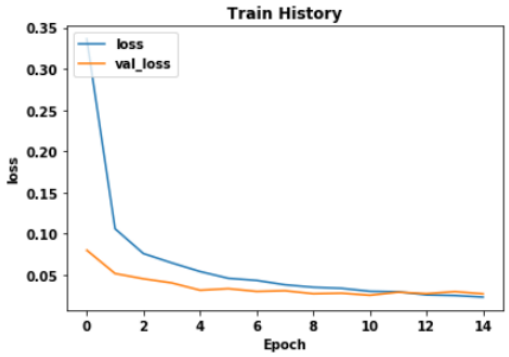
機器學習系統設計實務與應用

HW2 B063012054 林祐安

```
Epoch 15/15
60000/60000 [==============================] - 60s 997us/step - loss: 0.0230 - accuracy: 0.9925 - val_loss: 0.0269 - val_accura
cy: 0.9915
Test loss: 0.02690217216430756
Test accuracy: 0.9915000200271606
```



(未改變參數前)

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_13 (Dropout)         (None, 12, 12, 64)        0
_________________________________________________________________
flatten_7 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               1179776
_________________________________________________________________
dropout_14 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

(模型架構)

使用 CPU Ryzen5 2600 訓練每個 epoch 需耗時一分鐘左右，模型架構僅兩層卷積以及全連接層，理論上耗時不需太久，但在此次實作使用的樣本有 60000 份，數量不少，因此耗時長。為提高效率最簡單的方法是加大"batch_size"，原先為 256，由於資料有 60000 筆，可以試著把 batch_size 提高至 1024(每個 epoch 約耗時 47 秒)、4096(每個 epoch 約耗時 45 秒)，觀察發現 batch_size=1024 與 batch_size=4096 的耗時差不多，這表示再增加下去意義不大，且訓練效果可能會非常差勁。較為進階點可以修改 optimizer、甚至於模型架構。

----------以下為修改後模型----------

```
#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()
```

(修改後 code)

紅框以及紅箭頭部分是修改的地方，為防止第三次作業沒東西可以寫，這次作業先討論紅框部份，紅箭頭的部份是刪除一層 Dense。

1.)將第一層的卷積加大成 9X9 的 Mask，因為圖片僅有黑白兩種顏色且只有 1 個 channel，因此不需要小的 Mask 去抓取特徵，提高效率；第二層則維持不變，提高分辨率。

2.)修改第二次防止過擬合的 Dropout 至 0.25，前面已 Dropout 一次，模型非常簡單且只有黑白兩色，理論上 Dropout 可以拿掉，但此資料量不少，因此斟酌改成 0.25。

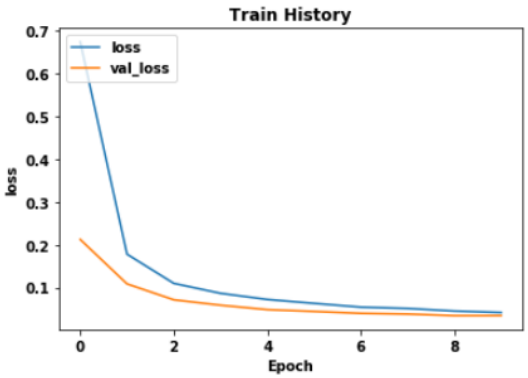
3.)optimizer 由 Adadelta 修改至 Adam，Adam 對於 local min. 有極佳的效果，收斂速度雖然不是最快，但是最穩定，也是現在最常使用的 optimizer。

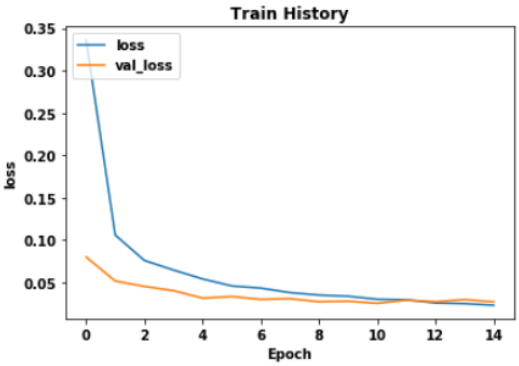
4.)由前面的經驗可以知道，在 epoch 12 以後已開始收斂，且 batch_size=1024 是瓶頸，預期更改後的模型效率較高，因參數量大幅減少，所以調整 epoch 至 10。

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 20, 20, 32)        2624
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 18, 18, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 9, 9, 64)          0
_________________________________________________________________
dropout_11 (Dropout)         (None, 9, 9, 64)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 5184)              0
_________________________________________________________________
dropout_12 (Dropout)         (None, 5184)              0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                51850
=================================================================
Total params: 72,970
Trainable params: 72,970
Non-trainable params: 0
_________________________________________________________________
```

(修改後模型架構，刪除一層全連接層)

```
60000/60000 [==============================] - 30s 499us/step - loss: 0.0417 - accuracy: 0.9872 - val_loss: 0.0344 - val_accura
cy: 0.9896
Test loss: 0.03439972898203414
Test accuracy: 0.9896000027656555
```





(修改後模型訓練 10th epoch 與右方修改前的模型 Train History)

先觀察修改後的模型，可以看到每個 epoch 的時間由一分鐘減少到半分鐘，快了一倍的時間，效率大幅提高，事實上在 4th epoch 時 loss 已下降到 0.0865，與原本的模型相比，雖然一開始的 Loss 較高，但由於動量關係，下降速度頗快，在 epoch 4 時兩者 loss 其實已差不多。

```
Epoch 4/10
60000/60000 [==============================] - 30s 497us/step - loss: 0.0865 - accuracy: 0.9747 - val_loss: 0.0586 - val_accura
cy: 0.9817
Epoch 5/10
60000/60000 [==============================] - 30s 501us/step - loss: 0.0721 - accuracy: 0.9785 - val_loss: 0.0483 - val_accura
cy: 0.9852
```

(修改後模型 4th、5th epoch)

```
Epoch 4/15
60000/60000 [==============================] - 63s 1ms/step - loss: 0.0645 - accuracy: 0.9809 - val_loss: 0.0402 - val_accurac
y: 0.9865
Epoch 5/15
60000/60000 [==============================] - 61s 1ms/step - loss: 0.0540 - accuracy: 0.9838 - val_loss: 0.0313 - val_accurac
```

(修改前模型 4th、5th epoch)

兩者的正確率、loss 差不多，因此這是一個非常成功的改良版模型，大幅提高了訓練效率。

以上是由 CPU 訓練的成果，接下來是使用 GTX 1066 做訓練。

```
Epoch 15/15
60000/60000 [==============================] - 5s 76us/step - loss: 0.0237 - accuracy: 0.9924 - val_loss: 0.0257 - val_accur
```

(未修改前模型)

```
Epoch 10/10
60000/60000 [==============================] - 3s 42us/step - loss: 0.0417 - accuracy: 0.9875 - val_loss: 0.0326 - val_accur
acy: 0.9891
```

(修改後的模型)

原先使用 CPU 需分別耗時 1 分鐘與 0.5 分鐘，使用 GPU 僅需 5 秒與 3 秒，使用 GPU 果然最有效率。明顯看到就算是使用 GPU，修改後的模型訓練時間也是未修改前的一半。

比較大的問題是前述紅字部份，既然拿掉了第一層 Dense，第二個 Dropout 的用處在哪？以及如果第一層 Dense 存在，那拿掉兩個 Dropout 是不是沒有太大影響？由於後者牽扯到模型架構的變動，這部份會留到第三次作業實作，先討論前者。因為原本還有一層 Dense 在中間，把 1/4 神經元關掉後壓成 128 個輸出再 Dropout 一次，這兩次 Dropout 因為中間加了一層 Dense 所以作用不同，但修改後的模型拿掉第一層 Dense 後其架構為 Dropout(0.25)->Flatten()->Dropout(0.25)，其實和 Flatten()->Dropout(7/16)道理相同，寫二層 Dropout 實屬多餘，這部份是在後續檢查時發現的問題，當初修改 model 時沒有注意到，因此特別拿出來討論；若使用相同的數據庫，而僅僅是增加卷積層(保留原本的第一層 Dense)，理論上 Dropout 影響會比較明顯，因為抓取的特徵更細微，丟掉這些特徵會影響前幾個 epoch 判讀，但此次圖片單純應不會有甚麼差別，對於 RGB 圖片較敏感；Dense 的去留則於下次作業的問題與討論提出。

本次實作 CNN 的 Hello World，雖然是很大眾的資料庫，但也是很重要的學習經驗，因為資料單純，更容易著手思考該怎麼處理，網路如何建立，參數的挑選都是需要一步步的累積。此手寫資料庫中其實有許多連人類都難以辨識的圖片，假設人類對此圖片認知為 9，但機器認為是 4，在人類不確定的狀況下如何判斷機器是錯的？當圖片辨識的境界到了一個高度，就會產生如此的問題，因此在討論正確率 99%附近時，無法斷定這個模型是否已完美，又或是需要再改良，整個資料庫有六萬張圖片，裡面不乏可以辨識但標錯的，當然，我們也不可能再去一張一張檢查。

Code link：

Training by CPU

Training by GPU