# 圖樣辨識

## 專題製作
## 期末報告書

主題: 手寫數字辨識

組長: 劉家豪

組員 1: 郭先旻

組員 2: 彭子豪

# 目錄

# 圖目錄

# 壹、前言

## 一、動機

手寫數字辨識適合做為學習圖樣辨識、深度學習的入門題目，各個教學時常都以這道題目由淺入深的帶領學生學習，此題目擁有著完善的資料、方法和數據，並且容易找到 step by step 的教學，我們這組所有人都是第一次接觸此領域，所以具有上述優勢的這項題目是很吸引我們的。

## 二、目的

學習是我們最大的目的，藉由理解手寫數字辨識的原理及運作，進而學習圖樣辨識的基礎，並在實作的過程中了解如何建立、訓練、測試模型，以及了解使用深度學習這項技術可能會遭遇的問題有哪些。

# 貳、研究方法

## 一、mnist 數據集

最開始我們遇到的問題是訓練模型所需要的數據，一般的情況我們可能需要先自己拿起筆寫出無數個數字，接著把它存成圖像檔放進電腦，還要再將所有的圖像檔處理好並標記上正確的答案，最後還必須將一些錯誤的檔案給清理掉，避免模型學習到錯誤的知識，這些事情需要耗費我們大量的時間，但這並不是我們這項專題的重點，事實上除了我們也有非常多人有這同樣的問題，所以 Yann LeCun、Corinna Cortes、Christopher J.C. Burges 三人提供了 mnist 數據集來解決這部分的問題。

mnist 數據集中擁有大量且乾淨的手寫數字圖片數據，幫我們省去了蒐集數據、處理數據等等事項，讓我們可以把時間花在學習建立、訓練、測試模型上。

mnist 數據來源： http://yann.lecun.com/exdb/mnist/

我們使用了 mnist 數據集中的 60000 筆 training data，以及 10000 筆的 test data。

每張圖片是 28 pixels*28 pixels， 如下圖。

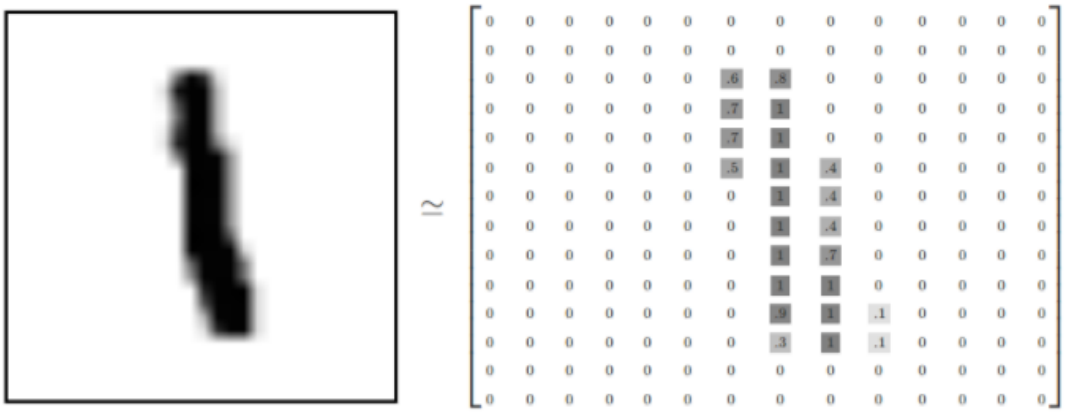

圖 1 mnist 數據解釋圖

來源：

https://fgc.stpi.narl.org.tw/activity/videoDetail/4b1141305d9cd231015d9d08fb62002d

# 二、softmax 回歸模型

SOFTMAX 回歸模型：softmax 回歸模型是一個較為常見且基本的模型， sotfmax 模型會將輸入（input）的資訊，透過乘上**權重**（weight）與加上**偏差**（bias），再經由 softmax 函式轉換成機率的方式，得到每一種類別所代表的可能性並作為輸出（output），進而決定輸入的資訊是屬於哪一種類別。

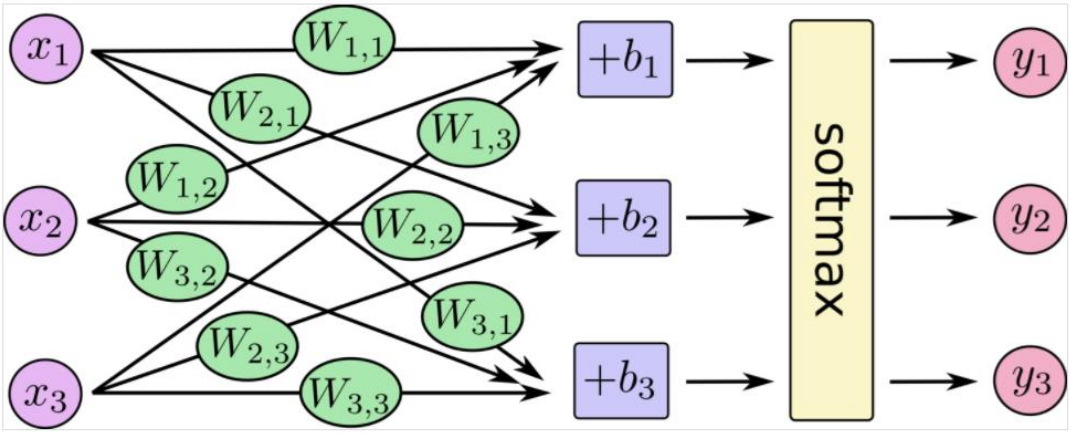


圖 2 softmax 模型架構圖

來源:

https://fgc.stpi.narl.org.tw/activity/videoDetail/4b1141305d9cd231015d9d08fb62002d

最初我們跟著網路上的教學先建立出最簡單的 softmax 回歸模型來進行手寫數字辨識。

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
import tensorflow as tf
```

```
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

```
y_ = tf.placeholder(tf.float32, [None, 10])
```

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
```

```
cross_entropy = tf.reduce_mean(
  tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
```

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

```
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
```

```
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
```

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

圖 3 softmax_mnist

訓練完的模型測試結果為辨識成功率 90.56%。

```
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

```
0.9056
```

圖 4 softmax_mnist_Accuracy

接著我們大幅增加模型的訓練次數。

```
for _ in range(10000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

圖 5 softmax_mnist_moreTraining

訓練完的模型測試結果為辨識成功率為 92.64%

```
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

```
0.9264
```

圖 6 softmax_mnist_moreTraining_Accuracy

經過多次測試，我們發現這個簡單的模型所能達到的極限大約落在辨識成功率 92%，但離我們最初訂定的目標 95%還有一段距離，所以接下來我們採用更進階的模型來達成更高的成功率。

教學資料來源：

https://blog.gtwang.org/programming/tensorflow-softmax-regression-hand-written-digits-recognizer-tutorial/

# 三、卷積神經網路(CNN)

卷積神經網絡(Convolutional Neural Network)簡稱 CNN，CNN 是所有深度學習課程、書籍必教的模型(Model)，CNN 在影像識別方面的威力非常強大，許多影樣辨識的模型也都是以 CNN 的架構為基礎去做延伸。

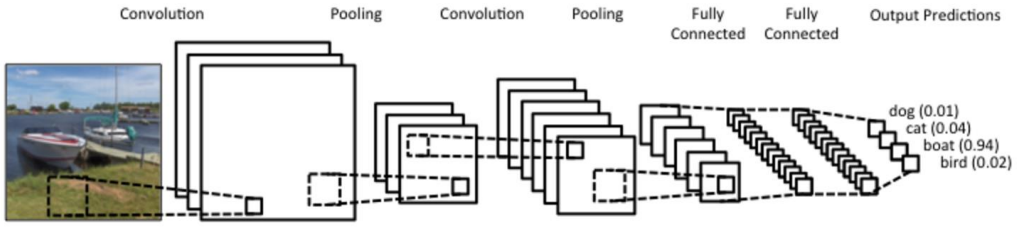


圖 7 CNN 概念圖

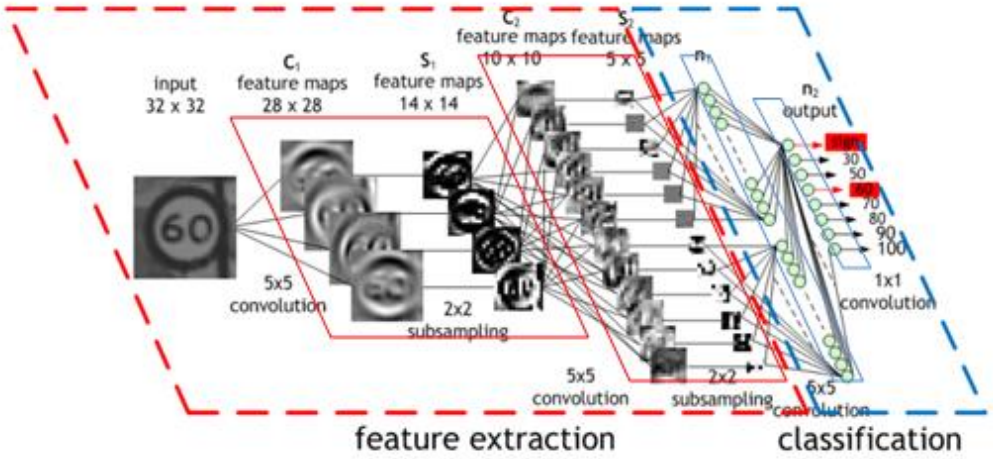


圖 8 CNN 概念圖 2

資料來源：參考資料[5]

# 1.建立模型

以上述的架構為原型，我們建立出了以下模型。

```
# build CNN maxpool model using Keras sequential model API
model_1 = Sequential()

# type your code of model here
model_1.add(Conv2D(16, (5,5), activation="relu", input_shape=input_shape, name='Conv2D_layer_1'))

model_1.add(MaxPooling2D(pool_size=(2,2), name='MaxP_layer_1'))

model_1.add(Conv2D(32, (3,3), activation="relu", name='Conv2D_layer_2'))

model_1.add(MaxPooling2D(pool_size=(2,2), name='MaxP_layer_2'))

model_1.add(Dropout(0.25, name='Drop_layer_1'))

model_1.add(Flatten())
model_1.add(Dense(64, activation="relu", name='Dense_layer_1'))
model_1.add(Dense(128, activation="relu", name='Dense_layer_2'))
model_1.add(Dropout(0.5, name='Drop_layer_2'))
model_1.add(Dense(10, activation="softmax", name='output_layer_1'))

model_1.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

print(model_1.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv2D_layer_1 (Conv2D)      (None, 24, 24, 16)        416
_________________________________________________________________
MaxP_layer_1 (MaxPooling2D)  (None, 12, 12, 16)        0
_________________________________________________________________
Conv2D_layer_2 (Conv2D)      (None, 10, 10, 32)        4640
_________________________________________________________________
MaxP_layer_2 (MaxPooling2D)  (None, 5, 5, 32)          0
_________________________________________________________________
Drop_layer_1 (Dropout)       (None, 5, 5, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 800)               0
_________________________________________________________________
Dense_layer_1 (Dense)        (None, 64)                51264
_________________________________________________________________
Dense_layer_2 (Dense)        (None, 128)               8320
_________________________________________________________________
Drop_layer_2 (Dropout)       (None, 128)               0
_________________________________________________________________
output_layer_1 (Dense)       (None, 10)                1290
=================================================================
Total params: 65,930
Trainable params: 65,930
Non-trainable params: 0
_________________________________________________________________
None
```

圖 9 CNN_mnist_CreateModel

## 2.訓練模型

接著使用 training data 來訓練此模型，訓練 20 次，大約花費了 6 分 50 秒。

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 291us/step - loss: 0.3567 - acc: 0.8857 - val_loss: 0.0710 - val_acc: 0.9755
Epoch 2/20
60000/60000 [==============================] - 17s 290us/step - loss: 0.1046 - acc: 0.9701 - val_loss: 0.0461 - val_acc: 0.9852
Epoch 3/20
60000/60000 [==============================] - 17s 288us/step - loss: 0.0771 - acc: 0.9777 - val_loss: 0.0366 - val_acc: 0.9878
Epoch 4/20
60000/60000 [==============================] - 17s 288us/step - loss: 0.0647 - acc: 0.9815 - val_loss: 0.0305 - val_acc: 0.9897
Epoch 5/20
60000/60000 [==============================] - 17s 288us/step - loss: 0.0544 - acc: 0.9843 - val_loss: 0.0294 - val_acc: 0.9901
Epoch 6/20
60000/60000 [==============================] - 17s 289us/step - loss: 0.0487 - acc: 0.9862 - val_loss: 0.0275 - val_acc: 0.9911
Epoch 7/20
60000/60000 [==============================] - 17s 291us/step - loss: 0.0452 - acc: 0.9866 - val_loss: 0.0266 - val_acc: 0.9912
Epoch 8/20
60000/60000 [==============================] - 17s 290us/step - loss: 0.0408 - acc: 0.9878 - val_loss: 0.0250 - val_acc: 0.9915
Epoch 9/20
60000/60000 [==============================] - 17s 292us/step - loss: 0.0358 - acc: 0.9892 - val_loss: 0.0274 - val_acc: 0.9920
Epoch 10/20
60000/60000 [==============================] - 18s 292us/step - loss: 0.0351 - acc: 0.9899 - val_loss: 0.0244 - val_acc: 0.9921
Epoch 11/20
60000/60000 [==============================] - 18s 292us/step - loss: 0.0335 - acc: 0.9902 - val_loss: 0.0255 - val_acc: 0.9926
Epoch 12/20
60000/60000 [==============================] - 18s 293us/step - loss: 0.0312 - acc: 0.9911 - val_loss: 0.0239 - val_acc: 0.9922
Epoch 13/20
60000/60000 [==============================] - 18s 294us/step - loss: 0.0299 - acc: 0.9911 - val_loss: 0.0220 - val_acc: 0.9932
Epoch 14/20
60000/60000 [==============================] - 17s 290us/step - loss: 0.0283 - acc: 0.9918 - val_loss: 0.0206 - val_acc: 0.9937
Epoch 15/20
60000/60000 [==============================] - 18s 303us/step - loss: 0.0256 - acc: 0.9927 - val_loss: 0.0215 - val_acc: 0.9936
Epoch 16/20
60000/60000 [==============================] - 18s 296us/step - loss: 0.0244 - acc: 0.9928 - val_loss: 0.0231 - val_acc: 0.9935
Epoch 17/20
60000/60000 [==============================] - 18s 294us/step - loss: 0.0228 - acc: 0.9930 - val_loss: 0.0217 - val_acc: 0.9938
Epoch 18/20
60000/60000 [==============================] - 18s 294us/step - loss: 0.0233 - acc: 0.9933 - val_loss: 0.0216 - val_acc: 0.9935
Epoch 19/20
60000/60000 [==============================] - 18s 296us/step - loss: 0.0230 - acc: 0.9933 - val_loss: 0.0219 - val_acc: 0.9941
Epoch 20/20
60000/60000 [==============================] - 18s 296us/step - loss: 0.0222 - acc: 0.9933 - val_loss: 0.0209 - val_acc: 0.9944
```

圖 10 CNN_mnist_trainingModel

## 3.測試模型

使用 training data 測試辨識成功率為 99.8%

```
# test the trained model using training data
score = model_1.evaluate(x_train, y_train, verbose=0)
print('Train loss:', score[0])
print('Train accuracy:', score[1])
```

```
Train loss: 0.006880404092968153
Train accuracy: 0.998
```

圖 11 CNN_mnist_trainingData_Accuracy

使用 test data 測試辨識成功率為 99.44%

```
# test the trained model using testing data
score = model_1.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.020870362593263417
Test accuracy: 0.9944
```

圖 12 CNN_mnist_testData_Accuracy

# 參、學習成果

透過此專題我們已經了解如何建立、訓練、測試模型，並且了解到一些我們所遇到的問題。

## 1. 隨機性

在訓練模型的時候，kernel 是隨積的，被丟掉的神經元也是隨積的，初始的權重也是隨積的，同一個模型所訓練出來的結果也不相同，需要花費大量的時間來重新訓練才能得出較好的結果。

## 2. Overfitting(過擬合)

以手寫數字辨識並且使用 mnist 數據集的情況來討論，發生 Overfitting 的機率比 Underfitting(欠擬合)的機率高很多。

Overfitting 會造成模型對於 training data 擁有非常高的辨識成功率，但對於 test data 的辨識成功率反而降低，我們建立的模型已經藉由使用 Dropout 來降低發生 Overfitting 的機率，但相對的也會增加 Underfitting 的機率。

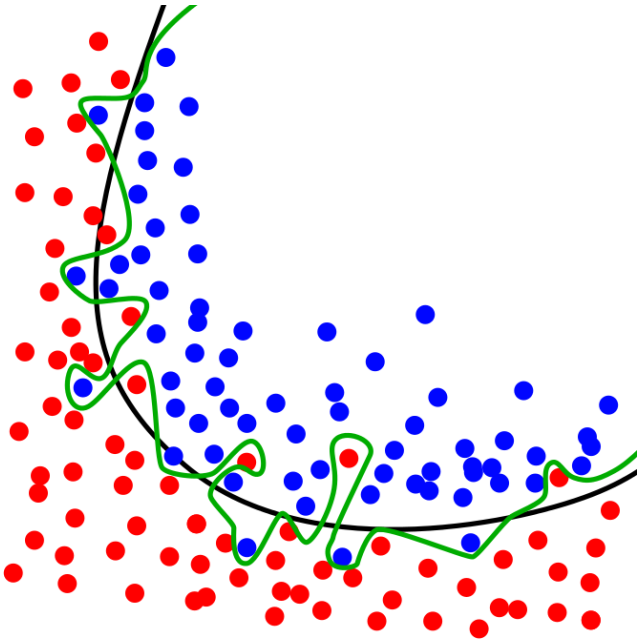

圖 13 Overfitting

圖片來源：

https://zh.wikipedia.org/wiki/%E9%81%8E%E9%81%A9

## 3. 如何建立出更好的模型

建立出更好的模型是一件很困難的事，根據理論較多層的深度學習模型會擁有更佳的效果，但由於隨機性和 Overfitting 都可能導致辨識成功率不增反減，建出更好的模型成為了一項大難題。

# 肆、參考資料


[1]http://yann.lecun.com/exdb/mnist/

[2]https://fgc.stpi.narl.org.tw/activity/videoDetail/4b1141305d9cd231015d9d08fb62002d

[3]https://fgc.stpi.narl.org.tw/activity/videoDetail/4b1141305d9cd231015d9d08fb62002d

[4]https://blog.gtwang.org/programming/tensorflow-softmax-regression-hand-written-digits-recognizer-tutorial/

[5]https://medium.com/@yehjames/%E8%B3%87%E6%96%99%E5%88%86%E6%9E%90-%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E7%AC%AC5-1%E8%AC%9B-%E5%8D%B7%E7%A9%8D%E7%A5%9E%E7%B6%93%E7%B6%B2%E7%B5%A1%E4%BB%8B%E7%B4%B9-convolutional-neural-network-4f8249d65d4f

[6]https://zh.wikipedia.org/wiki/%E9%81%8E%E9%81%A9