的極限在哪：

對於這個問題，我認為只要有效地透過 drop out layer 來解決 overfit 的問題，CNN 理論上是可以做到 universal approximation 的。但是在實作上，我們是否有足夠強大的硬體設備來訓練超級巨大的 CNN，以及我們使否有足夠強大的硬體條件來讓我們實際應用超級巨大的 CNN，都會是現實應用中要考慮的問題，所以 universal approximation 只是理論上的極限，實際應用上的極限會受到硬體設備的限制。

# II. 高鐵辨識碼

## 一、Model 1：

1. 資料預處理：

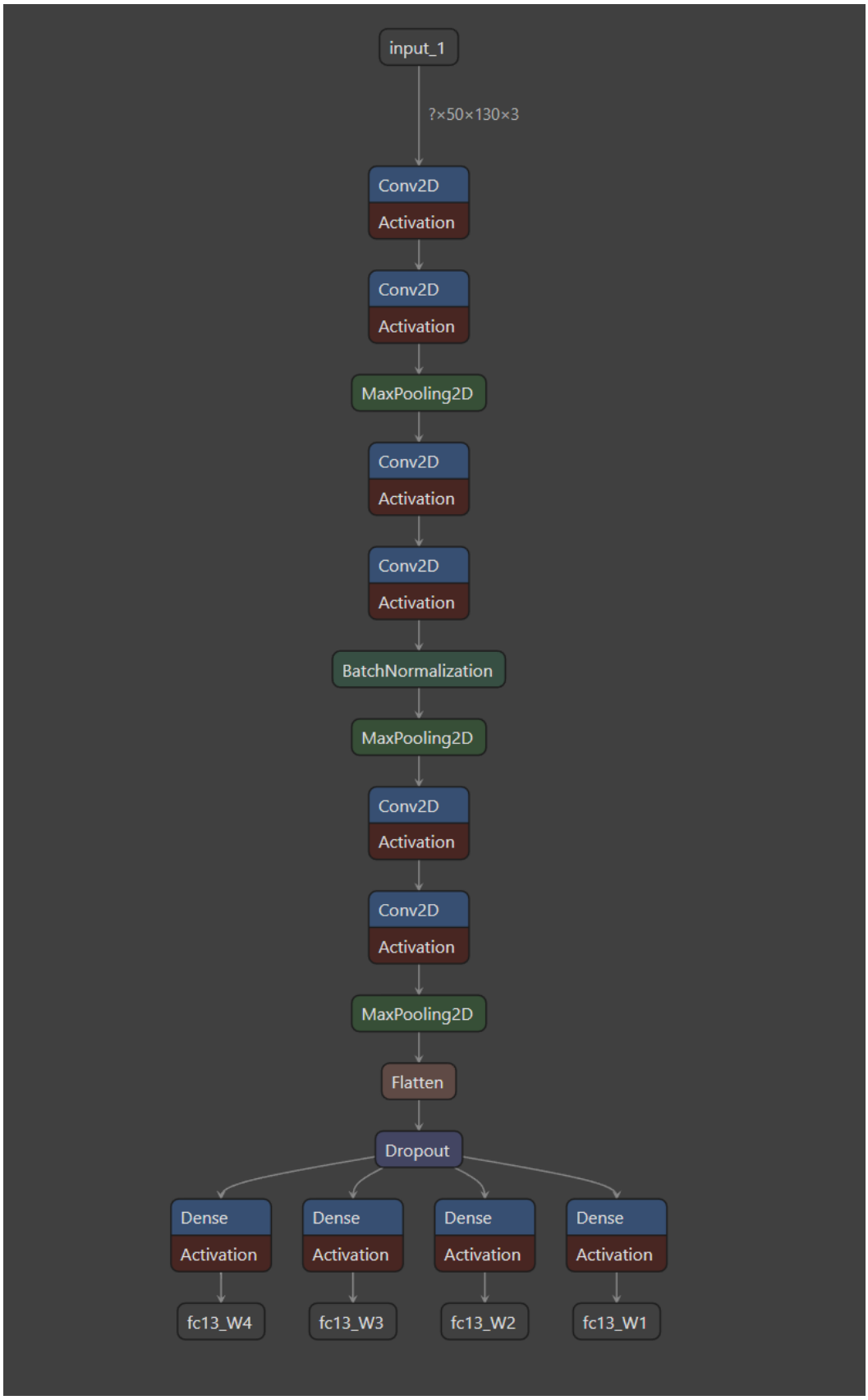
   使用 nopre 作為預處理方式，也就是僅將影像 resize 在正規化。

2. 網路架構：

   此模型的總參數為 866412，使用了 6 層 convolutional layer、3 層 maxpooling layer、1 層 batch normalization layer、1 層 dropout layer、4 層 dense layer 作為 output layer。

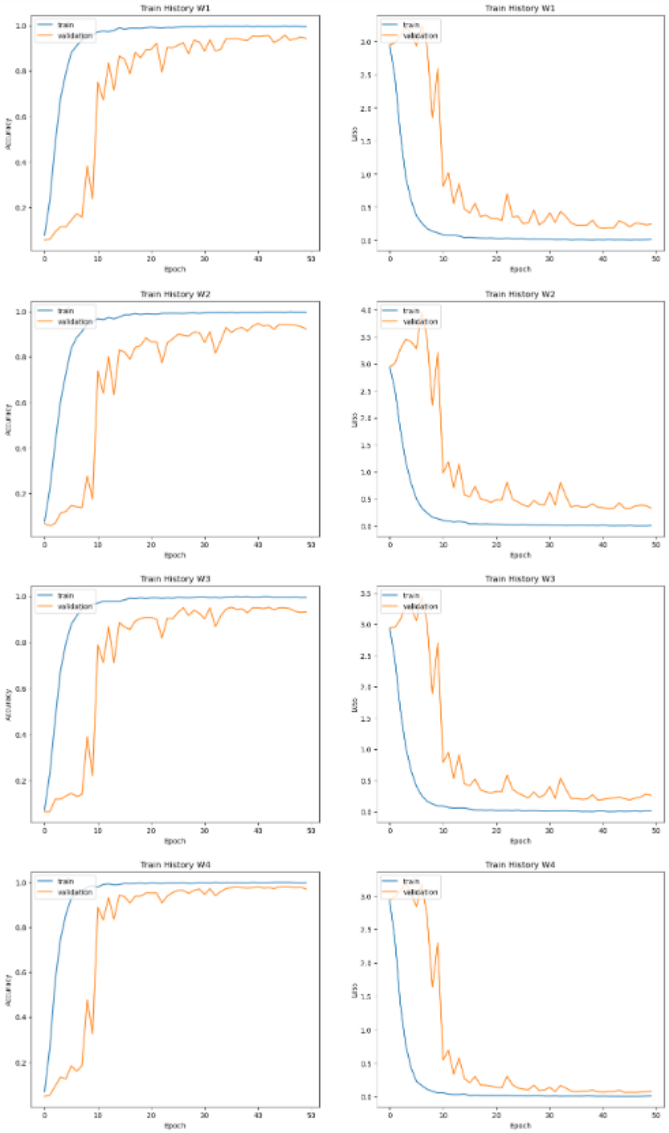
| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | [(None, 50, 130, 3)] | 0 | [] |
| conv11_W1 (Conv2D) | (None, 50, 130, 32) | 2432 | ['input_1[0][0]'] |
| conv12_W1 (Conv2D) | (None, 48, 128, 32) | 9248 | ['conv11_W1[0][0]'] |
| max_pool1_W1 (MaxPooling2D) | (None, 24, 64, 32) | 0 | ['conv12_W1[0][0]'] |
| conv23_W1 (Conv2D) | (None, 24, 64, 64) | 51264 | ['max_pool1_W1[0][0]'] |
| conv24_W1 (Conv2D) | (None, 22, 62, 64) | 36928 | ['conv23_W1[0][0]'] |
| bn1_W1 (BatchNormalization) | (None, 22, 62, 64) | 256 | ['conv24_W1[0][0]'] |
| max_pool2_W1 (MaxPooling2D) | (None, 11, 31, 64) | 0 | ['bn1_W1[0][0]'] |
| conv36_W1 (Conv2D) | (None, 11, 31, 128) | 73856 | ['max_pool2_W1[0][0]'] |
| conv37_W1 (Conv2D) | (None, 9, 29, 128) | 147584 | ['conv36_W1[0][0]'] |
| max_pool3_W1 (MaxPooling2D) | (None, 4, 14, 128) | 0 | ['conv37_W1[0][0]'] |
| flatten (Flatten) | (None, 7168) | 0 | ['max_pool3_W1[0][0]'] |
| dropout (Dropout) | (None, 7168) | 0 | ['flatten[0][0]'] |
| fc13_W1 (Dense) | (None, 19) | 136211 | ['dropout[0][0]'] |
| fc13_W2 (Dense) | (None, 19) | 136211 | ['dropout[0][0]'] |
| fc13_W3 (Dense) | (None, 19) | 136211 | ['dropout[0][0]'] |
| fc13_W4 (Dense) | (None, 19) | 136211 | ['dropout[0][0]'] |

Total params: 866412 (3.31 MB)
Trainable params: 866284 (3.30 MB)
Non-trainable params: 128 (512.00 Byte)


input_1
?×50×130×3
Conv2D
Activation
Conv2D
Activation
MaxPooling2D
Conv2D
Activation
Conv2D
Activation
BatchNormalization
MaxPooling2D
Conv2D
Activation
Conv2D
Activation
MaxPooling2D
Flatten
Dropout
Dense
Activation
Dense
Activation
Dense
Activation
Dense
Activation
fc13_W4
fc13_W3
fc13_W2
fc13_W1

3. 訓練過程 ：

Loss function 選用 categorical cross entropy、optimizer 選用 adam、validation split = 0.2、epochs=50、batch size = 128、learning rate = 0.5。此外，還有使用到兩個 callbacks，分別為 EarlyStopping 和 ReduceLROnPlateau。EarlyStopping 會使訓練在 validation loss 連續不降低時停止訓練，ReduceLROnPlateau 會使 learning rate 在 validation loss 連續不降低時變小。加上這兩個 callbacks 的主要目的為盡量避免 overfitting 的出現。將四個數字的 accuracy 和 loss 視覺化，結果如下圖




可以看出，雖然在訓練過程中 accuracy 和 loss 皆有上下起伏的時候，但是由於 callbacks 的幫助，整體的趨勢依然是往好的方向發

展。

4. 訓練、測試準確度 ：

```
Epoch 47: val_loss did not improve from 0.78517
16/16 [==============================] - 25s 2s/step - loss: 0.0279 - fc13_W1_loss: 0.0079 - fc13_W2_loss: 0.0046 - fc13_W3_los
s: 0.0124 - fc13_W4_loss: 0.0030 - fc13_W1_accuracy: 0.9970 - fc13_W2_accuracy: 0.9990 - fc13_W3_accuracy: 0.9960 - fc13_W4_acc
uracy: 0.9995 - val_loss: 0.9139 - val_fc13_W1_loss: 0.2561 - val_fc13_W2_loss: 0.3741 - val_fc13_W3_loss: 0.2208 - val_fc13_W4
_loss: 0.0629 - val_fc13_W1_accuracy: 0.9360 - val_fc13_W2_accuracy: 0.9420 - val_fc13_W3_accuracy: 0.9460 - val_fc13_W4_accura
cy: 0.9800 - lr: 5.1200e-04
Epoch 48/50

16/16 [==============================] - ETA: 0s - loss: 0.0220 - fc13_W1_loss: 0.0068 - fc13_W2_loss: 0.0052 - fc13_W3_loss:
0.0066 - fc13_W4_loss: 0.0034 - fc13_W1_accuracy: 0.9985 - fc13_W2_accuracy: 0.9975 - fc13_W3_accuracy: 0.9975 - fc13_W4_accura
cy: 0.9985
Epoch 48: val_loss did not improve from 0.78517
16/16 [==============================] - 25s 2s/step - loss: 0.0220 - fc13_W1_loss: 0.0068 - fc13_W2_loss: 0.0052 - fc13_W3_los
s: 0.0066 - fc13_W4_loss: 0.0034 - fc13_W1_accuracy: 0.9985 - fc13_W2_accuracy: 0.9975 - fc13_W3_accuracy: 0.9975 - fc13_W4_acc
uracy: 0.9985 - val_loss: 0.9435 - val_fc13_W1_loss: 0.2516 - val_fc13_W2_loss: 0.3869 - val_fc13_W3_loss: 0.2379 - val_fc13_W4
_loss: 0.0670 - val_fc13_W1_accuracy: 0.9420 - val_fc13_W2_accuracy: 0.9400 - val_fc13_W3_accuracy: 0.9340 - val_fc13_W4_accura
cy: 0.9760 - lr: 5.1200e-04
Epoch 49/50
16/16 [==============================] - ETA: 0s - loss: 0.0336 - fc13_W1_loss: 0.0084 - fc13_W2_loss: 0.0050 - fc13_W3_loss:
0.0142 - fc13_W4_loss: 0.0059 - fc13_W1_accuracy: 0.9965 - fc13_W2_accuracy: 0.9980 - fc13_W3_accuracy: 0.9950 - fc13_W4_accura
cy: 0.9975
Epoch 49: val_loss did not improve from 0.78517
16/16 [==============================] - 24s 1s/step - loss: 0.0336 - fc13_W1_loss: 0.0084 - fc13_W2_loss: 0.0050 - fc13_W3_los
s: 0.0142 - fc13_W4_loss: 0.0059 - fc13_W1_accuracy: 0.9965 - fc13_W2_accuracy: 0.9980 - fc13_W3_accuracy: 0.9950 - fc13_W4_acc
uracy: 0.9975 - val_loss: 0.9671 - val_fc13_W1_loss: 0.2336 - val_fc13_W2_loss: 0.3798 - val_fc13_W3_loss: 0.2843 - val_fc13_W4
_loss: 0.0695 - val_fc13_W1_accuracy: 0.9500 - val_fc13_W2_accuracy: 0.9340 - val_fc13_W3_accuracy: 0.9300 - val_fc13_W4_accura
cy: 0.9780 - lr: 5.1200e-04
Epoch 50/50
16/16 [==============================] - ETA: 0s - loss: 0.0427 - fc13_W1_loss: 0.0111 - fc13_W2_loss: 0.0097 - fc13_W3_loss:
0.0148 - fc13_W4_loss: 0.0072 - fc13_W1_accuracy: 0.9955 - fc13_W2_accuracy: 0.9965 - fc13_W3_accuracy: 0.9955 - fc13_W4_accura
cy: 0.9980
Epoch 50: val_loss did not improve from 0.78517
16/16 [==============================] - 26s 2s/step - loss: 0.0427 - fc13_W1_loss: 0.0111 - fc13_W2_loss: 0.0097 - fc13_W3_los
s: 0.0148 - fc13_W4_loss: 0.0072 - fc13_W1_accuracy: 0.9955 - fc13_W2_accuracy: 0.9965 - fc13_W3_accuracy: 0.9955 - fc13_W4_acc
uracy: 0.9980 - val_loss: 0.9184 - val_fc13_W1_loss: 0.2419 - val_fc13_W2_loss: 0.3324 - val_fc13_W3_loss: 0.2634 - val_fc13_W4
_loss: 0.0807 - val_fc13_W1_accuracy: 0.9440 - val_fc13_W2_accuracy: 0.9240 - val_fc13_W3_accuracy: 0.9320 - val_fc13_W4_accura
cy: 0.9700 - lr: 5.1200e-04
```

由上圖可以看出，在訓練尾聲時，training accuracy 來到 99%左右，validation accuracy 來到 94%左右。

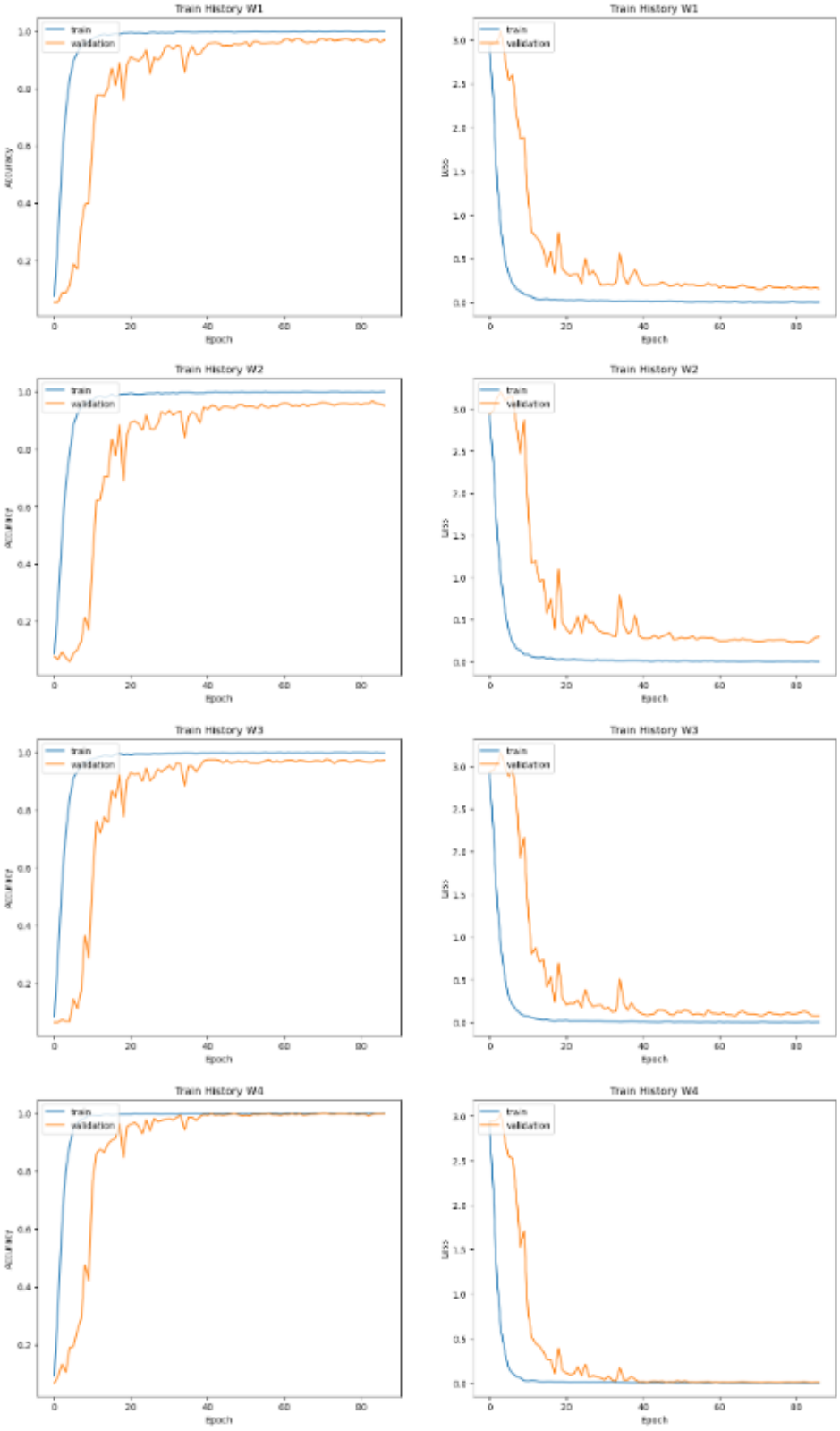
接著用 3000 筆測試資料來看 testing accuracy，結果如下圖，

```
3000 picture total wrong = 288
image accuracy = 0.904
word accuracy = 0.936
average per execute time: 162.885307 ms
total execute time = 512.031889 s
```

Testing accuracy 來到 90.4%，測試 3000 筆資料所需時間為 512 s。

5. 討論 ：

- 訓練時間 ： 由於用上 GPU 加速的緣故(原先助教們給的 code 有版本問題，因此我仿照助教的 code 稍微改寫一下即可用 GPU 加速運算)，因此 epochs=50 的總共訓練時間為 1250 s 左右。
- Epochs : 由於 callbacks 的幫助，訓練 50 次的情況下沒有 overfitting 的現象，因此我嘗試加大 epochs 數(epochs=100)，看看準確度是否能夠繼續提升。訓練過程如下(視覺化後)，




可以看出，比起 epochs=50，epochs=100 能夠繼續優化模型，讓 validation accuracy 提升到 96 %左右。在訓練過程中，訓練道第 87 次的時候，訓練過程因為 callbacks 的關係停止訓練，

代表說此模型已經到達極限，callbacks 的作用如下圖。

```
Epoch 87: val_loss did not improve from 0.46865

Epoch 87: ReduceLROnPlateau reducing learning rate to 0.00013421773910522462.
16/16 [==============================] - 26s 2s/step - loss: 0.0052 - fc13_W1_l
oss: 0.0019 - fc13_W2_loss: 8.7930e-04 - fc13_W3_loss: 0.0019 - fc13_W4_loss:
4.7529e-04 - fc13_W1_accuracy: 0.9995 - fc13_W2_accuracy: 1.0000 - fc13_W3_accu
racy: 0.9990 - fc13_W4_accuracy: 1.0000 - val_loss: 0.5388 - val_fc13_W1_loss:
0.1584 - val_fc13_W2_loss: 0.2940 - val_fc13_W3_loss: 0.0787 - val_fc13_W4_los
s: 0.0078 - val_fc13_W1_accuracy: 0.9680 - val_fc13_W2_accuracy: 0.9520 - val_f
c13_W3_accuracy: 0.9720 - val_fc13_W4_accuracy: 0.9980 - lr: 1.6777e-04
Epoch 87: early stopping
```

接著看測試結果，結果如下圖

```
3000 picture total wrong = 164
image accuracy = 0.94533
word accuracy = 0.96575
average per execute time: 175.419968 ms
total execute time = 535.173676 s
```

可以看到 testing accuracy 來到 94.53%，預測時間為 535.17 s，跟 epochs=50 的模型比起來，確實有在進步。但是代價就是訓練時間提升到 2175 s 左右。

- 結論：有了 callbacks 的幫助，我們可以將 epochs 設定為比較大的數值，讓模型一直做訓練，並利用 callbacks 的幫助來避免 overfit 的問題。這樣一來，我們可以更好的去找到模型的最佳表現。不過這麼做就需要犧牲掉訓練時長，但是我認為這無傷大雅，因為我們更應該去在意預測所花費的時間，而不是訓練所花費的時間。

二、Model 2：

1. 資料預處理：

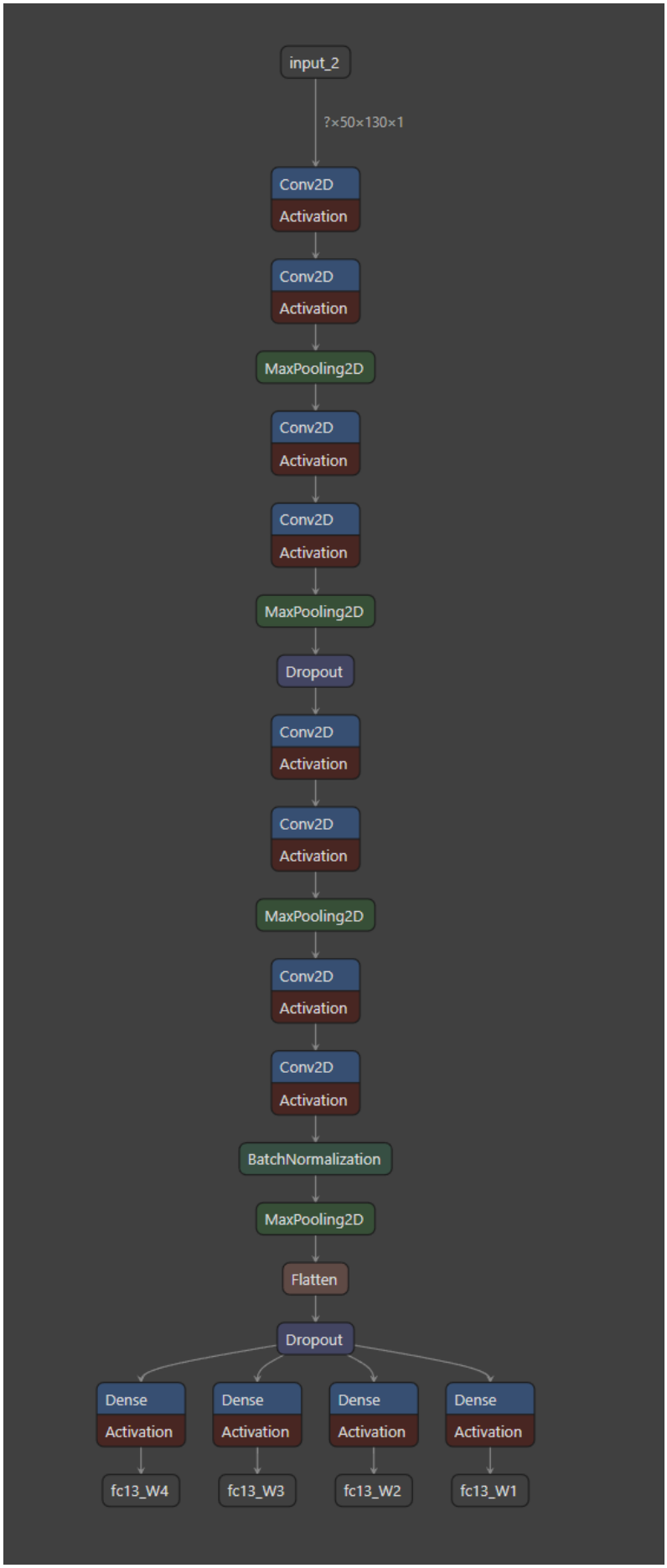
   這次改為使用 dn2 的資料預處理方式，也就是先去除雜訊，然後 resize，最後在正規化。

2. 網路架構：

   Model 2 比起 model 1 擴大一些。這個模型有 9 個 convolutional layer、4 個 max pooling layer、兩個 drop out 機率為 0.5 的 drop out layer、1 個 batch normalization layer、4 層 dense layer 作為 output layer。

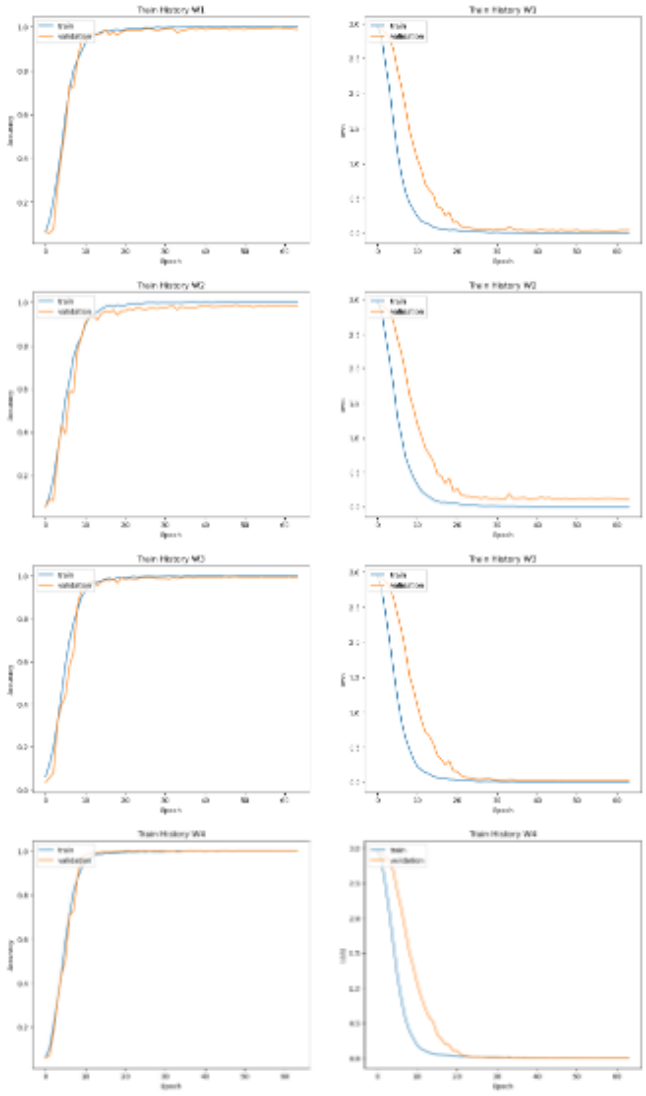
| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_2 (InputLayer) | [(None, 50, 130, 1)] | 0 | [] |
| conv11_W1 (Conv2D) | (None, 50, 130, 32) | 832 | ['input_2[0][0]'] |
| conv12_W1 (Conv2D) | (None, 48, 128, 32) | 9248 | ['conv11_W1[0][0]'] |
| max_pool1_W1 (MaxPooling2D) | (None, 24, 64, 32) | 0 | ['conv12_W1[0][0]'] |
| conv23_W1 (Conv2D) | (None, 24, 64, 64) | 51264 | ['max_pool1_W1[0][0]'] |
| conv24_W1 (Conv2D) | (None, 22, 62, 64) | 36928 | ['conv23_W1[0][0]'] |
| max_pool2_W1 (MaxPooling2D) | (None, 11, 31, 64) | 0 | ['conv24_W1[0][0]'] |
| dropout_2 (Dropout) | (None, 11, 31, 64) | 0 | ['max_pool2_W1[0][0]'] |
| conv36_W1 (Conv2D) | (None, 11, 31, 128) | 73856 | ['dropout_2[0][0]'] |
| conv37_W1 (Conv2D) | (None, 9, 29, 128) | 147584 | ['conv36_W1[0][0]'] |
| max_pool3_W1 (MaxPooling2D) | (None, 4, 14, 128) | 0 | ['conv37_W1[0][0]'] |
| conv49_W1 (Conv2D) | (None, 4, 14, 128) | 147584 | ['max_pool3_W1[0][0]'] |
| conv410_W1 (Conv2D) | (None, 2, 12, 256) | 295168 | ['conv49_W1[0][0]'] |
| bn2_W1 (BatchNormalization) | (None, 2, 12, 256) | 1024 | ['conv410_W1[0][0]'] |
| max_pool4_W1 (MaxPooling2D) | (None, 1, 6, 256) | 0 | ['bn2_W1[0][0]'] |
| flatten_1 (Flatten) | (None, 1536) | 0 | ['max_pool4_W1[0][0]'] |
| dropout_3 (Dropout) | (None, 1536) | 0 | ['flatten_1[0][0]'] |
| fc13_W1 (Dense) | (None, 19) | 29203 | ['dropout_3[0][0]'] |
| fc13_W2 (Dense) | (None, 19) | 29203 | ['dropout_3[0][0]'] |
| fc13_W3 (Dense) | (None, 19) | 29203 | ['dropout_3[0][0]'] |
| fc13_W4 (Dense) | (None, 19) | 29203 | ['dropout_3[0][0]'] |

Total params: 880300 (3.36 MB)
Trainable params: 879788 (3.36 MB)
Non-trainable params: 512 (2.00 KB)


input_2
?×50×130×1
Conv2D
Activation
Conv2D
Activation
MaxPooling2D
Conv2D
Activation
Conv2D
Activation
MaxPooling2D
Dropout
Conv2D
Activation
Conv2D
Activation
MaxPooling2D
Conv2D
Activation
Conv2D
Activation
BatchNormalization
MaxPooling2D
Flatten
Dropout
Dense
Activation
Dense
Activation
Dense
Activation
Dense
Activation
fc13_W4
fc13_W3
fc13_W2
fc13_W1

3. 訓練過程 ：

Loss function 選用 categorical cross entropy、optimizer 選用 adam、validation split = 0.2、epochs=100、batch size = 128、learning rate = 0.5。此外，還有使用到兩個 callbacks，分別為 EarlyStopping 和 ReduceLROnPlateau。EarlyStopping 會使訓練在 validation loss 連續不降低時停止訓練，ReduceLROnPlateau 會使 learning rate 在 validation loss 連續不降低時變小。加上這兩個 callbacks 的主要目的為盡量避免 overfitting 的出現。將四個數字的 accuracy 和 loss 視覺化，結果如下圖



由於 callbacks 的關係，訓練在 epochs=64 時停止，如下圖

```
Epoch 64: ReduceLROnPlateau reducing learning rate to 0.00032768002711236477.
16/16 [==============================] - 24s 2s/step - loss: 0.0086 - fc13_W1_loss: 0.0028 - fc13_W
2_loss: 0.0020 - fc13_W3_loss: 0.0019 - fc13_W4_loss: 0.0019 - fc13_W1_accuracy: 0.9995 - fc13_W2_a
ccuracy: 0.9995 - fc13_W3_accuracy: 1.0000 - fc13_W4_accuracy: 1.0000 - val_loss: 0.1860 - val_fc13
_W1_loss: 0.0449 - val_fc13_W2_loss: 0.1181 - val_fc13_W3_loss: 0.0226 - val_fc13_W4_loss: 3.7681e-
04 - val_fc13_W1_accuracy: 0.9880 - val_fc13_W2_accuracy: 0.9840 - val_fc13_W3_accuracy: 0.9920 - v
al_fc13_W4_accuracy: 1.0000 - lr: 4.0960e-04
Epoch 64: early stopping
```

4. 訓練、測試準確度 ：

```
Epoch 61/100
16/16 [==============================] - ETA: 0s - loss: 0.0095 - fc13_W1_loss: 0.0017 - fc13_W2_loss: 0.0029 - fc13_W3_loss:
0.0026 - fc13_W4_loss: 0.0022 - fc13_W1_accuracy: 1.0000 - fc13_W2_accuracy: 0.9995 - fc13_W3_accuracy: 1.0000 - fc13_W4_accura
cy: 1.0000
Epoch 61: val_loss did not improve from 0.17642
16/16 [==============================] - 24s 2s/step - loss: 0.0095 - fc13_W1_loss: 0.0017 - fc13_W2_loss: 0.0029 - fc13_W3_los
s: 0.0026 - fc13_W4_loss: 0.0022 - fc13_W1_accuracy: 1.0000 - fc13_W2_accuracy: 0.9995 - fc13_W3_accuracy: 1.0000 - fc13_W4_acc
uracy: 1.0000 - val_loss: 0.1813 - val_fc13_W1_loss: 0.0368 - val_fc13_W2_loss: 0.1202 - val_fc13_W3_loss: 0.0239 - val_fc13_W4
_loss: 3.6630e-04 - val_fc13_W1_accuracy: 0.9920 - val_fc13_W2_accuracy: 0.9860 - val_fc13_W3_accuracy: 0.9920 - val_fc13_W4_ac
curacy: 1.0000 - lr: 4.0960e-04
Epoch 62/100
16/16 [==============================] - ETA: 0s - loss: 0.0092 - fc13_W1_loss: 0.0025 - fc13_W2_loss: 0.0029 - fc13_W3_loss:
0.0018 - fc13_W4_loss: 0.0020 - fc13_W1_accuracy: 0.9995 - fc13_W2_accuracy: 0.9995 - fc13_W3_accuracy: 1.0000 - fc13_W4_accura
cy: 1.0000
Epoch 62: val_loss did not improve from 0.17642
16/16 [==============================] - 24s 2s/step - loss: 0.0092 - fc13_W1_loss: 0.0025 - fc13_W2_loss: 0.0029 - fc13_W3_los
s: 0.0018 - fc13_W4_loss: 0.0020 - fc13_W1_accuracy: 0.9995 - fc13_W2_accuracy: 0.9995 - fc13_W3_accuracy: 1.0000 - fc13_W4_acc
uracy: 1.0000 - val_loss: 0.1884 - val_fc13_W1_loss: 0.0446 - val_fc13_W2_loss: 0.1180 - val_fc13_W3_loss: 0.0244 - val_fc13_W4
_loss: 0.0013 - val_fc13_W1_accuracy: 0.9920 - val_fc13_W2_accuracy: 0.9860 - val_fc13_W3_accuracy: 0.9960 - val_fc13_W4_accura
cy: 1.0000 - lr: 4.0960e-04
Epoch 63/100
16/16 [==============================] - ETA: 0s - loss: 0.0107 - fc13_W1_loss: 0.0022 - fc13_W2_loss: 0.0034 - fc13_W3_loss:
0.0023 - fc13_W4_loss: 0.0028 - fc13_W1_accuracy: 1.0000 - fc13_W2_accuracy: 1.0000 - fc13_W3_accuracy: 0.9990 - fc13_W4_accura
cy: 0.9995
Epoch 63: val_loss did not improve from 0.17642
16/16 [==============================] - 23s 1s/step - loss: 0.0107 - fc13_W1_loss: 0.0022 - fc13_W2_loss: 0.0034 - fc13_W3_los
s: 0.0023 - fc13_W4_loss: 0.0028 - fc13_W1_accuracy: 1.0000 - fc13_W2_accuracy: 1.0000 - fc13_W3_accuracy: 0.9990 - fc13_W4_acc
uracy: 0.9995 - val_loss: 0.1865 - val_fc13_W1_loss: 0.0442 - val_fc13_W2_loss: 0.1142 - val_fc13_W3_loss: 0.0270 - val_fc13_W4
_loss: 0.0010 - val_fc13_W1_accuracy: 0.9900 - val_fc13_W2_accuracy: 0.9820 - val_fc13_W3_accuracy: 0.9920 - val_fc13_W4_accura
cy: 1.0000 - lr: 4.0960e-04
Epoch 64/100
16/16 [==============================] - ETA: 0s - loss: 0.0086 - fc13_W1_loss: 0.0028 - fc13_W2_loss: 0.0020 - fc13_W3_loss:
0.0019 - fc13_W4_loss: 0.0019 - fc13_W1_accuracy: 0.9995 - fc13_W2_accuracy: 0.9995 - fc13_W3_accuracy: 1.0000 - fc13_W4_accura
cy: 1.0000Restoring model weights from the end of the best epoch: 49.

Epoch 64: val_loss did not improve from 0.17642


Epoch 64: ReduceLROnPlateau reducing learning rate to 0.00032768002711236477.
16/16 [==============================] - 24s 2s/step - loss: 0.0086 - fc13_W1_loss: 0.0028 - fc13_W2_loss: 0.0020 - fc13_W3_los
s: 0.0019 - fc13_W4_loss: 0.0019 - fc13_W1_accuracy: 0.9995 - fc13_W2_accuracy: 0.9995 - fc13_W3_accuracy: 1.0000 - fc13_W4_acc
uracy: 1.0000 - val_loss: 0.1860 - val_fc13_W1_loss: 0.0449 - val_fc13_W2_loss: 0.1181 - val_fc13_W3_loss: 0.0226 - val_fc13_W4
_loss: 3.7681e-04 - val_fc13_W1_accuracy: 0.9880 - val_fc13_W2_accuracy: 0.9840 - val_fc13_W3_accuracy: 0.9920 - val_fc13_W4_ac
curacy: 1.0000 - lr: 4.0960e-04
Epoch 64: early stopping
```

由上圖可以看出，在訓練尾聲時，training accuracy 可以到達接近 100 %，而 validation accuracy 可以到達 99 %左右的準確度。至於訓練時間，總時長約為 1600 s，單一 epoch 的訓練時間約為 25 s 左右，與 model 1 差不多。接著看 testing accuracy，結果如下

```
3000 picture total wrong = 104
image accuracy =  0.9653333333333334
word accuracy =  0.98975
average per execute time: 323.102378 ms
total execute time = 1046.821111 s
```

Testing accuracy 來到 96.533%，總運算時間為 1046.82 s。

5. 討論 ：

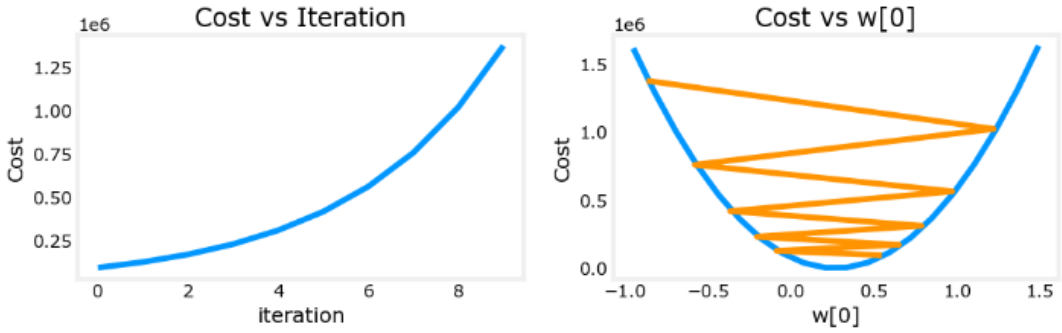
- Model 1 v.s. model 2 : 兩個模型的差別主要在於 model 2 多加了三層的 convolutional layer 和一層的 drop out layer。在準確度的表現上，可以看出 model 2 比 model 1 高一點，但是預測所

需要花費的時間也高一些。至於哪個模型比較好，我認為要根據使用場合來決定。舉例來說，如果今天是農曆年要搶高鐵票，那我會傾向於使用 model 1，這是因為農曆年有很多人要搶高鐵票，時間會是一個很重要的因素，如果在驗證碼的地方卡太久，高鐵票很可能就被搶光了。但是如果今天只是在學期中想要買張學生票回家，那我覺得 model 2 是個不錯的選擇，因為這時不會有太多人要一起搶票，自然可以選擇準確度較高的模型來幫助我們。

## III. 心得總結 ：

MNIST 資料集做起來比較容易，因為他只需要去辨認一張含有一個數字的圖像就可以了，也因此用小模型就可以有很不錯的準確度。在 MNIST 資料集的作業中，我覺得主要的收穫就是了解到 CNN 與一般 NN 之間的差距。圖像辨識要做得好，特徵就要抓得好，而 CNN 剛好可以很有效地抓取到重要特徵，此外，他還可以自己判斷並將不重要的特徵捨去掉，這樣的特性讓 CNN 在影像辨識的問題中成為一個非常好的選擇。利用 convolutional layer 還有是當的配合 drop out layer 與 max pooling layer，除了可以讓 CNN 更容易地抓取到重要的特徵，還可以很有效的避免掉 overfitting 的問題，我認為這就是 CNN 強大的地方，不像是一般 NN，很容易會造成 overfitting 的問題，泛化能力也沒有 CNN 來的好。

高鐵辨識碼的作業中，我覺得最大的收穫就是學習到如何訓練一個 CNN，以及調整參數會造成甚麼影響。其中我覺得最重要的就是 CNN 的架構以及 learning rate 的調整。網路架構的重要性在於適不適合拿來作該訓練資料的架構。太大的架構會在訓練上與預測上花費很多時間，成效也未必會比較好。太小的架構雖然訓練與預測都會比較快速，但是對於比較複雜的資料來說，準確度會無法有效提升。Learning rate 的重要性在於是否可以有效的讓模型進步。太小的 learning rate 其實不會有太大的問題，只要願意花費比較多的時間來讓 loss 慢慢下降就可以了，但是果 learning rate 太大，loss 會無法有效的下降，就像是下面的圖一樣




由於 learning rate 太大的關係，每一次更新權重反而讓 cost 在 loss function 上跳躍，造成 cost 不降反升。為了避免掉這樣的情況，我認為利用 callbacks 是一個很好的選擇。在一開始訓練用比較大的 learning rate，當訓練趨近飽和時，下調 learning rate 的大小，這樣就可以有效避免掉訓練太慢以及 learning rate 太大的問題。

# IV. Reference

[1] maxmilian “thsrc_captcha” https://github.com/maxmilian/thsrc_captcha

[2] gary9987 “keras-TaiwanHighSpeedRail-captcha” https://github.com/gary9987/keras-TaiwanHighSpeedRail-captcha

[3] deepchecks “ Learning Rate in Machine Learning” https://deepchecks.com/glossary/learning-rate-in-machine-learning/

[4] erhwenkuo “deep-learning-with-keras-notebooks” https://github.com/erhwenkuo/deep-learning-with-keras-notebooks/blob/master/2.7-mnist-recognition-cnn.ipynb

[5] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com/