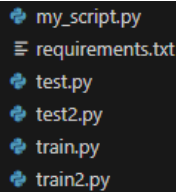


# HW4

612415013 蕭宥羽

## 1. How to execute codes.

- ✚ `my_script.py` 腳本可以對程式設置不同的參數，可以一次順序執行不同的參數設定(使用 `test2.py` `train2.py`)



```
epochs = 20
learning_rate = 0.01
img3 = os.path.join(base_path, "result", "Change_learning_rate", "0_01")
weight3 = os.path.join(base_path, "weights", "Change_learning_rate", "0_01", "weight.pth")
command = f"python {train_script_path} --epochs {epochs} --learning_rate {learning_rate} --img_path {img3} --weight_path {weight3}"
test = f"python {test_script_path} --img_path {img3} --weight_path {weight3}"
subprocess.run(command, shell=True)
subprocess.run(test, shell=True)

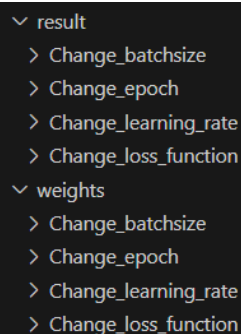
epochs = 20
learning_rate = 0.001
img4 = os.path.join(base_path, "result", "Change_learning_rate", "0_001")
weight4 = os.path.join(base_path, "weights", "Change_learning_rate", "0_001", "weight.pth")
command = f"python {train_script_path} --epochs {epochs} --learning_rate {learning_rate} --img_path {img4} --weight_path {weight4}"
test = f"python {test_script_path} --img_path {img4} --weight_path {weight4}"
subprocess.run(command, shell=True)
subprocess.run(test, shell=True)
```

- ✚ 使 `python my_script.py` 便可以執行程式
- ✚ 使用 `pytorch` 中的 `lossfunction` 會出現 `tensor` 的錯誤，對 `target` 做 `one-hot` 的處理，避免這些錯誤

```
one_hot = torch.zeros((target.shape[0],2)).to(device)
one_hot[target==0]=torch.Tensor([1,0]).to(device)
one_hot[target==1]=torch.Tensor([0,1]).to(device)

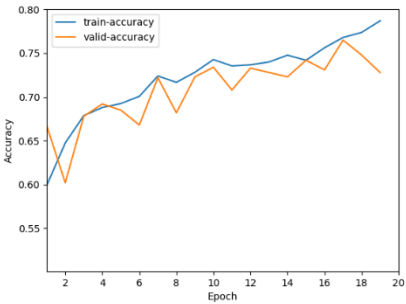
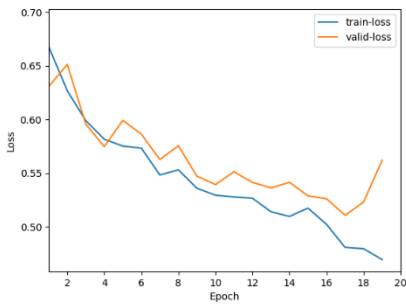
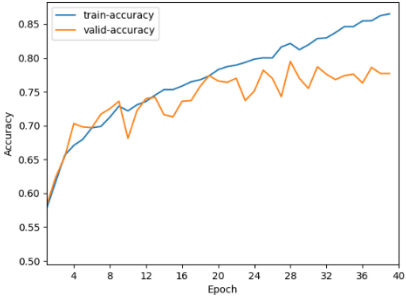
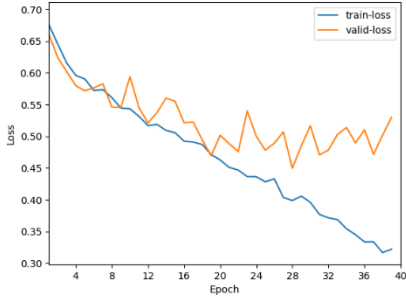
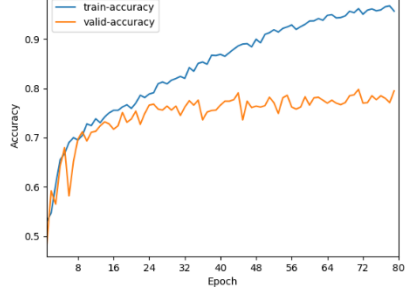
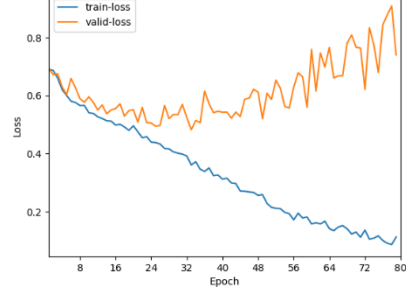
# forward + backward + optimize
output = model(data)
_, preds = torch.max(output.data, 1)
loss = criterion(output, one_hot)
```

- ✚ 執行完後會將結果與權重存下來，接著自動進行測試，將測試的準確率寫入.txt 中存下來

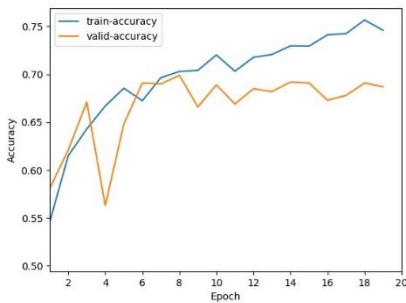
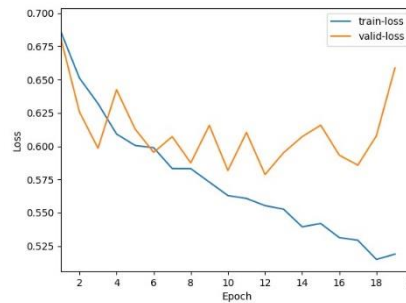


## 2. Experimental results

### Epoch 調整

Epoch	Training accuracy	Loss curve	Test acc
20			74.8%
40			76.1%
80			77.6%

### Batch size 調整

Batchsize	Training accuracy	Loss curve	Test acc
8			65.4%

16			71.1%
32			71.8%



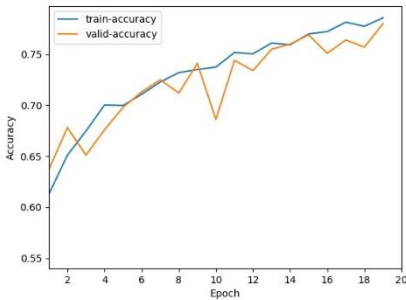
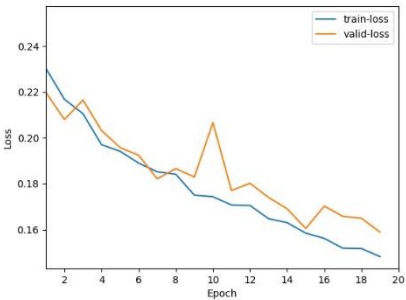
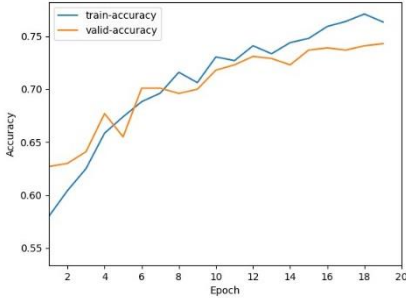
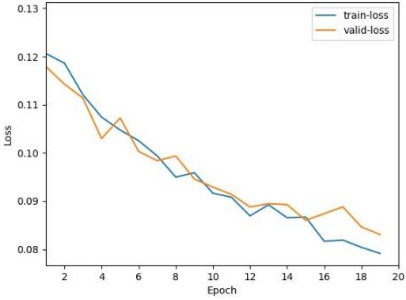
## Learning rate 調整

Learning rate	Training accuracy	Loss curve	Test acc
0.001			69.8%
0.01			73.6%

0.1			56.0%
-----	--	--	-------

### Loss function 調整

Loss function	Training accuracy	Loss curve	Test acc
Cross Entropy Loss			73.9%
FocalLoss			74.3%
L1Loss			73.4%

MSELoss			75.4%
Smooth L1Loss			73.4%

### 3. Conclusion

#### ✚ Epoch 調整

從 loss 曲線觀察，可以看到在 epoch 40 與 80，訓練損失 (train-loss) 下降並趨於穩定，達到了非常低的水平。然而，驗證損失 (valid-loss) 在這段時間內卻呈現明顯的上升趨勢。這種情況通常暗示著模型出現了過度擬合 (overfitting) 的問題，即模型在訓練集上表現良好，但在未見過的驗證集上表現較差。

就準確率 (accuracy) 曲線而言，訓練準確率 (train-acc) 隨著 epoch 次數的增加而逐漸提高，呈現出越來越好的趨勢。然而，驗證準確率 (valid-acc) 雖然沒有明顯的下降，但也無法進一步提升。這種情況表明模型在訓練集上學習到了特定的模式和特徵，但這些特徵可能對未見過的數據並不具有泛化能力，導致驗證集上的性能無法得到有效改善。

#### ✚ Batch size 調整

根據批次大小 (batch size) 的增加，觀察到曲線的波動逐漸減小。這種現象主要是由於訓練穩定性的提高所致。較小的批次大小可能導致訓練過程更加不穩定，因為每次權重更新所使用的樣本數量較少，梯度可能會出現較大的波動。當梯度波動大時，模型在學習過程中容易受到噪聲的干擾，進而影響模型的收斂情況。

相反地，較大的批次大小可以減少這種波動，使訓練過程更加穩定。因為每個批次包含更多的樣本，這意味著每次更新的梯度更具代表性，能夠更好地反映整個數據集的趨勢，從而減少了梯度的變動程度。這種穩定性有助於模型更有效地學習數據的特徵和模式，進而提高了訓練的效率和準確性。而我們從 test accuracy 可以看出這種趨勢。

#### ✚ learning 調整

根據表格中的結果顯示，學習率為 0.01 時，模型達到了最佳的性能。而學習率為 0.001 時，訓練的效果

較差，可能因為學習率過小而導致訓練過程過於緩慢，無法有效地收斂到最優解。而當學習率設置為 0.1 時，導致模型震盪無法收斂。所以根據實驗顯示，過小的學習率可能導致訓練過程過於緩慢，難以收斂到最優解；而過大的學習率可能導致優化算法無法有效地收斂，或者在最優解附近產生振盪。

#### loss function 調整

##### 1. Cross Entropy Loss (交叉熵損失):

交叉熵通常用於多類別分類問題，尤其是當目標是 one-hot 編碼形式時。它衡量了模型預測與實際標籤之間的差異，並且將標籤的分佈概率與預測的概率進行比較。數學公式如下：

$$L_{CE} = - \sum_{i=1}^C y_i \log(p_i)$$

##### 2. Focal Loss (焦點損失):

Focal Loss 是為了應對類別不平衡問題而提出的一種損失函數。它在標準交叉熵損失的基礎上，對易分類樣本的損失進行了降低，從而提高了對困難樣本的關注度。數學公式如下：

$$L_{FL} = - \sum_{i=1}^C (1 - p_i)^\gamma y_i \log(p_i)$$

##### 3. L1 Loss (L1 損失):

L1 損失是回歸問題中常用的一種損失函數。它衡量了模型預測值與實際目標值之間的差異的絕對值。使用 L1 損失可以使模型更加關注那些較大的誤差。數學公式如下：

$$L_{L1} = \sum_{i=1}^N |y_i - p_i|$$

##### 4. MSE Loss (均方誤差損失):

均方誤差損失也是回歸問題中常用的一種損失函數。它衡量了模型預測值與實際目標值之間的差異的平方。與 L1 損失相比，它對大誤差的懲罰更重。數學公式如下：

$$L_{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - p_i)^2$$

##### 5. Smooth L1 Loss (平滑 L1 損失):

平滑 L1 損失是均方誤差損失和 L1 損失之間的一種折衷。它在離原點較近時使用 L1 損失，從而減少對小誤差的懲罰，而在離原點較遠時則使用均方誤差損失，從而減少對大誤差的懲罰。

不同的損失函數可以導致曲線上的波動和形狀的變化，這可能會影響模型的收斂速度和最終性能。

交叉熵損失通常會在訓練初期迅速下降，因為它對模型預測的自信度和標籤之間的差異有較高的懲罰。這可能會導致損失函數曲線快速收斂。正確率曲線可能會與損失函數曲線呈現相似的趨勢。焦點損失在一定程度上可以改善類別不平衡問題，因此損失函數曲線可能會比交叉熵損失曲線更加平滑。這可能會影響模型的收斂速度和訓練穩定性。L1 損失和均方誤差損失通常用於回歸問題，對於分類問題可能會導致損失函數曲線的不穩定。如果誤差值較大，則損失函數曲線可能會呈現較大的波動。平滑 L1 損失是一種折衷方案，可以在一定程度上平衡 L1 損失和均方誤差損失之間的差異。這可能會導致損失函數曲線的平滑性提高，從而改善模型的收斂速度和訓練穩定性。

根據理論而言，Cross Entropy Loss 通常被認為是處理分類問題的首選損失函數，因其能夠有效地衡量模型預測與實際標籤之間的差異。然而，有時在實際測試中，我們觀察到最高的測試準確率是使用 MSELoss 的情況。所以我會使用 MSELoss。

## 4. Discussion

這次的實驗給我帶來了一些挑戰，因為我不僅需要測試不同的參數，還要在實驗中考慮不同的損失函數。

為了提高效率，我決定撰寫一個自動化腳本來執行這些實驗，並將結果保存為.txt 文件。這個過程需要仔細地考慮腳本的邏輯和實現細節，花了我相當多的時間，但最終使得實驗的執行更加方便和高效。

在實驗過程中，我也遇到了一些意外，特別是當我嘗試使用不同的損失函數時，PyTorch 中的 tensor 錯誤讓我頭痛不已。為了解決這個問題，我深入研究了損失函數的計算方式，並採用了對目標值進行 one-hot 編碼的方法。使得我能夠順利地進行實驗，獲得了最終的結果。