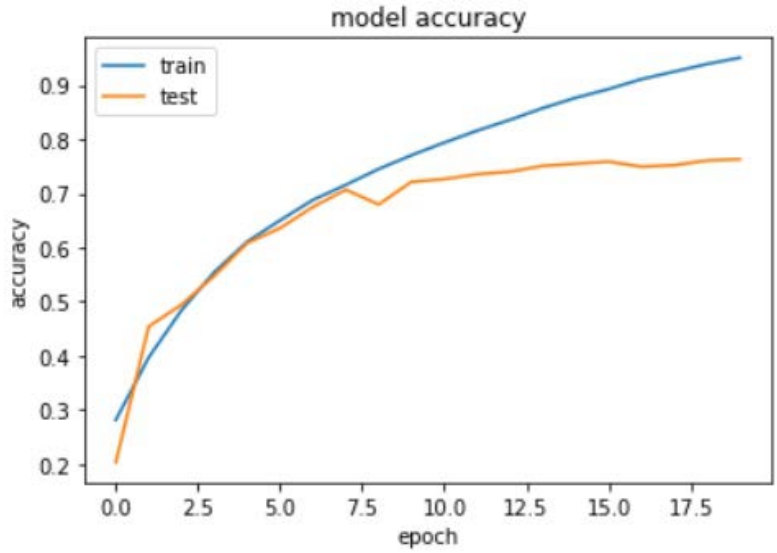
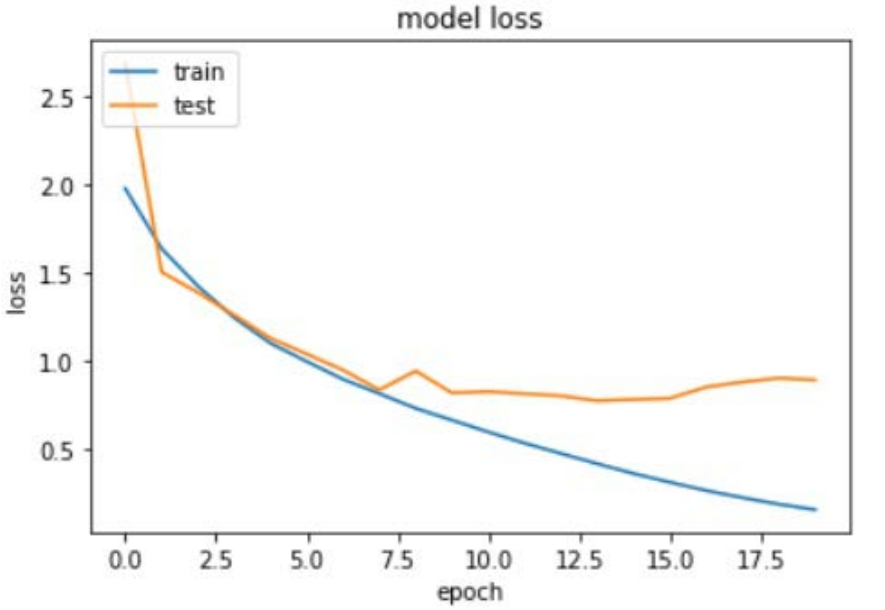
E14056114 徐顯舜 機械 109

1.結果呈現



雖然一開始的正確率不高，但經過幾次的訓練過程後越來越趨近於 **76%**

```
Epoch 20/20
40000/40000 [==============================
Test loss: 0.9276220597267151
Test accuracy: 0.7585
```



可以發現 loss 越低，準確率越高，而最後的 loss 還是些微

的上升，但跟原始 cnn 程式相比好了一些。

```
40000/40000 [==============================] - 299s 7ms/step - loss: 1.0969 - acc: 0.6108 - val_loss: 1.1247 - val_acc: 0.6084
Epoch 6/20
40000/40000 [==============================] - 294s 7ms/step - loss: 0.9932 - acc: 0.6502 - val_loss: 1.0361 - val_acc: 0.6357
Epoch 7/20
40000/40000 [==============================] - 294s 7ms/step - loss: 0.8924 - acc: 0.6879 - val_loss: 0.9441 - val_acc: 0.6749
Epoch 8/20
40000/40000 [==============================] - 295s 7ms/step - loss: 0.8111 - acc: 0.7157 - val_loss: 0.8325 - val_acc: 0.7067
Epoch 9/20
40000/40000 [==============================] - 298s 7ms/step - loss: 0.7279 - acc: 0.7450 - val_loss: 0.9397 - val_acc: 0.6796
Epoch 10/20
40000/40000 [==============================] - 305s 8ms/step - loss: 0.6606 - acc: 0.7705 - val_loss: 0.8148 - val_acc: 0.7214
Epoch 11/20
40000/40000 [==============================] - 300s 7ms/step - loss: 0.5936 - acc: 0.7939 - val_loss: 0.8228 - val_acc: 0.7265
Epoch 12/20
40000/40000 [==============================] - 303s 8ms/step - loss: 0.5299 - acc: 0.8159 - val_loss: 0.8100 - val_acc: 0.7353
Epoch 13/20
40000/40000 [==============================] - 304s 8ms/step - loss: 0.4715 - acc: 0.8359 - val_loss: 0.7987 - val_acc: 0.7402
Epoch 14/20
40000/40000 [==============================] - 306s 8ms/step - loss: 0.4134 - acc: 0.8576 - val_loss: 0.7710 - val_acc: 0.7507
Epoch 15/20
40000/40000 [==============================] - 306s 8ms/step - loss: 0.3579 - acc: 0.8767 - val_loss: 0.7781 - val_acc: 0.7549
Epoch 16/20
40000/40000 [==============================] - 301s 8ms/step - loss: 0.3083 - acc: 0.8927 - val_loss: 0.7844 - val_acc: 0.7587
Epoch 17/20
```

(訓練過程擷取)

2.討論

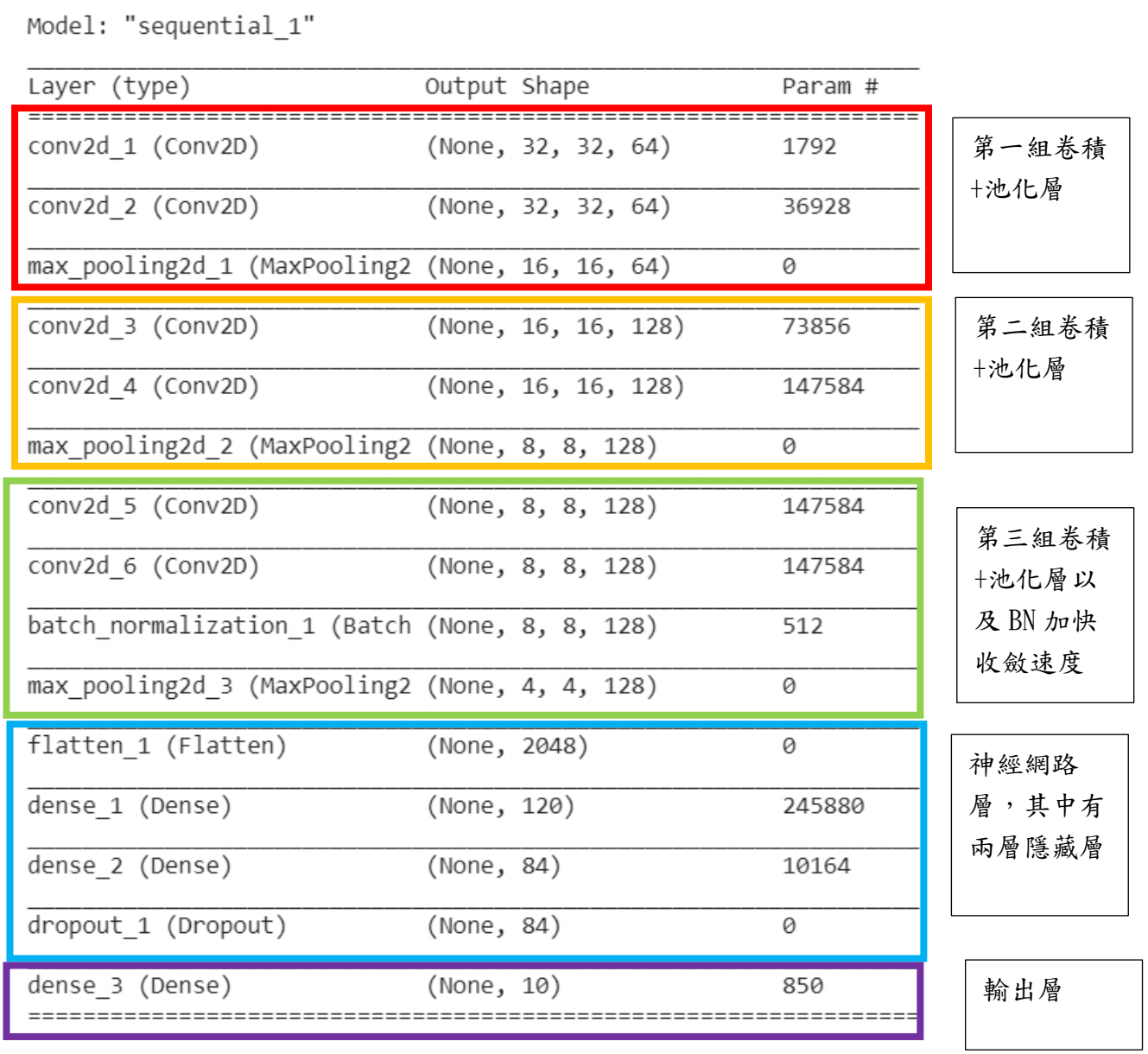
環境: Google colab

前置設定:在一開始執行以下程式

```
!pip install tensorflow==1.14.0
import tensorflow as tf
print(tf.__version__)
```

因為要使用的一些 function 貌似 Tensorflow 2.20 不支援，所以要降版至 1.14.0

整體模型流程(但看完其他網路文章發現有些層放錯位置):



這次的話其實設計的過程沒有太大的想法(主要做很多次的 convolution 和看看其他的優化器和 function)，主要是去熟悉這些 function 的參數和意義。

```
model.add(Conv2D(filters=64, kernel_size=3, input_shape=(32, 32, 3), activation='relu', padding='same'))
```

此行意思為有 64 個 filters（即我們稱的 filter 或 kernel），

kernel 大小為 3×3，卷積後輸入以及輸出的圖形大小不變（padding = ‘same’，即圖像周圍會補值），輸入的圖形維度為 32x32x3，使用 relu 啟動函數。由於定義了 64 個 filters，因此本層會輸出 64 個 16×16 的影像。

```
model.add(MaxPooling2D((2, 2), strides=(2, 2)))
```

接下來是池化層，本層參數比較少，我們只要定義 pool size 為(2,2)，即尺寸縮小為一半。本層接收了上一層卷積層的 64 個 32×32 影像後，會輸出 16 個 14×14 的影像。

```
 model.add(BatchNormalization(axis=-1))
```

使得深層神經網絡訓練更加穩定，加快了收斂的速度，甚至同時起到了正則化的作用。可以 training 加速，可以防止 gradient vanishing 的問題，可以幫助 sigmoid 或是 tanh 這種的 activation function 可以讓參數的 initialization 的影響較小

```
model.add(Flatten())
```

接著我們需要一個平坦層，將特徵值轉為一維資料以供後續的全連結層使用。本層接收第三個池化層傳來的 4x4x128（=2048）資料之後，轉為 1 維有 2048 的值的矩陣。

```
model.add(Dense(120, activation='relu'))
```

建立全連結層中的隱藏層(即傳統神經網路中的全連結層)，指定其神經元數目為 120 個(此數目可調整)，啟用函數使用 Relu。

```
model.add(Dropout(rate=0.25))
```

在各兩層的卷積與池化後，我們加入 Dropout 層，它最大的功用是減少過度擬合。在深度學習的訓練過程中，Dropout 會讓每次 batch run 都依據機率丟棄一定比例的神經元不予計算，使得每一次都好像在訓練不同的神經網路一樣。上方的程式中，我們定義該 Dropout 層每次訓練時要丟棄 25%的神經元。

```
model.add(Dense(10, activation='softmax'))
```

本模型的最後一層是輸出層，也就是要輸出十種 0~9 的分類值，一般我們都會使用 softmax 作為分類模型輸出層的啟動函數。

```
model.compile(loss='categorical_crossentropy',
              optimizer=ada(lr=0.01),
              metrics=['accuracy'])
```

模型建立好之後，接下來要準備訓練此模型，在訓練之前，要使用 compile 來定義訓練的參數。損失函數使用深度學習分類模型中最常用的交叉熵 cross entropy，梯度下降法使用 Adagrad(learning rate = 0.01，但看到網路上很多人建議用 adam，但還沒特別研究這些 optimizer 差別)，模型的評估方式則是以 accuracy 為優先。

瑕疵與改進地方:

查看其他網路文章發現 Dropout 和 Batch normalization 層放錯地方， Dropout 層一般加在全連結層後以防止過度擬合，提升模型泛化能力。而很少見到卷積層後接 Dropout（原因主要是卷積參數少，不易過擬合），而且 Batch normalization 可替換 Dropout，而且表現得更好。但兩者放置位置不同:只在有全連接的時候用 Dropout，在卷積之間使用 Batch normalization。

所以程式碼應該為:

```
model.add(Conv2D(filters=128, kernel_size=3, activation='relu', p
adding='same'))
model.add(BatchNormalization(axis=-1))
model.add(Activation('relu'))
```

或

```
model.add(Dense(120, activation='relu'))
model.add(Dropout(rate=0.25))
```