機器學習系統設計實務與應用

HW3 B063012054 林祐安

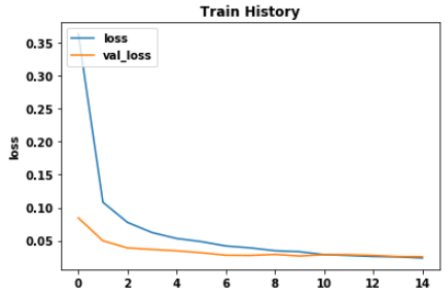
於前一次作業問題與討論提到：

原架構 Dropout->Dense->Dropout->Dense，因圖片單純、架構簡單，拿掉兩個 Dropout 是不是沒有太大影響？

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               1179776
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

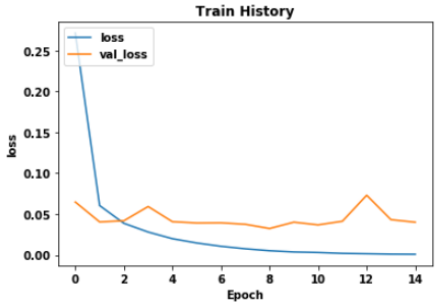
(刪除兩層 Dropout 模型架構)

```
Epoch 15/15
60000/60000 [==============================] - 5s 76us/step - loss: 0.0237 - accuracy: 0.9924 - val_loss: 0.0257 - val_accurac
y: 0.9921
Test loss: 0.02571241261522082
Test accuracy: 0.9921000003814697
```



(尚未修改前的模型)

```
Epoch 15/15
60000/60000 [==============================] - 5s 77us/step - loss: 8.4892e-04 - accuracy: 0.9998 - val_loss: 0.0400 - val_accu
racy: 0.9907
Test loss: 0.04003214264022042
Test accuracy: 0.9907000064849854
```



(刪除兩層 Dropout 的模型)

比較兩者趨勢圖，很明顯看到"刪除兩層 Dropout 的模型"於 valid loss 方面震盪，原以為 loss 無法下降，但仔細觀察其值為 0.05 左右，與"尚未修改前的模型"loss 相差約 0.015，兩者都是收斂狀態，其正確率也約 99%，確實這兩層 Dropout 對於這種單純且淺層的網路作用不大。

第二個問題與討論是紅箭頭處：

```
#improved model
model2=Sequential()
# 第一層conv2D更改Mask大小從3->9
model2.add(Conv2D(32, kernel_size=(9, 9),activation='relu',input_shape=input_shape))

model2.add(Conv2D(64, (3, 3), activation='relu'))

# 建立池化層，池化大小=2x2，取最大值
model2.add(MaxPooling2D(pool_size=(2, 2)))
# Dropout層隨機斷開輸入神經元，用於防止過度擬合，斷開比例:0.25
model2.add(Dropout(0.25))
# Flatten層把多維的輸入一維化，常用在從卷積層到全連接層的過渡。
model2.add(Flatten())
# 更改數值成0.5->0.25
model2.add(Dropout(0.25))
# 使用 softmax activation function，將結果分類
model2.add(Dense(num_classes, activation='softmax'))

# 更改optimizer Adadelta->Adam
model2.compile(loss=keras.losses.categorical_crossentropy,
               optimizer=keras.optimizers.Adam(),
               metrics=['accuracy'])

# 更改batch size 256->1024 epochs 15->10
train_history = model2.fit(x_train, y_train,
          batch_size=1024,
          epochs=10,
          #verbose=1,
          validation_data=(x_test, y_test))
#儲存訓練架構及結果
#model.save('my_model_cnn.h5')

# 顯示損失函數、訓練成果(分數)
score = model2.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])


plt.plot(train_history.history['loss'])
plt.plot(train_history.history['val_loss'])
plt.title('Train History')
plt.ylabel('loss')
plt.xlabel('Epoch')
plt.legend(['loss', 'val_loss'], loc='upper left')
plt.show()
```

紅框以及紅箭頭部分是修改的地方，紅箭頭的部份是刪除一層 Dense，紅框部分為修改參數，於上次作業已討論過，本次討論紅箭頭的部分。一般來說簡單的 CNN 模型只會用到一層全連接層，會用到兩層 FC 可能有以下考量：

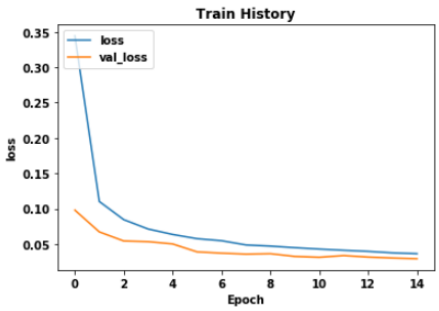
1. 減少參數量，龐大的網路例如 YOLO、VGGNet，第一層 FC 參數量可能上億，透過第二層 FC 減少運算量以及記憶體占用。
2. 影響辨識率最主要還是 Conv2D 層，額外再增加 FC 效果有限，因此大部分龐大的模型都是用 2~3 層 FC，甚至有些會用 1X1 的 Conv2D 代替所有的 FC 層。

綜觀以上兩點，於此實作模型過於淺層簡單，多一層 Dense 效果應是差不了多少。

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 9216)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 9216)              0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                92170
=================================================================
Total params: 110,986
Trainable params: 110,986
Non-trainable params: 0
_________________________________________________________________
```

(原模型刪除第一層 Dense)

```
Epoch 15/15
60000/60000 [==============================] - 5s 78us/step - loss: 0.0364 - accuracy: 0.9888 - val_loss: 0.0293 - val_accurac
y: 0.9900
Test loss: 0.02932262028489495
Test accuracy: 0.9900000095367432
```



(原模型刪除第一層 Dense)

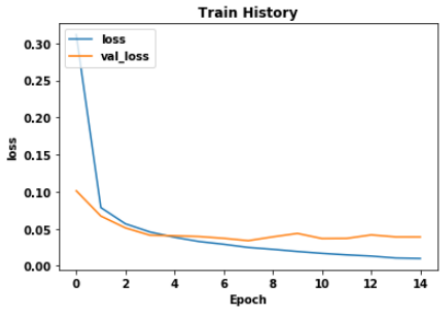
可以見到結果確實如前面所述，效果與原本模型差不多，Loss=0.03、acc=99%。

再來是將兩種情形結合，模型簡化，去除第一層 Dense 與兩層 Dropout。

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                92170
=================================================================
Total params: 110,986
Trainable params: 110,986
Non-trainable params: 0
_________________________________________________________________
```

(去除第一層 Dense 與兩層 Dropout)

```
Epoch 15/15
60000/60000 [==============================] - 4s 66us/step - loss: 0.0099 - accuracy: 0.9973 - val_loss: 0.0390 - val_accurac
y: 0.9888
Test loss: 0.0390026563316529
Test accuracy: 0.9887999892234802
```



(去除第一層 Dense 與兩層 Dropout)

可以發現訓練時間快了一些，原本 76us->66us，速度提升 13%，其代價是正確率下降 0.2%左右。

前面紅字所述，有些模型會用 1X1 的 Conv2D 代替 FC 層，以下實作：

```
model.add(MaxPooling2D(pool_size=(2, 2)))
#1st Dropout
model.add(Dropout(0.25))
# 1st FC replaced by Conv2D
model.add(Conv2D(128, (12, 12),padding='valid',activation='relu'))
#2nd Dropout
model.add(Dropout(0.5)).
# 2nd FC replaced by Conv2D
model.add(Conv2D(10, (1, 1),padding='valid',activation='softmax'))
model.add(Flatten())
# 編譯: 選擇損失函數、優化方法及成效衡量方式
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

僅更動後兩層 Dense 替換成 Conv2D，其餘皆未變動。

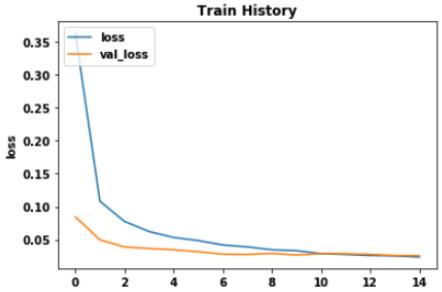
Layer (type)	Output Shape	Param #
conv2d_54 (Conv2D)	(None, 26, 26, 32)	320
conv2d_55 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_19 (MaxPooling	(None, 12, 12, 64)	0
dropout_6 (Dropout)	(None, 12, 12, 64)	0
conv2d_56 (Conv2D)	(None, 1, 1, 128)	1179776
dropout_7 (Dropout)	(None, 1, 1, 128)	0
conv2d_57 (Conv2D)	(None, 1, 1, 10)	1290
flatten_10 (Flatten)	(None, 10)	0

Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0

(two FC layers replaced by Conv2D)

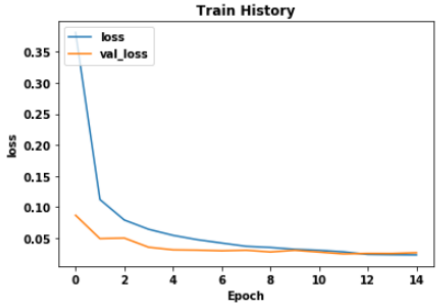
**其參數量與未更改前的模型配置相同

```
Epoch 15/15
60000/60000 [==============================] - 5s 76us/step - loss: 0.0237 - accuracy: 0.9924 - val_loss: 0.0257 - val_accurac
y: 0.9921
Test loss: 0.02571241261522082
Test accuracy: 0.9921000003814697
```



(尚未修改前的模型)

```
Epoch 15/15
60000/60000 [==============================] - 7s 111us/step - loss: 0.0232 - accuracy: 0.9930 - val_loss: 0.0268 - val_accurac
y: 0.9924
Test loss: 0.026805323436100387
Test accuracy: 0.9923999905586243
```



(two FC layers replaced by Conv2D)

觀察兩者其實沒有甚麼太大的差異，具體而言，將最後的全連接層替換層卷積層是不適當得根據不同應用決定，例如在此實例中，替換成卷積層效果並無提升太多，但耗時大大增加。
原本耗時 76us->111us，增加近 45%時間，非常耗資源，這也顯示出卷積層對計算負擔比全連接層還要重許多。近年來發展出另一個取代 FC 的做法：Global Average Pooling(GAP)，這些做法其根本目的都是要減少參數以降低 overfitting。

實作中較大的問題是該如何更動架構、加加減減甚麼層，因為盲目的一直增加 Conv2D 沒有甚麼意義，而且到了一定程度模型會爛掉，因此嘗試了一些新奇的玩意。在撰寫報告時有一段是大部分的 CNN 模型使用的 FC 是幾層，於前次作業中個人斷定 2 層 Dense 是多餘的，僅需要一層，但深入查找資料後發現 VGG 使用高達 3 層 Dense，也是目前看過最多 FC 的模型，YOLO 為 2 層，但這些是很複雜的模型，因此對於實作這種簡單的模型還是 1 層 FC 就夠了。於一篇 stackoverflow 看到有人回復：某些模型會將 FC 使用卷積層代替。額外參考了兩篇文章，因為自行修改架構總是出現維度對不上的問題，參考第二篇後才知道需額外添加一層 Flatten。

在本次作業做了許多架構更動的部分，基本上不會再增加層數，不僅會浪費時間，效果提升也不會很明顯，但替換成其他 layer 也能發現，效果有限，畢竟模型實在是不夠複雜。尋找參考資料時第一次知道可以將全連接層替換成卷積層甚至 GAP，深入實作發現這個方法好像頗雞肋，也許效果有稍微提升，但是增加的時間實在是不成比例，難怪會再發展出 GAP，具體來說也還不知道為何要使用 Conv 替代 FC，個人認為是為了減少權重參數量，避免 overfitting。

Code link