# MNIST：調整參數 (CNN 準確率需高於 MLP)

資管碩二 7107029022 邱靖詒

## ◆ MNIST Handwritten Identification Dataset

### A. Packages

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import itertools
import keras
from keras import layers
from keras import models
from keras import optimizers
from keras.datasets import mnist
from keras.utils import to_categorical, np_utils
from keras.models import Sequential
from keras.optimizers import RMSprop
from keras.layers import Dense,Dropout,Flatten,Conv2D,MaxPooling2D
from sklearn.metrics import confusion_matrix
```

### B. Accuracy drawing function

In [2]:
```python
# train/test趨勢圖
def show_train_history(history, train, validation, modeltype, num, epochs):
    plt.plot(history.history[train], linewidth=3)
    plt.plot(history.history[validation], linewidth=3)
    plt.title('Train History')
    my_x_ticks = np.arange(0,epochs,1)
    plt.xticks(my_x_ticks)
    plt.ylabel(train)
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='best')
    plt.grid(True)
    if train == 'acc':
        plt.savefig("image/MNIST_acc_model_" + modeltype + str(num) + ".jpg", dpi=300)
    if train == 'loss':
        plt.savefig("image/MNIST_loss_model_" + modeltype + str(num) + ".jpg", dpi=300)
    plt.show()
```

## ◆ MLP

### A. Definition of training set and testing set

In [3]:
```python
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 將每一幅影像都轉換為一個長向量，大小為28*28=784
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# 將影像的畫素歸到0~1
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
60000 train samples
10000 test samples
```

In [5]:
```python
# 將類別向量轉換為二進制矩陣
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

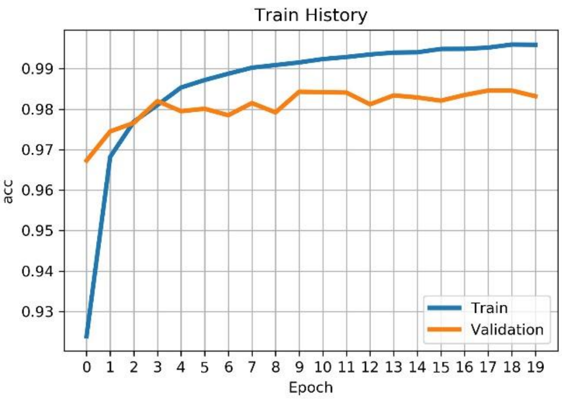
### B. Parameter setting

- modeltype = 'MLP'
- optimizer = 'rmsprop'
- batch_size = 128
- num_classes = 10
- epochs = 20
- verbose = 1

## C. Model summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 512)               401920    
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656    
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130      
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
```

## D. Result





Test Accuracy : 0.983200

Test Loss : 0.112051

# ◆ CNN

## A. Definition of training set and testing set

MLP 因為直接送進神經元處理，所以 60000 筆轉換為一筆成 28x28 = 784 個神經元輸入。CNN 因為必須先進行卷積和池化 (Max-Pool) 運算，所以必須保留影像的維度，因此 60000 筆轉換成一筆成 28 (長) x 28(寬) x 1(高)的影像單位。

先把資料讀取與轉換，再把 Features 進行標準化與 Label 的 One-Hot encoding。

In [40]:
```
np.random.seed(10)
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train40 = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32')
x_test40 = x_test.reshape(x_test.shape[0], 28, 28, 1).astype('float32')
print(x_train40.shape[0], 'train samples')
print(x_test40.shape[0], 'test samples')

x_train40_norm = x_train40 / 255
x_test40_norm = x_test40 /255

y_trainOneHot = np_utils.to_categorical(y_train)
y_testOneHot = np_utils.to_categorical(y_test)
```

```
60000 train samples
10000 test samples
```

## B. CNN Model 1

### (i) Parameter setting

- modeltype = 'CNN'
- optimizer = 'sgd'
- batch_size = 64

- epochs = 20
- verbose = 1

**(ii) Model 1 summary**

模型設計 Max-Pooling 運算可以把影像縮減取樣(downsampling)，比如原本影像是 4x4，經過 Max-Pooling 運算後，影像大小為 2x2，其優點為減少需要處理的資料點、讓影像位置的差異變小、參數的數量和計算量下降(避免 Overfitting 狀況)

padding：補 0 策略，使用「same」，代表保留邊界處的卷積結果，通常會導致輸出 shape 與輸入 shape 相同。

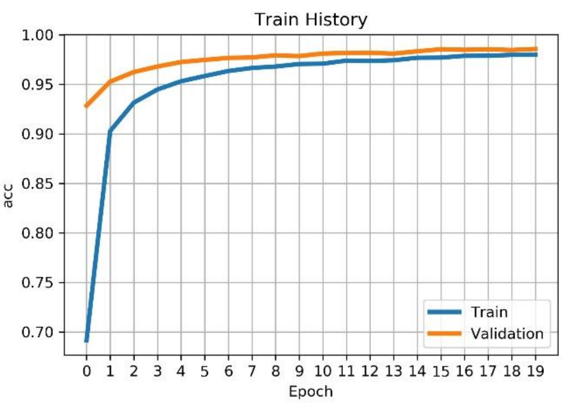
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 16)        416
_________________________________________________________________
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
_________________________________________________________________
flatten_1 (Flatten)          (None, 1764)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               225920
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 242,062
Trainable params: 242,062
Non-trainable params: 0
_________________________________________________________________
```

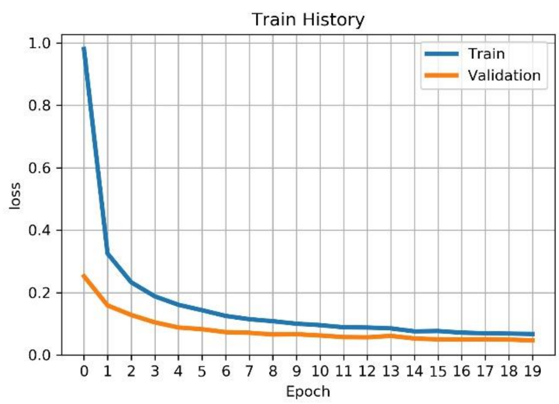
卷積層 1＋ 池化層 1

卷積層 2＋ 池化層 1

神經網路(平坦層、隱藏層、輸出層)

**(iii) Result**




Test Accuracy : 0.988400

Test Loss : 0.035157

## C. CNN Model 2

**(i) Parameter setting**

- modeltype = 'CNN'
- optimizer = 'rmsprop'
- batch_size = 128
- epochs = 20
- verbose = 1

與 CNN Model 1 相比，修改優化器為 RMSprop，batch size 改為 128。

### (ii) Model 2 summery

與 CNN Model 1 相比，

相同：激發函數 RELU

不同：修改 kernel_size 為 3x3，filters 輸出維度也有調整，Dropout 有 0.25 和 0.5。

```
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #
     =================================================================
[1]  conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
     _________________________________________________________________
[2]  conv2d_4 (Conv2D)            (None, 24, 24, 64)        18496
     _________________________________________________________________
[3]  max_pooling2d_3 (MaxPooling2 (None, 12, 12, 64)        0
     _________________________________________________________________
[4]  dropout_4 (Dropout)          (None, 12, 12, 64)        0
     _________________________________________________________________
[5]  flatten_2 (Flatten)          (None, 9216)              0
     _________________________________________________________________
[6]  dense_6 (Dense)              (None, 128)               1179776
     _________________________________________________________________
[7]  dropout_5 (Dropout)          (None, 128)               0
     _________________________________________________________________
[8]  dense_7 (Dense)              (None, 10)                1290
     =================================================================
     Total params: 1,199,882
     Trainable params: 1,199,882
     Non-trainable params: 0
     _________________________________________________________________
```

[1] 使用 32 個卷積濾波器，每個濾波器的大小為 3x3。

[2] 使用 64 個卷積濾波器，每個濾波器的大小為 3x3。

[3] 選擇最佳者進行池化。

[4] 隨機打開和關閉神經元來改善收斂。

[5] 因維度大而使用平坦，只需要輸出分類。

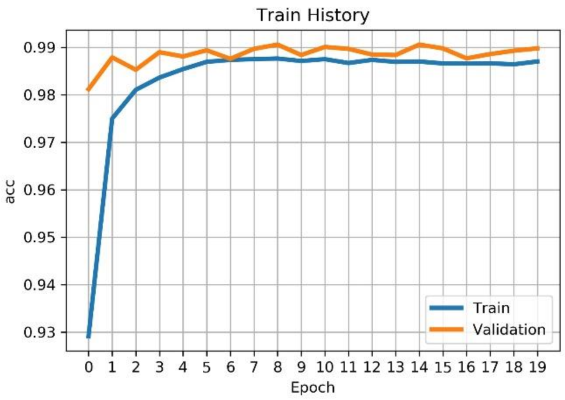
[6] 全連接層以獲取所有相關數據。

[7] 使數據更加收斂。

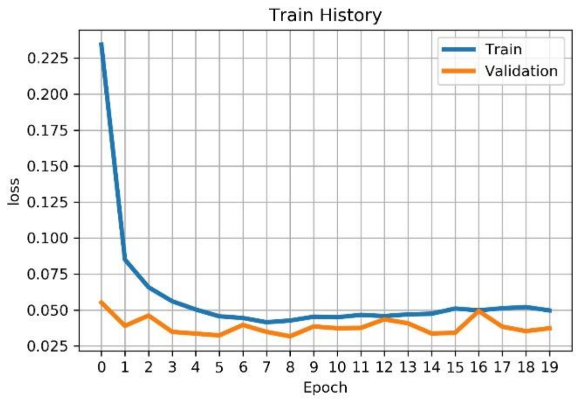
[8] 輸出 softmax 把矩陣壓縮為輸出機率。

### (iii) Result

效果比 CNN Model 1 好一些。




Test Accuracy : 0.989800




Test Loss : 0.037317

## D. CNN Model 3

### (i) Parameter setting

- modeltype = 'CNN'
- optimizer = 'rmsprop'
- batch_size = 128
- epochs = 20
- verbose = 1

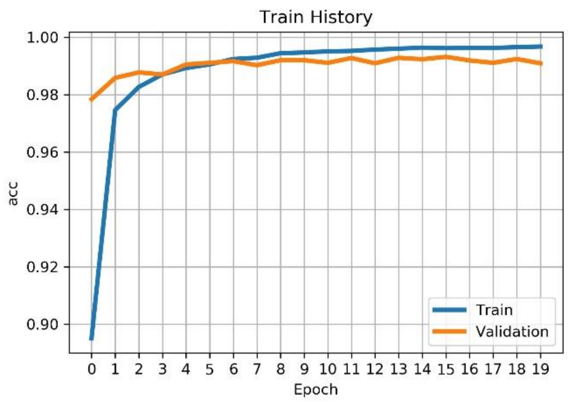
### (ii) Model 3 summery

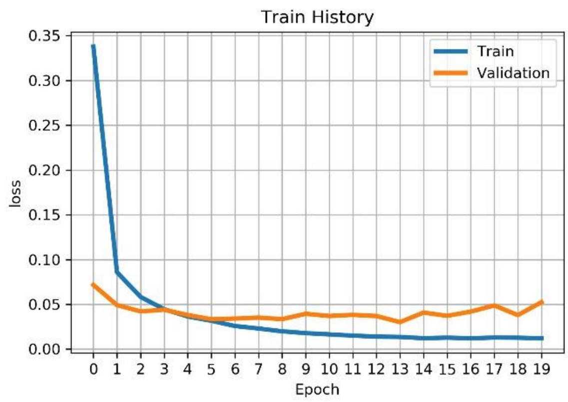
與 CNN Model 2 相比，修改 Dense 層之參數。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_3 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               73856
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 130,890
Trainable params: 130,890
Non-trainable params: 0
_________________________________________________________________
```

### (iii) Result

CNN Model 1、CNN Model 2、CNN Model 3 三個模型，以 CNN Model 3 效果最好，準確率高達 0.997。




Test Accuracy : 0.997400

Test Loss : 0.0.12839

- **CNN Model Selection Experiment**

CNN 架構有很多選擇，如何選擇「最佳」的模型架構？「最佳」之定義可以是架構最簡單的，也可以是架構能有效的提高準確率，以下是針對 MNIST 手寫數字辨識資料集來提出不同 CNN 架構實驗。

代號：

24C5 代表使用 filter 的 kernel size 是 5x5 和 stride 為 1 的卷積層，帶有 24 feature maps 。

24C5S2 代表使用 filter 的 kernel size 是 5x5 和 stride 為 2 的卷積層，帶有 24 feature maps 。

P2 代表使用 filter 的 kernel size 是 2x2 和 stride 為 2 的最大池化層。

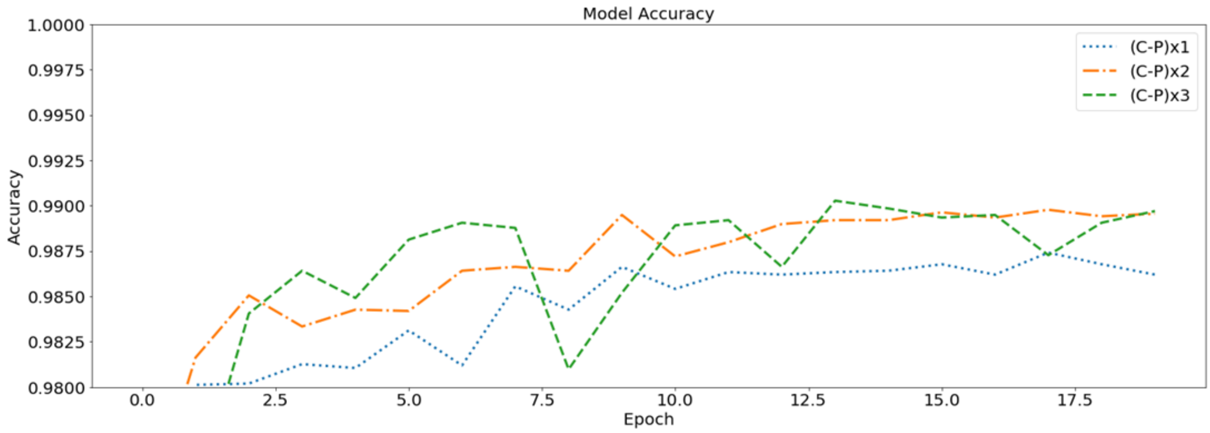
256 代表 256 個單元的全連接層。

### (i) Convolution-subsampling Pairs

input image 28x28 → one pair 14x14 → two pairs 7x7→ three pairs 4x4

結果：

```
CNN (C-P)x1: Epochs=20, Train accuracy=0.99996, Validation accuracy=0.98742
CNN (C-P)x2: Epochs=20, Train accuracy=0.99996, Validation accuracy=0.98978
CNN (C-P)x3: Epochs=20, Train accuracy=0.99986, Validation accuracy=0.99028
```




從上面的實驗中，three pairs 卷積效果看似比 two pairs 卷積還要好，但為了提高效率，這種改善方法並不能保證不會產生額外的計算成本，因此選擇使用 two pais。

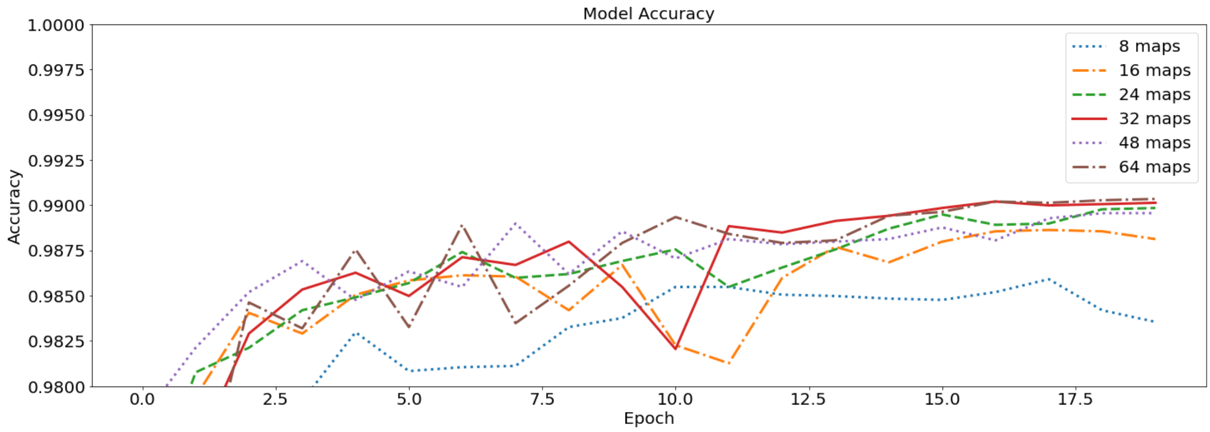
### (ii) Feature Maps

根據上一個步驟，認為 two pairs 已足夠，現在要決定要用多少 feature maps。

- [8C5-P2] - [16C5-P2]
- [16C5-P2] - [32C5-P2]
- [24C5-P2] - [48C5-P2]
- [32C5-P2] - [64C5-P2]
- [48C5-P2] - [96C5-P2]
- [64C5-P2] - [128C5-P2]

結果：

```
CNN 8 maps: Epochs=20, Train accuracy=0.99946, Validation accuracy=0.98591
CNN 16 maps: Epochs=20, Train accuracy=1.00000, Validation accuracy=0.98863
CNN 24 maps: Epochs=20, Train accuracy=0.99996, Validation accuracy=0.98985
CNN 32 maps: Epochs=20, Train accuracy=0.99996, Validation accuracy=0.99020
CNN 48 maps: Epochs=20, Train accuracy=0.99996, Validation accuracy=0.98956
CNN 64 maps: Epochs=20, Train accuracy=0.99996, Validation accuracy=0.99035
```




從上面的實驗可以看出，第一卷積層中的 32 個 maps 和第二卷積層中的 64 個 maps 是最好的。

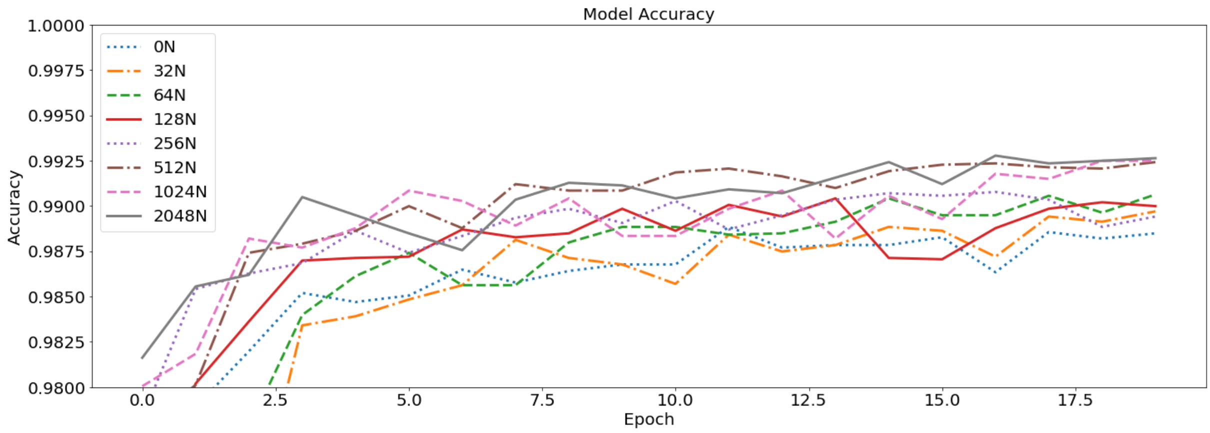
### (iii) Dense Layer

根據上一個步驟，決定使用 32 和 64 個 feature maps，現在要決定要用多少層 dense layer。

- [32C5-P2] - [64C5-P2] - 0
- [32C5-P2] - [64C5-P2] - 32
- [32C5-P2] - [64C5-P2] - 64
- [32C5-P2] - [64C5-P2] – 128
- [32C5-P2] - [64C5-P2] - 256
- [32C5-P2] - [64C5-P2] – 512
- [32C5-P2] - [64C5-P2] - 1024
- [32C5-P2] - [64C5-P2] - 2048

結果：

```
CNN 0N: Epochs=20, Train accuracy=0.99993, Validation accuracy=0.98885
CNN 32N: Epochs=20, Train accuracy=0.99982, Validation accuracy=0.98970
CNN 64N: Epochs=20, Train accuracy=0.99986, Validation accuracy=0.99063
CNN 128N: Epochs=20, Train accuracy=0.99996, Validation accuracy=0.99042
CNN 256N: Epochs=20, Train accuracy=0.99996, Validation accuracy=0.99078
CNN 512N: Epochs=20, Train accuracy=0.99996, Validation accuracy=0.99242
CNN 1024N: Epochs=20, Train accuracy=0.99996, Validation accuracy=0.99249
CNN 2048N: Epochs=20, Train accuracy=0.99996, Validation accuracy=0.99278
```



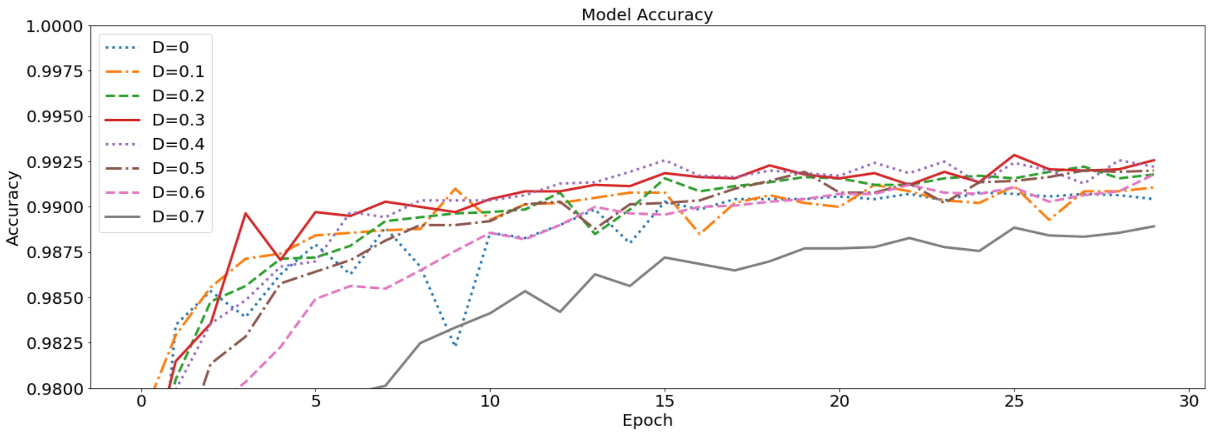

從實驗結果得知，以 Dense 的 units 設為 128 (輸出維度為 128) 效果最好。

### (iv) Dropout

Dropout 可防止過擬合，分為 0%、10%、20%、30%、40%、50%、60%、70%八種情況。

結果：

```
CNN D=0: Epochs=30, Train accuracy=1.00000, Validation accuracy=0.99078
CNN D=0.1: Epochs=30, Train accuracy=0.99971, Validation accuracy=0.99121
CNN D=0.2: Epochs=30, Train accuracy=0.99864, Validation accuracy=0.99221
CNN D=0.3: Epochs=30, Train accuracy=0.99729, Validation accuracy=0.99285
CNN D=0.4: Epochs=30, Train accuracy=0.99390, Validation accuracy=0.99256
CNN D=0.5: Epochs=30, Train accuracy=0.98951, Validation accuracy=0.99199
CNN D=0.6: Epochs=30, Train accuracy=0.98147, Validation accuracy=0.99178
CNN D=0.7: Epochs=30, Train accuracy=0.96455, Validation accuracy=0.98892
```




從實驗中發現，以 Dropout 為 0.3 的效果最好。

## (v) Advanced Features

除了使 kernel size 為 5x5 以外，也可使用兩個連續的 3x3，並搭配使用 strides = 2 的捲積層進行二次採樣，而不是使用最大池化層，最後再加上批量標準化 batch normalization 和資料擴增 data augmentation。

- 使用 '32C3-32C3' 來取代 '32C5'

```
j=1
model[j] = Sequential()
model[j].add(Conv2D(32,kernel_size=3,activation='relu',input_shape=(28,28,1)))
model[j].add(Conv2D(32,kernel_size=3,activation='relu'))
model[j].add(MaxPool2D())
model[j].add(Dropout(0.4))
model[j].add(Conv2D(64,kernel_size=3,activation='relu'))
model[j].add(Conv2D(64,kernel_size=3,activation='relu'))
model[j].add(MaxPool2D())
model[j].add(Dropout(0.4))
model[j].add(Flatten())
model[j].add(Dense(128, activation='relu'))
model[j].add(Dropout(0.4))
model[j].add(Dense(10, activation='softmax'))
model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

- 使用'32C5S2' 來取代 'P2'

```
j=2
model[j] = Sequential()
model[j].add(Conv2D(32,kernel_size=5,activation='relu',input_shape=(28,28,1)))
model[j].add(Conv2D(32,kernel_size=5,strides=2,padding='same',activation='relu'))
model[j].add(Dropout(0.4))
model[j].add(Conv2D(64,kernel_size=5,activation='relu'))
model[j].add(Conv2D(64,kernel_size=5,strides=2,padding='same',activation='relu'))
model[j].add(Dropout(0.4))
model[j].add(Flatten())
model[j].add(Dense(128, activation='relu'))
model[j].add(Dropout(0.4))
model[j].add(Dense(10, activation='softmax'))
model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

- 增加 batch normalization

```
j=3
model[j] = Sequential()
model[j].add(Conv2D(32,kernel_size=3,activation='relu',input_shape=(28,28,1)))
model[j].add(BatchNormalization())
model[j].add(Conv2D(32,kernel_size=3,activation='relu'))
model[j].add(BatchNormalization())
model[j].add(Conv2D(32,kernel_size=5,strides=2,padding='same',activation='relu'))
model[j].add(BatchNormalization())
model[j].add(Dropout(0.4))
model[j].add(Conv2D(64,kernel_size=3,activation='relu'))
model[j].add(BatchNormalization())
model[j].add(Conv2D(64,kernel_size=3,activation='relu'))
model[j].add(BatchNormalization())
model[j].add(Conv2D(64,kernel_size=5,strides=2,padding='same',activation='relu'))
model[j].add(BatchNormalization())
model[j].add(Dropout(0.4))
model[j].add(Flatten())
model[j].add(Dense(128, activation='relu'))
model[j].add(Dropout(0.4))
model[j].add(Dense(10, activation='softmax'))
model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

- 增加 data augmentation

```
j=4
model[j] = Sequential()

model[j].add(Conv2D(32,kernel_size=3,activation='relu',input_shape=(28,28,1)))
model[j].add(BatchNormalization())
model[j].add(Conv2D(32,kernel_size=3,activation='relu'))
model[j].add(BatchNormalization())
model[j].add(Conv2D(32,kernel_size=5,strides=2,padding='same',activation='relu'))
model[j].add(BatchNormalization())
model[j].add(Dropout(0.4))

model[j].add(Conv2D(64,kernel_size=3,activation='relu'))
model[j].add(BatchNormalization())
model[j].add(Conv2D(64,kernel_size=3,activation='relu'))
model[j].add(BatchNormalization())
model[j].add(Conv2D(64,kernel_size=5,strides=2,padding='same',activation='relu'))
model[j].add(BatchNormalization())
model[j].add(Dropout(0.4))

model[j].add(Flatten())
model[j].add(Dense(128, activation='relu'))
model[j].add(BatchNormalization())
model[j].add(Dropout(0.4))
model[j].add(Dense(10, activation='softmax'))

model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

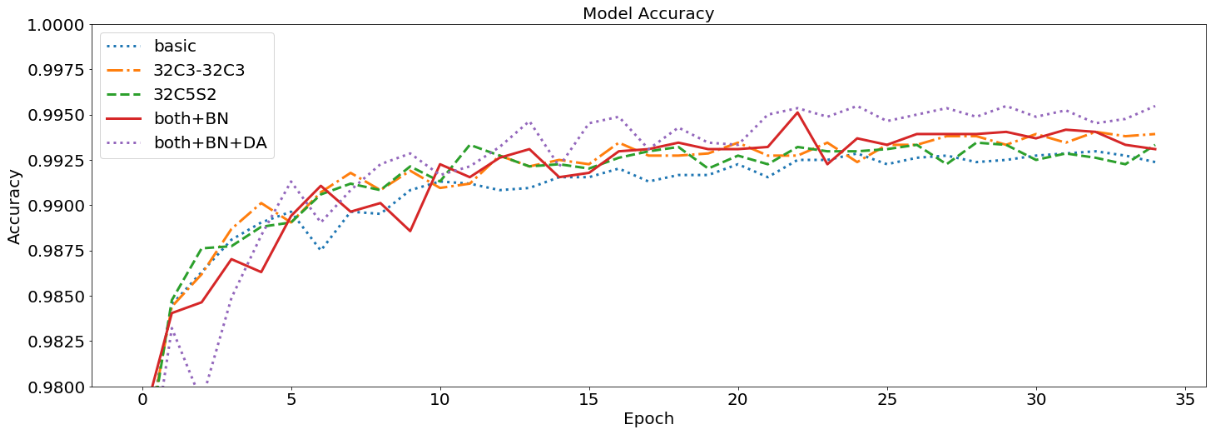
```
X_train2, X_val2, Y_train2, Y_val2 = train_test_split(X_train, Y_train, test_size = 0.2)
history = [0] * nets
names = ["basic","32C3-32C3","32C5S2","both+BN","both+BN+DA"]
epochs = 35
for j in range(nets-1):
    history[j] = model[j].fit(X_train2,Y_train2, batch_size=64, epochs = epochs,
        validation_data = (X_val2,Y_val2), callbacks=[annealer], verbose=0)
    print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}".format(
        names[j],epochs,max(history[j].history['acc']),max(history[j].history['val_acc']) ))

datagen = ImageDataGenerator(
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1)

j = nets-1
history[j] = model[j].fit_generator(datagen.flow(X_train2,Y_train2, batch_size=64),
    epochs = epochs, steps_per_epoch = X_train2.shape[0]//64,
    validation_data = (X_val2,Y_val2), callbacks=[annealer], verbose=0)
print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}".format(
    names[j],epochs,max(history[j].history['acc']),max(history[j].history['val_acc']) ))
```

結果：

```
CNN basic: Epochs=35, Train accuracy=0.99530, Validation accuracy=0.99298
CNN 32C3-32C3: Epochs=35, Train accuracy=0.99687, Validation accuracy=0.99405
CNN 32C5S2: Epochs=35, Train accuracy=0.99893, Validation accuracy=0.99345
CNN both+BN: Epochs=35, Train accuracy=0.99917, Validation accuracy=0.99512
CNN both+BN+DA: Epochs=35, Train accuracy=0.99476, Validation accuracy=0.99548
```




實驗結果發現，四種方法都提高了模型的準確率，而最後一個步驟將四種方法結合在一起，其準確率高達 99.5%，或許訓練時間迭代更多次，效果會更好。

## (vi) Conclusion

綜合以上實驗的結果，MNIST 手寫數字辨識資料集的「最佳」模型為：

1. 使用[32C3-32C3-32C5S2] - [64C3-64C3-64C5S2] - 128 的架構。
2. 搭配 Dropout 為 0.3 (=30%)、增加 batch normalization、增加 data augmentation。