

深度第一次作業

目錄

- 1 實驗目的
- 2 資料集簡介
- 3 資料前處理方式
- 4 方法
- 5 實驗結果
- 6 結論

實驗目的

實驗本資料集對於分類行為下，不同參數對普通Multilayer perceptron(MLP,多層感知器)所造成之影響，以及和eXtreme Gradient Boosting (XGboost)，在準確率，精確性，召回率以及F1-score 等評估指標的評價

資料集及分類目的

資料集

Predict students' dropout and academic

success<https://www.kaggle.com/datasets/thedevastator/higher-education-predictors-of-student-retention/code>

資料比數:4424是一份由34+1個欄位所構成

大致欄位有

狀況、申請方式、申請順序、課程、就讀時段（白天／夜間）、
先前學歷、國籍、母親學歷、父親學歷等等

分類任務是依照Target分成3類分別是Graduate、Dropout、Enrolled(畢業、退學、在學)

各類別 筆數:比例

Graduate	2209:0.4993
Dropout	1421:0.3212
Enrolled	7940:1795

資訊獲利 (Information Gain)

整體熵 = 1.4713

Curricular units 2nd sem (grade) 0.5150

Curricular units 2nd sem (approved) 0.4560

Curricular units 1st sem (grade) 0.4539

Curricular units 1st sem (approved) 0.3563

Curricular units 2nd sem (evaluations) 0.1391

Tuition fees up to date 0.1325

Curricular units 1st sem (evaluations) 0.1302

Course 0.0959

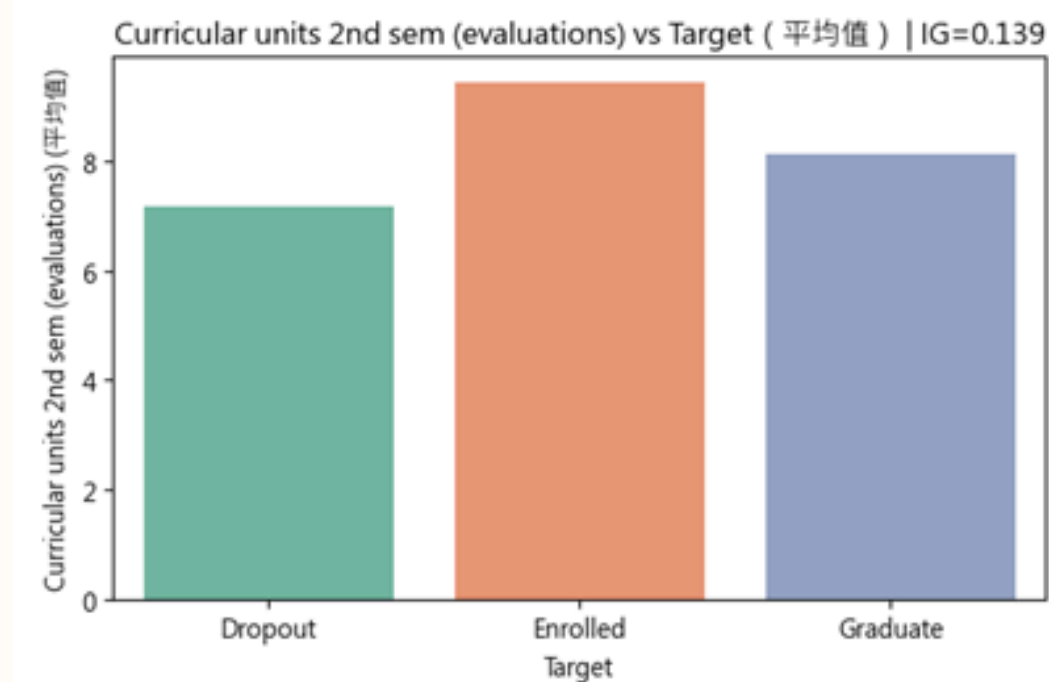
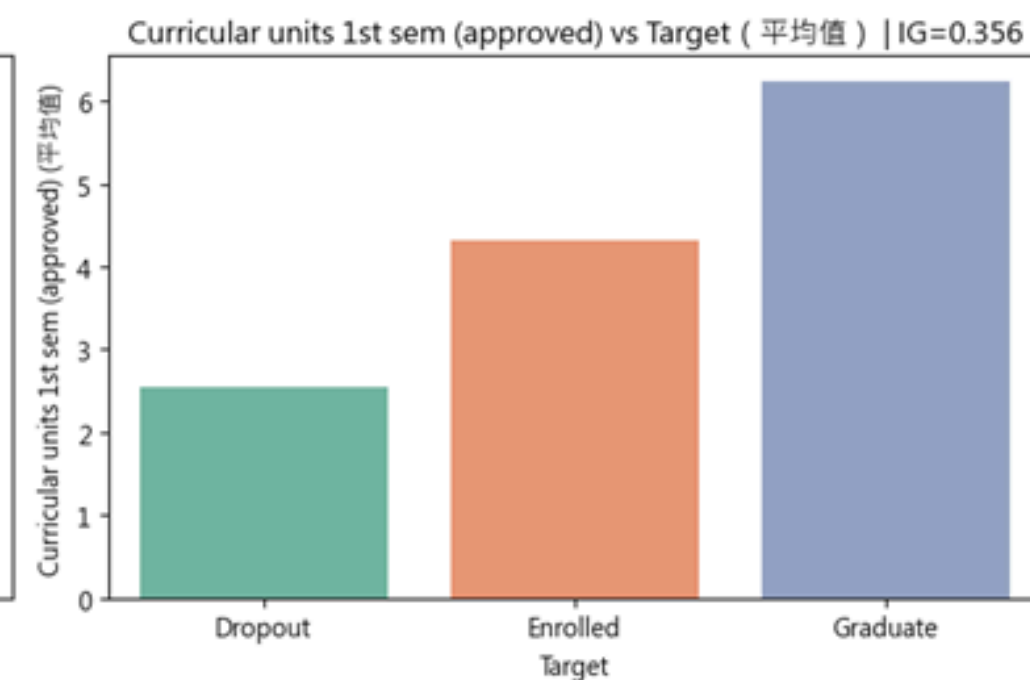
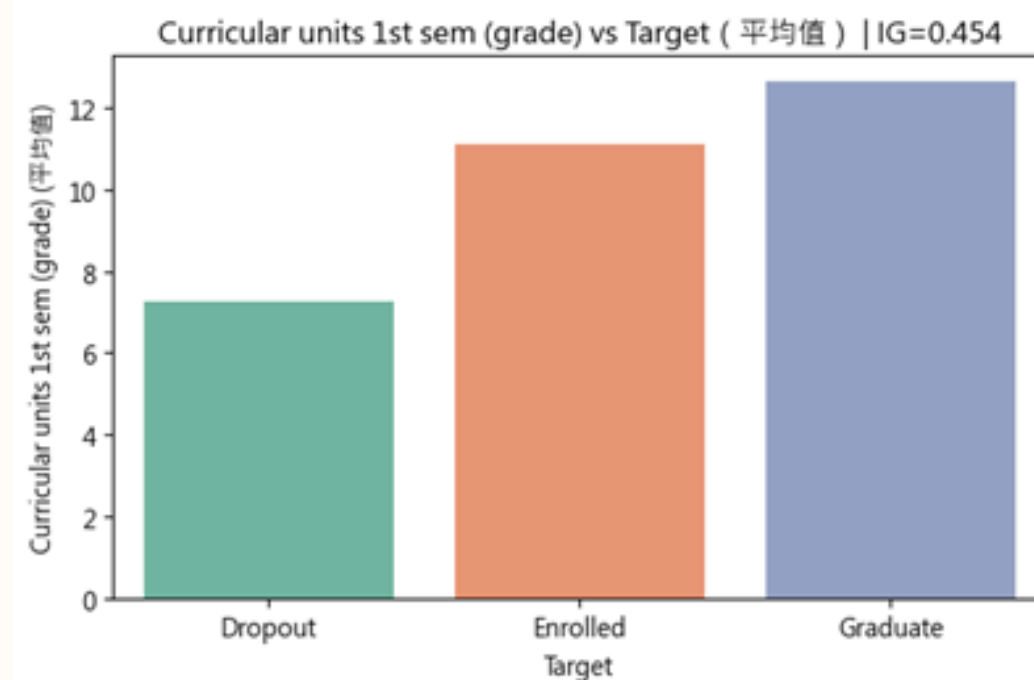
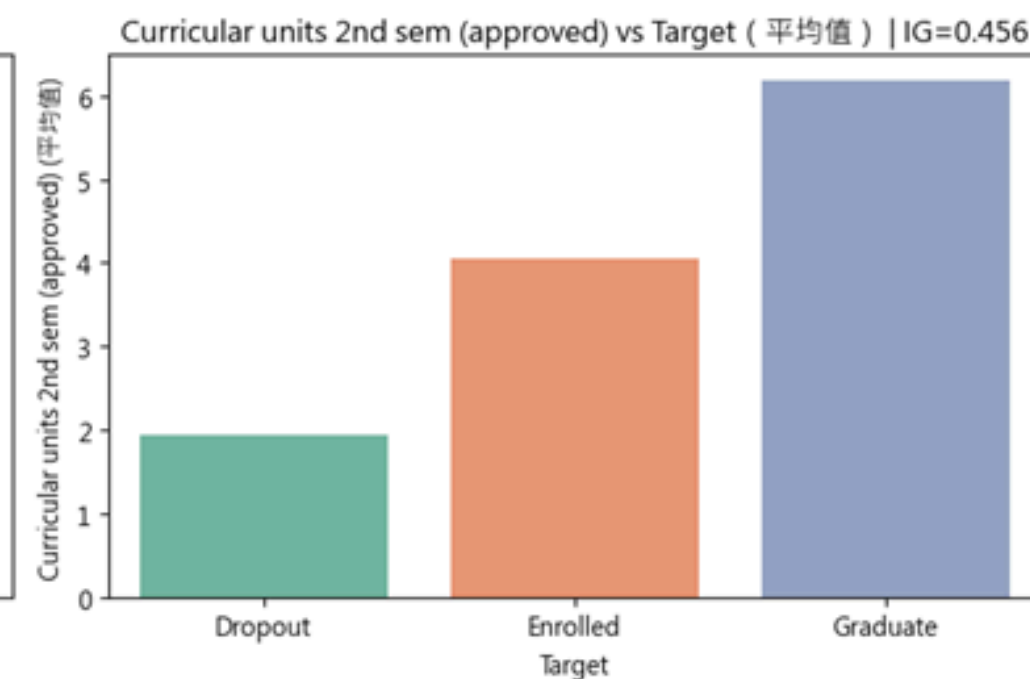
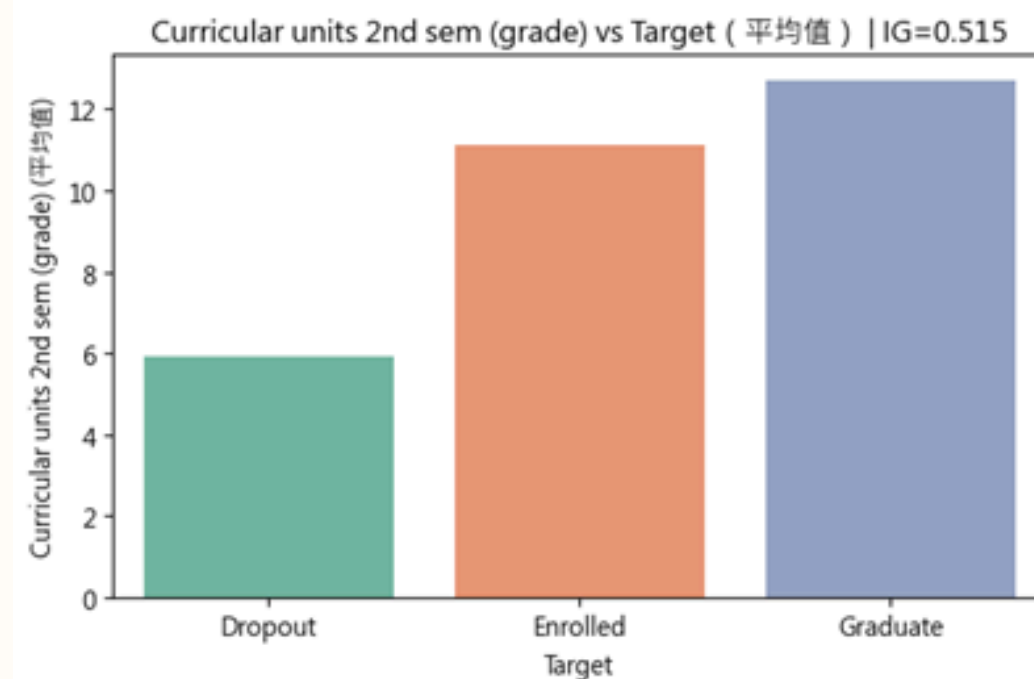
Age at enrollment 0.0895

Application mode 0.0753

Curricular units 2nd sem (enrolled) 0.0727

Curricular units 1st sem (enrolled) 0.0725

IG 值前5名平均值 與Target關係



資料前處理方式

Target設為目標欄位，並把Target 轉為類別

數值資料標準化（每個數值欄位變成均值 0、標準差 1）

使用StandardScaler()

對類別資料做One-Hot Encoding(熱編碼)

合併成單一特徵矩陣

對Target資料做LabelEncoding(0到k-1)

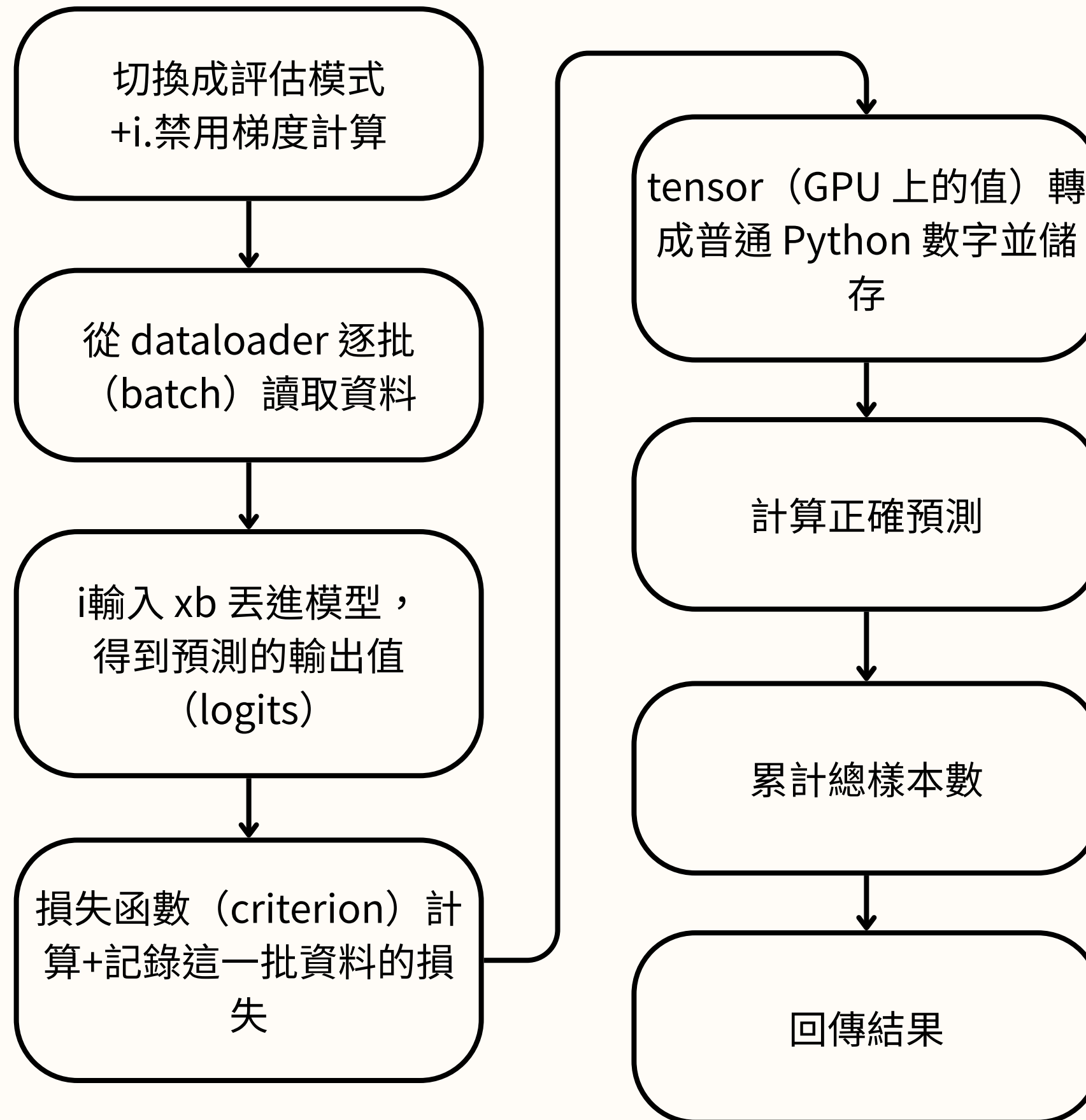
方法

對處理好的資料作資料切分，

- 訓練:驗證:測試->70:15:14
- 筆數為-> 3096:664:664

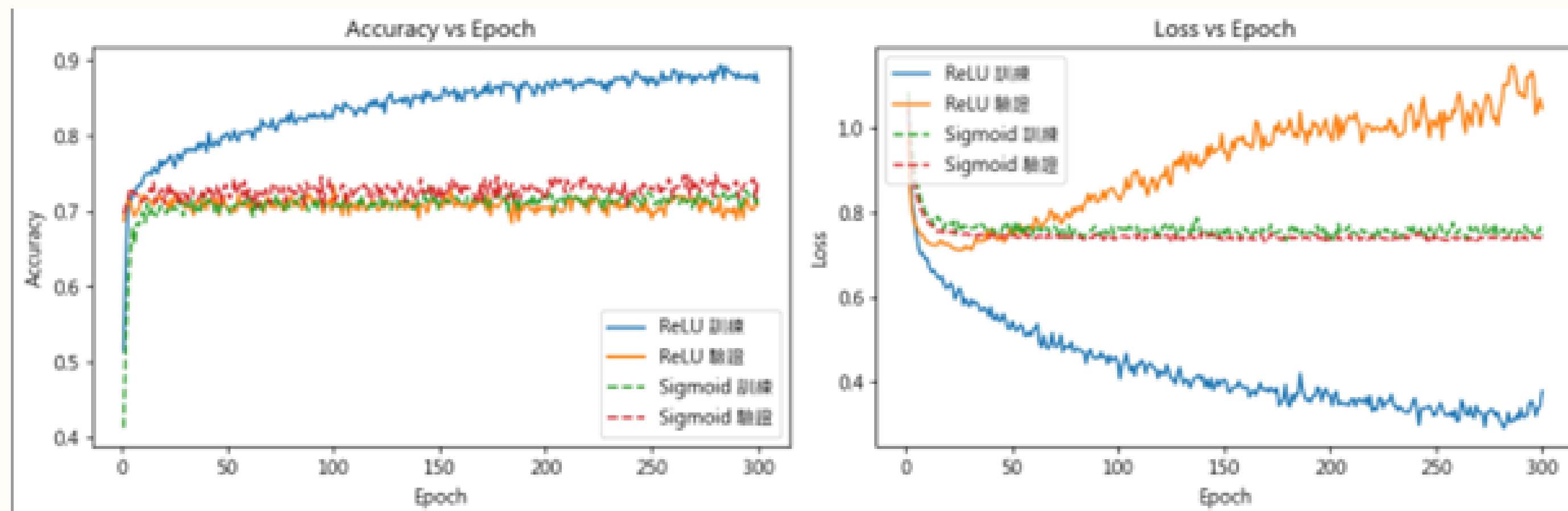
轉成numpy 在打包成PyTorch Dataset /
Dataloader

建立loss函數

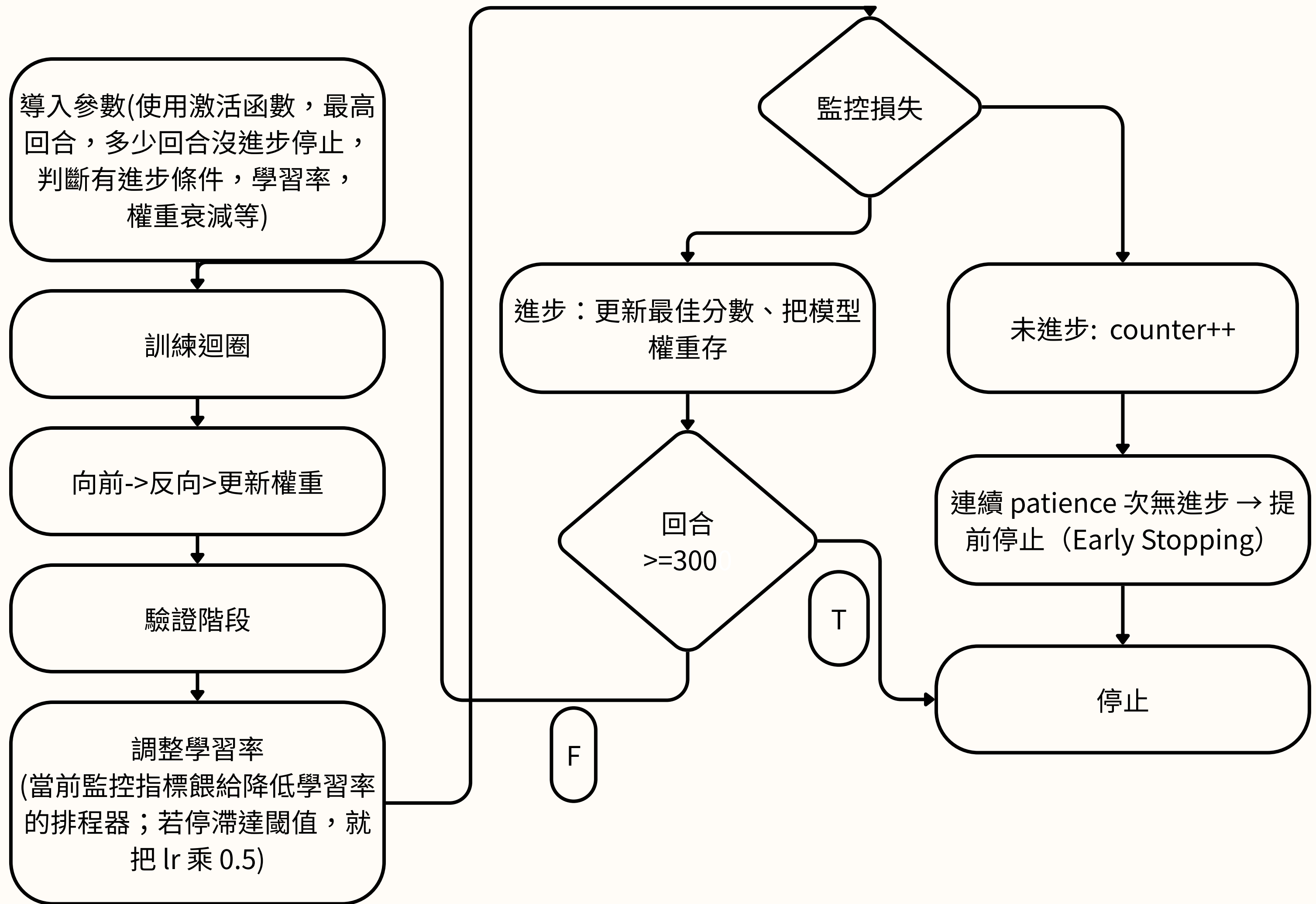


模型訓練

固定多次訓練relu容易過擬和



模型訓練



實驗設計

1. 隱藏層數(2、5)
2. 訓練 Batch size(256、1024)
3. 初始學習率(0.001、0.0003)
4. 激活函數(Relu、LeakyRelu(0.01))

以及另外訓練XGBoost作為比較

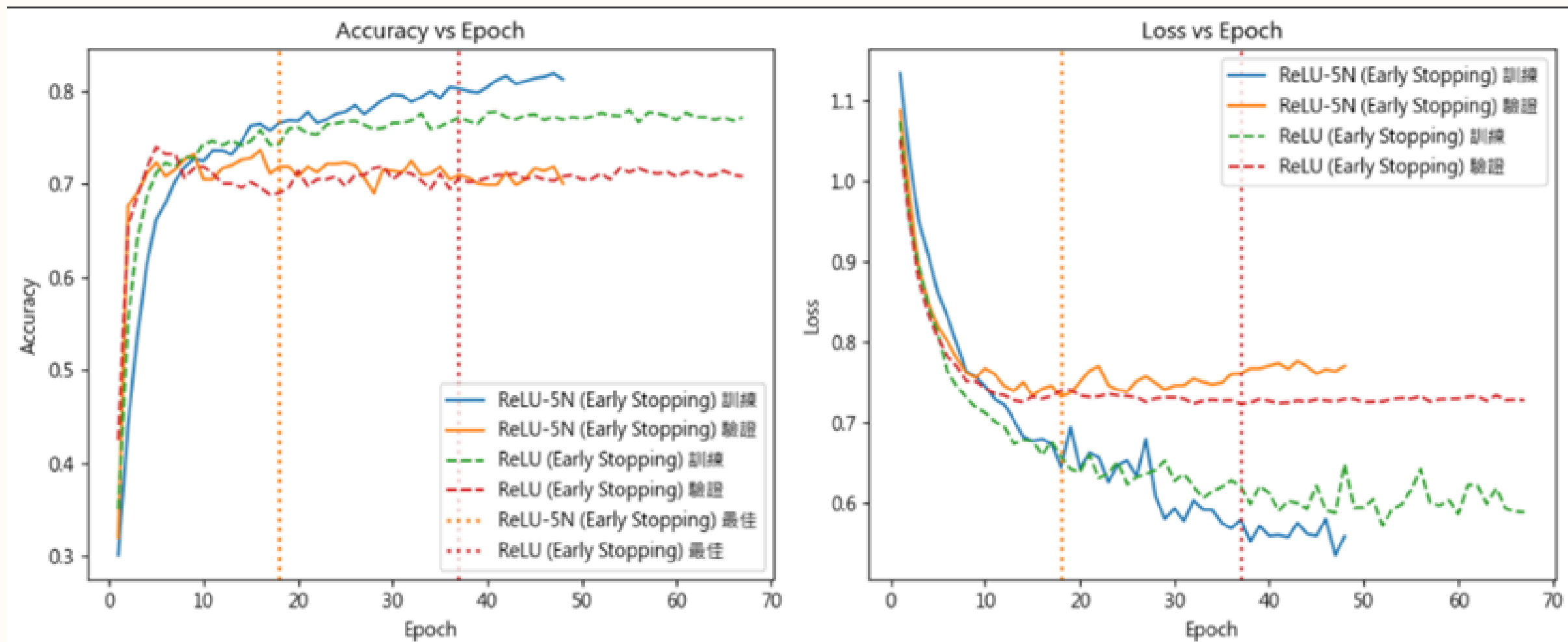
使用超參數

```
n_estimators=200,    # 樹的數量
max_depth=6,         # 每棵樹的最大深度 → 控制模型複雜度
learning_rate=0.05,  # 學習率 (eta) → 每次樹貢獻的權重)
subsample=0.8,       # 每次建樹時隨機抽取 80% 訓練樣本
colsample_bytree=0.8, # 每棵樹只使用 80% 的特徵 (特徵隨機化)
random_state=42,     # 固定隨機種子
```

結果

隱藏層數(2、5)

訓練



=== ReLU-5N (Early Stopping) 模型最終結果 ===

Train Loss: 0.5576

Val Loss: 0.7698

Train Acc: 0.8123

Val Acc: 0.7003

=== ReLU (Early Stopping) 模型最終結果 ===

Train Loss: 0.5878

Val Loss: 0.7275

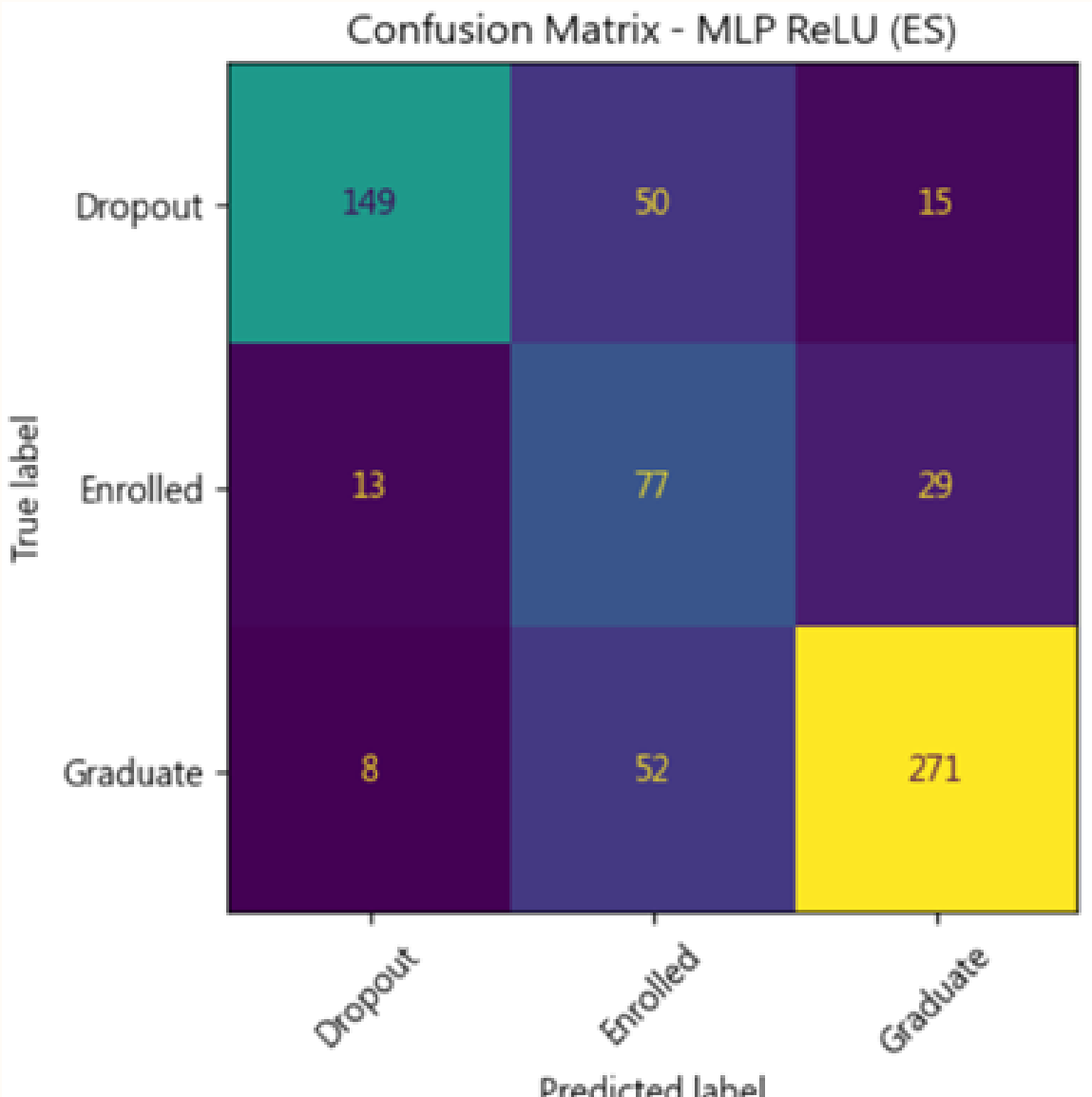
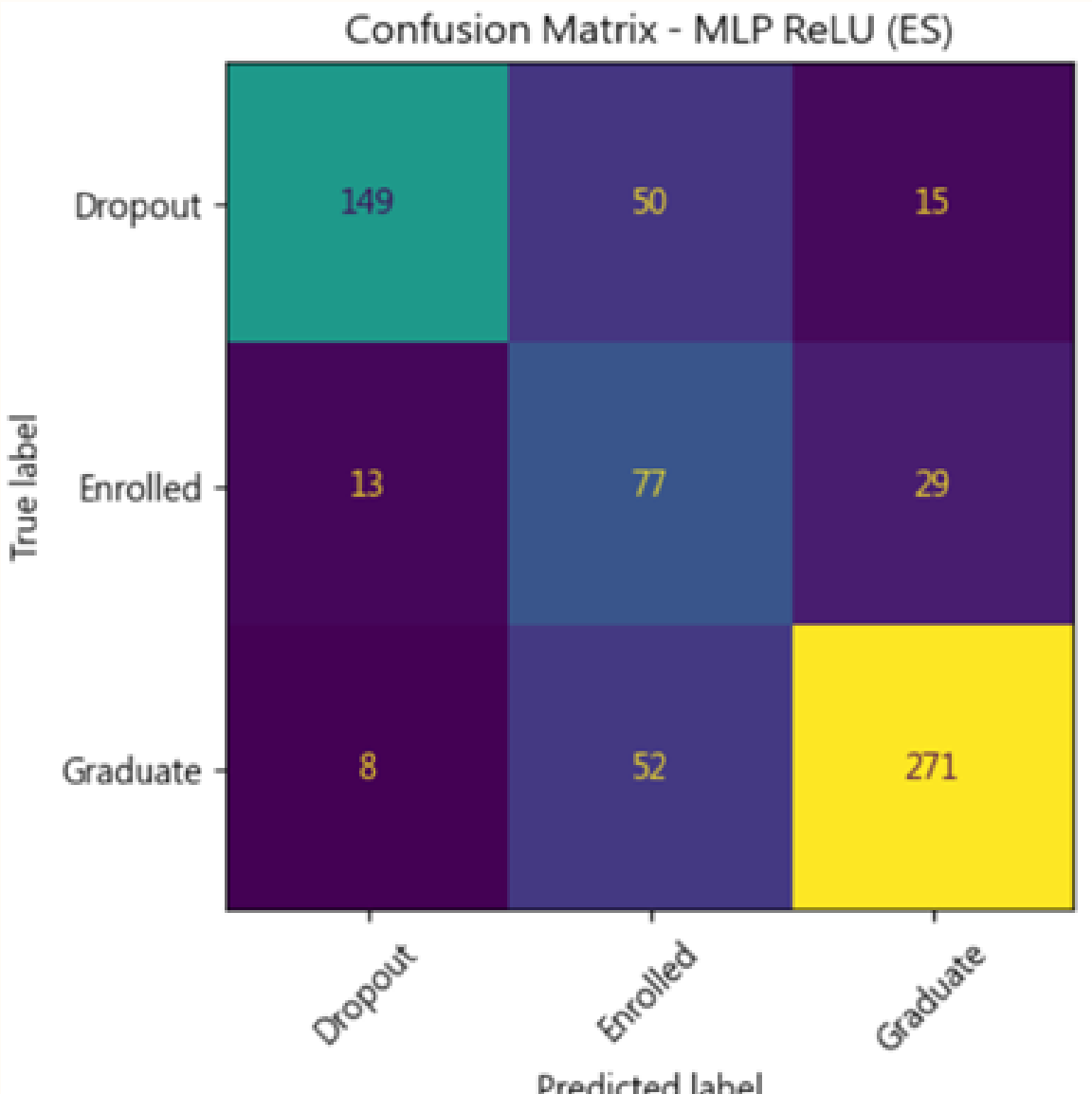
Train Acc: 0.7713

Val Acc: 0.7078

測試

Accuracy 、 Precision 、 Recall 、 F1

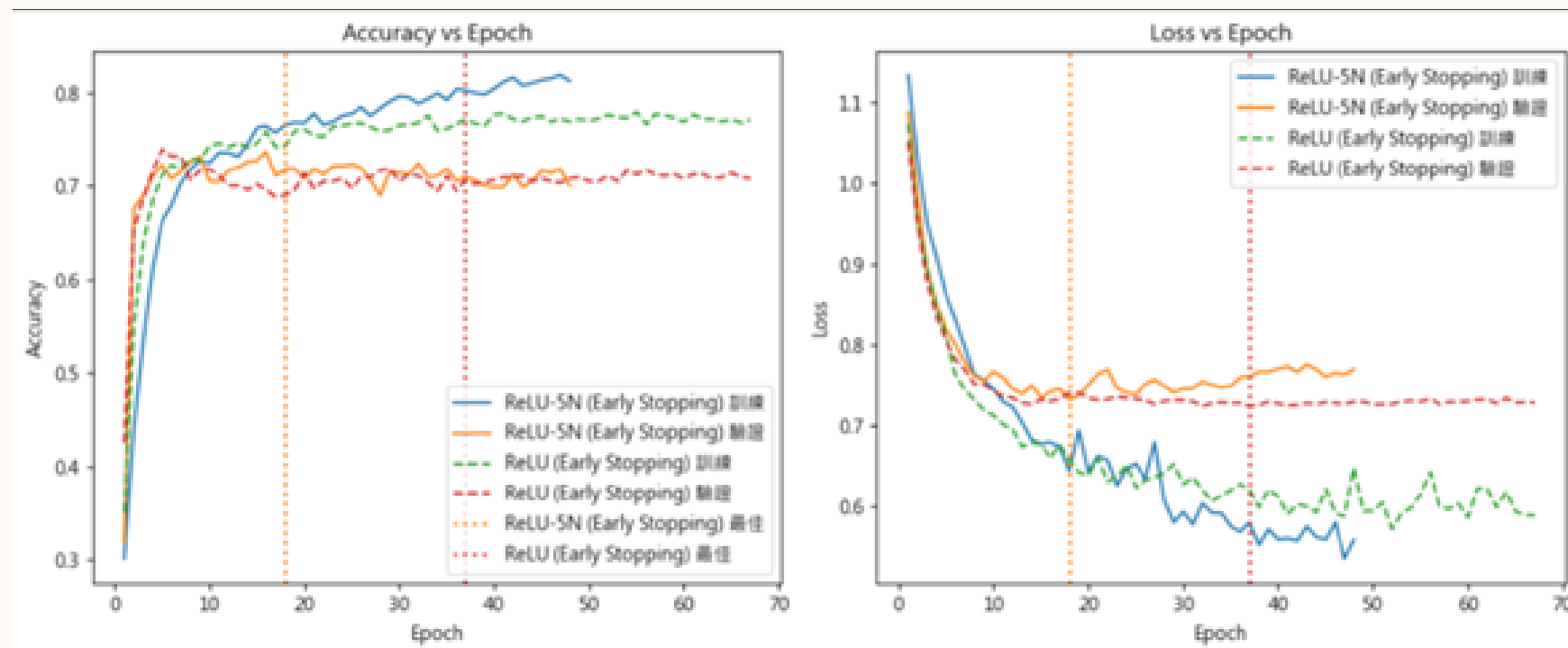
2	MLP ReLU (ES)	0.748494	0.788433	0.748494	0.760966
3	MLP ReLU-5N (ES)	0.745482	0.797678	0.745482	0.761025



訓練 Batch size

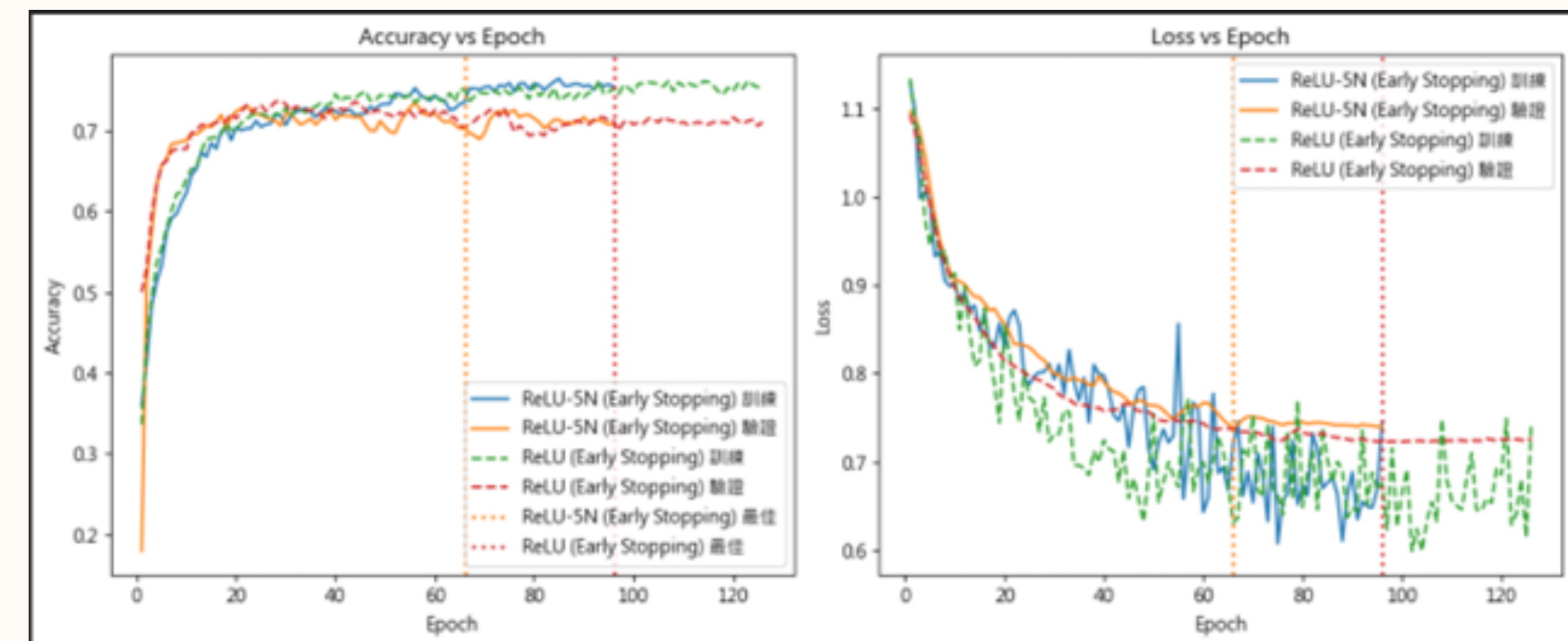
訓練、驗證

Batch size=256



```
=== ReLU-SN (Early Stopping) 模型最終結果 ===  
Train Loss: 0.5576  
Val Loss: 0.7698  
Train Acc: 0.8123  
Val Acc: 0.7003  
  
=== ReLU (Early Stopping) 模型最終結果 ===  
Train Loss: 0.5878  
Val Loss: 0.7275  
Train Acc: 0.7713  
Val Acc: 0.7078
```

Batch size=1024

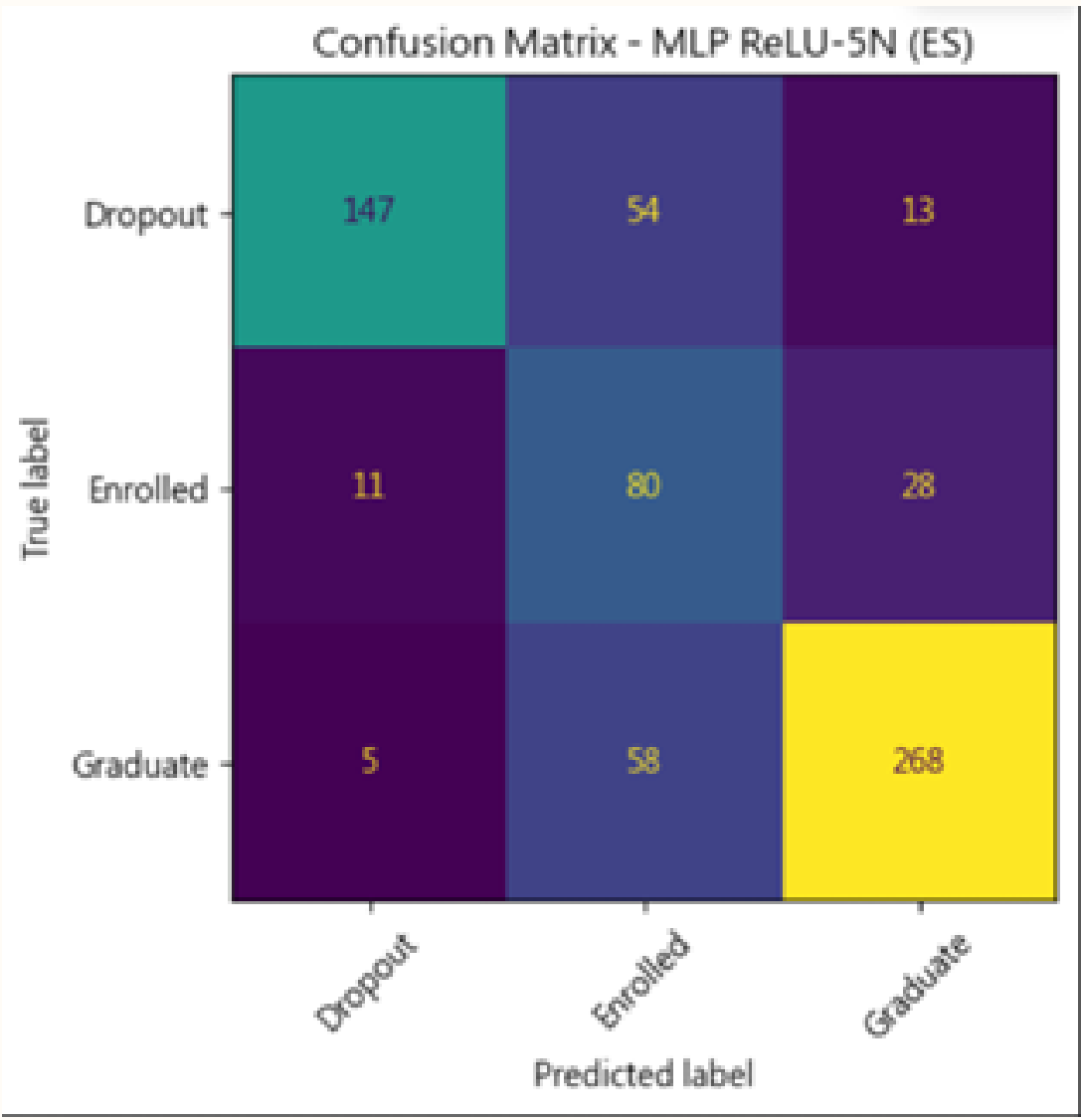
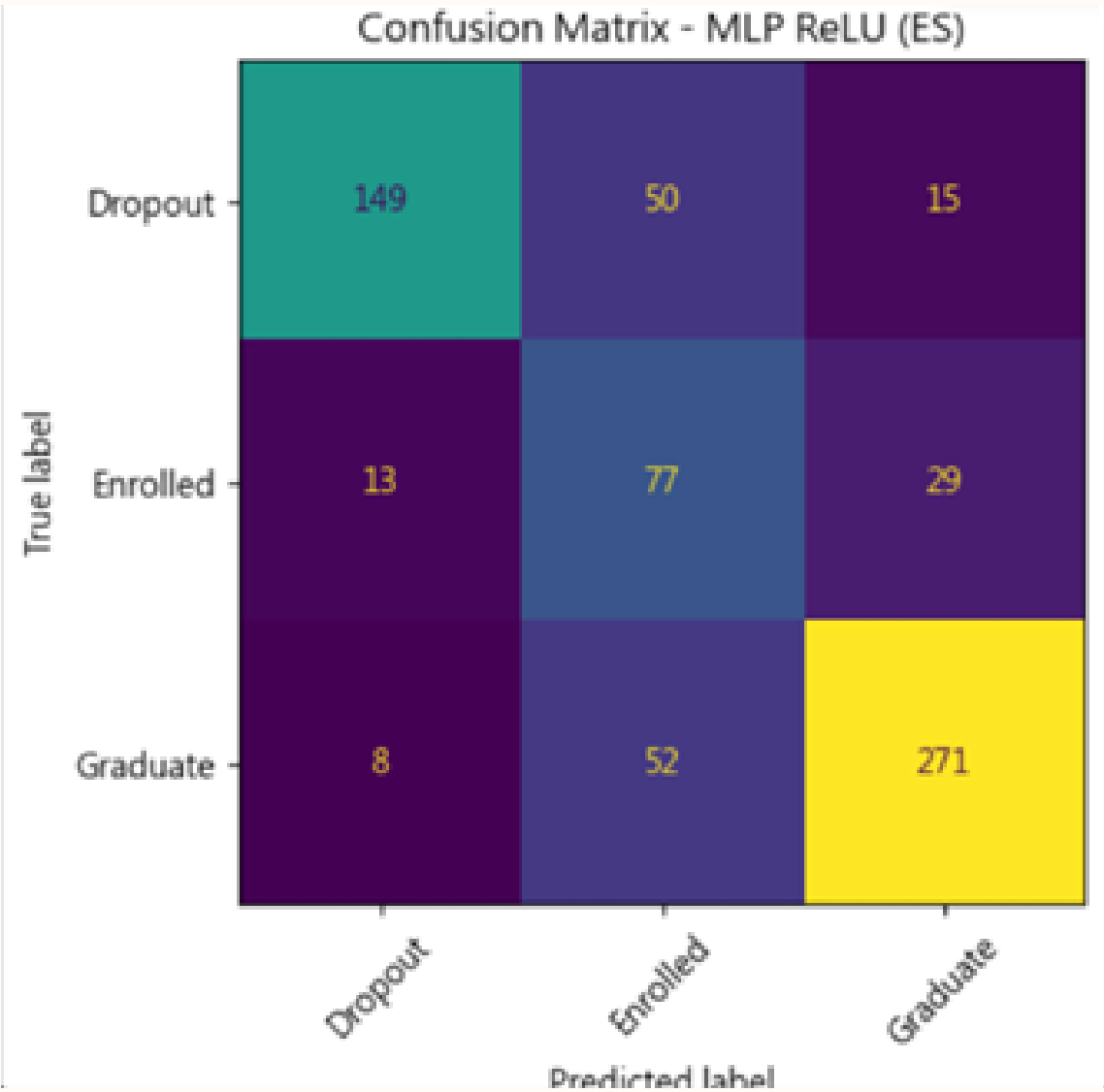


```
=== ReLU-SN (Early Stopping) 模型最終結果 ===  
Train Loss: 0.7450  
Val Loss: 0.7404  
Train Acc: 0.7545  
Val Acc: 0.7093  
  
=== ReLU (Early Stopping) 模型最終結果 ===  
Train Loss: 0.7418  
Val Loss: 0.7268  
Train Acc: 0.7526  
Val Acc: 0.7108
```

測試

Batch size=256

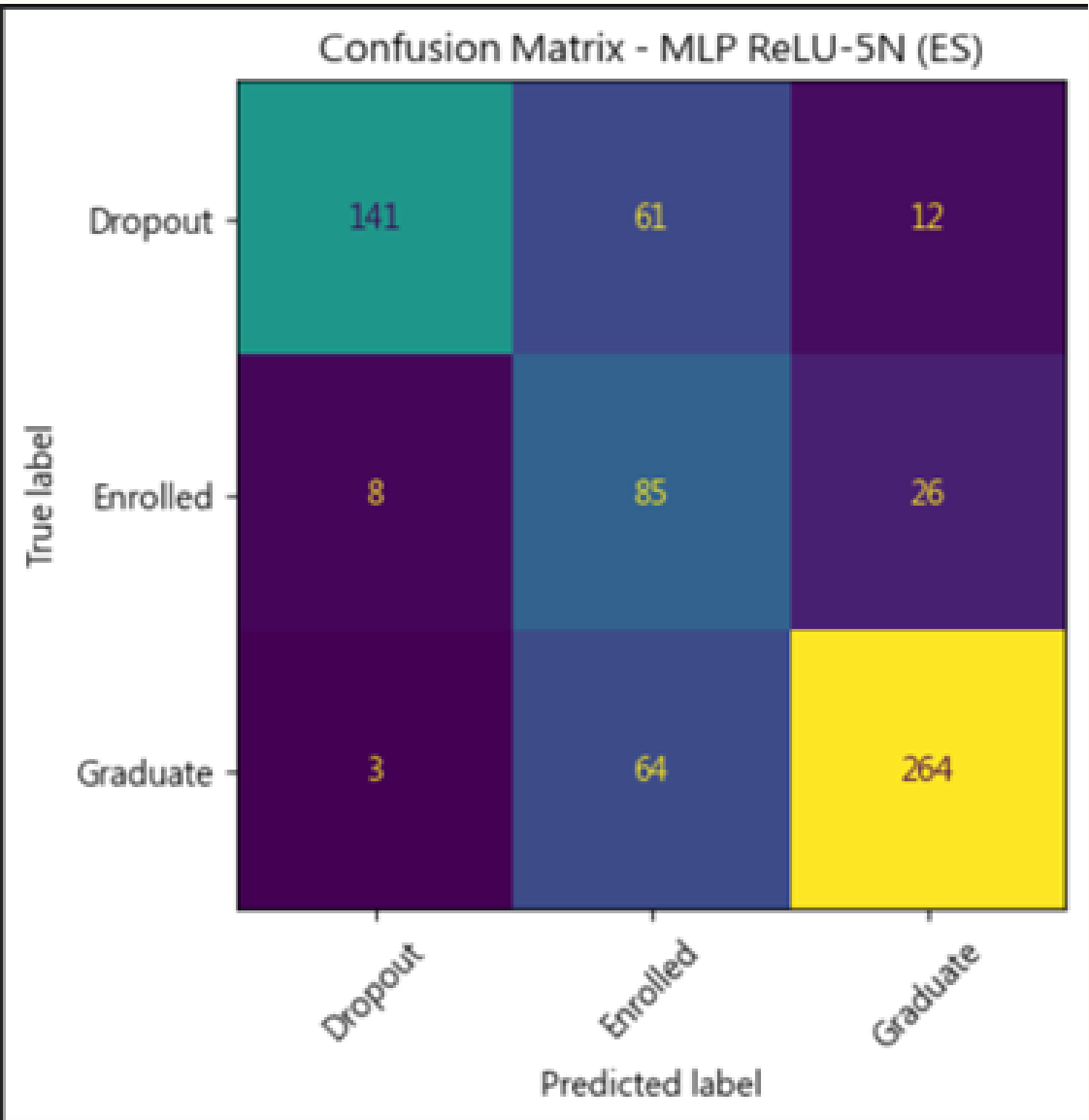
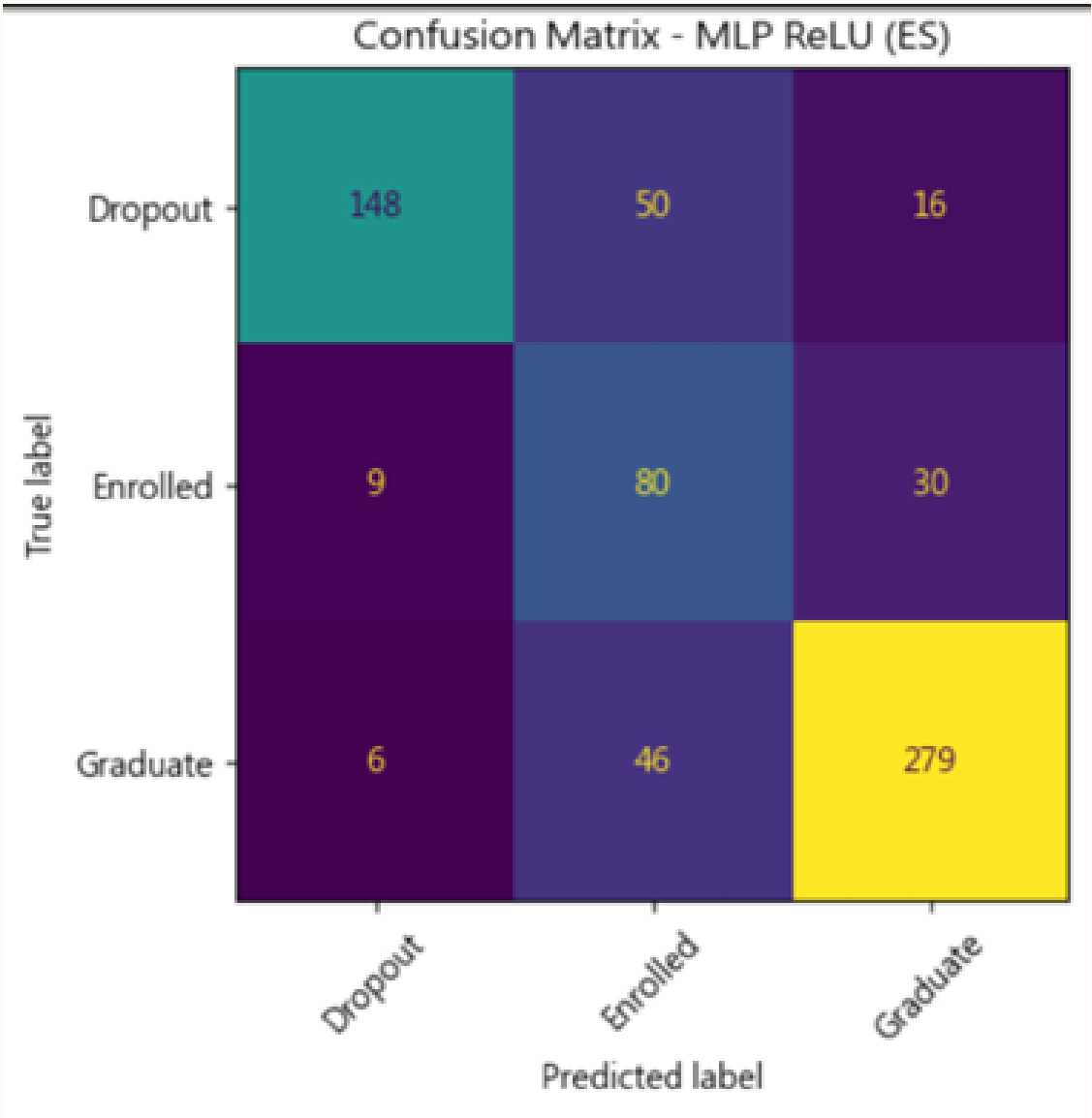
2	MLP ReLU (ES)	0.748494	0.788433	0.748494	0.760966
3	MLP ReLU-5N (ES)	0.745482	0.797678	0.745482	0.761025



測試

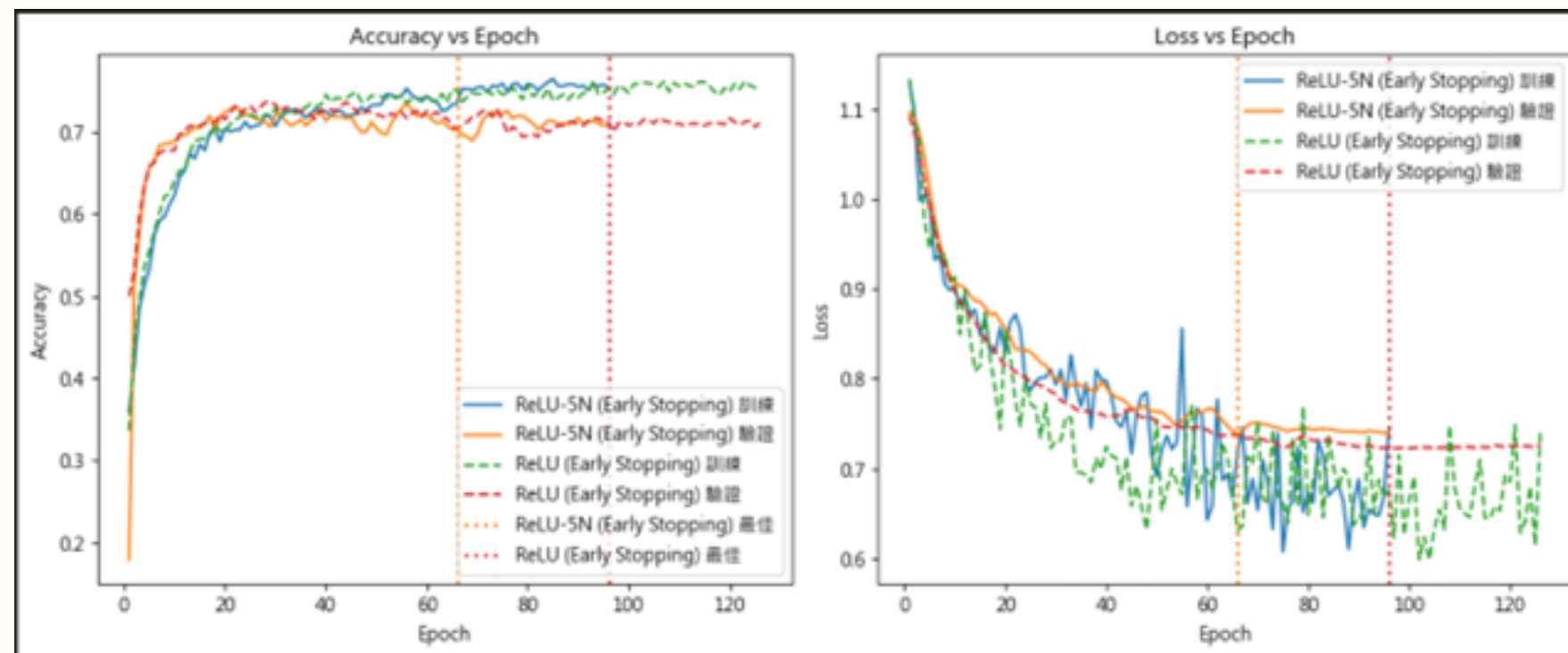
Batch size=1024

2	MLP ReLU (ES)	0.763554	0.802031	0.763554	0.774270
3	MLP ReLU-5N (ES)	0.737952	0.807275	0.737952	0.756731



1. 初始學習率

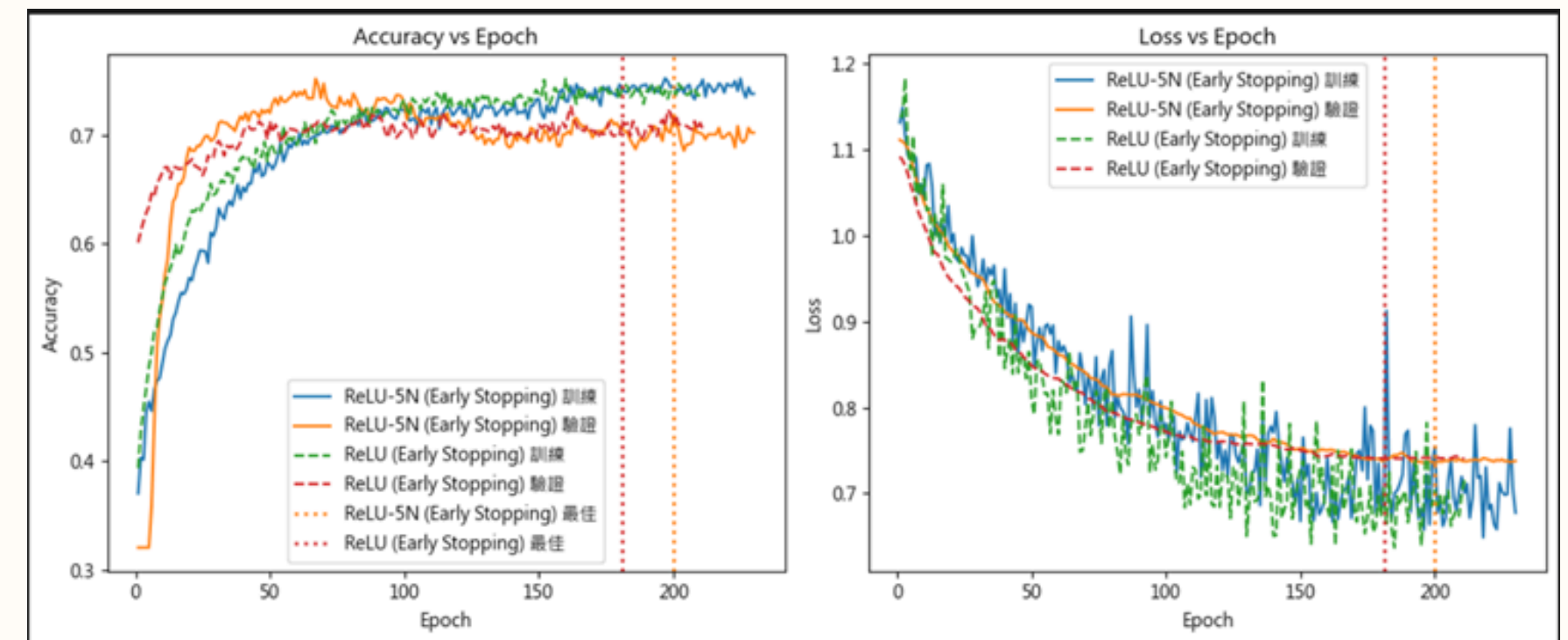
IR=1e-3(0.001)



```
=== ReLU-5N (Early Stopping) 模型最終結果 ===  
Train Loss: 0.7450  
Val Loss: 0.7404  
Train Acc: 0.7545  
Val Acc: 0.7093
```

```
=== ReLU (Early Stopping) 模型最終結果 ===  
Train Loss: 0.7418  
Val Loss: 0.7268  
Train Acc: 0.7526  
Val Acc: 0.7108
```

IR=3e-4(0.0003)



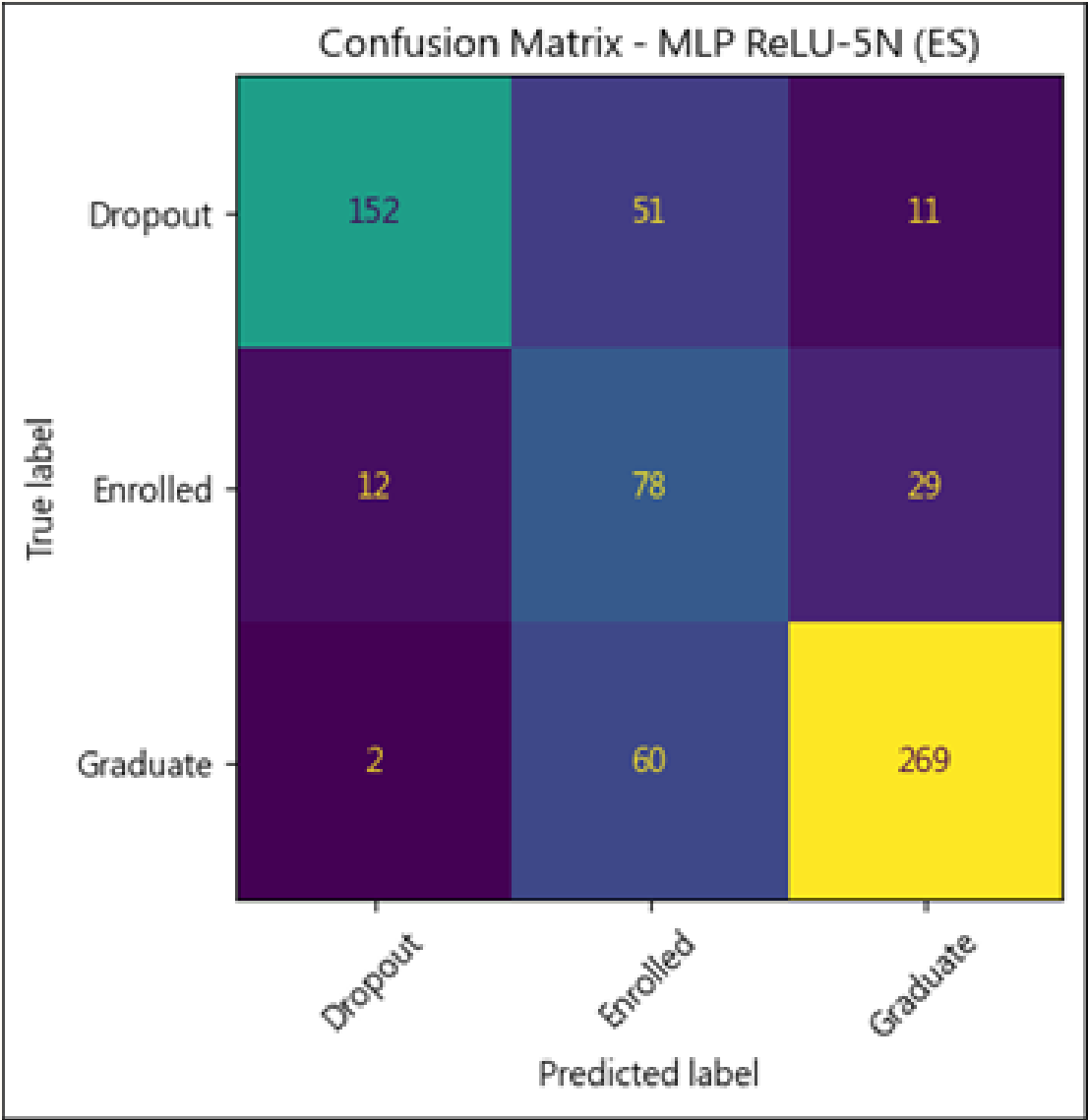
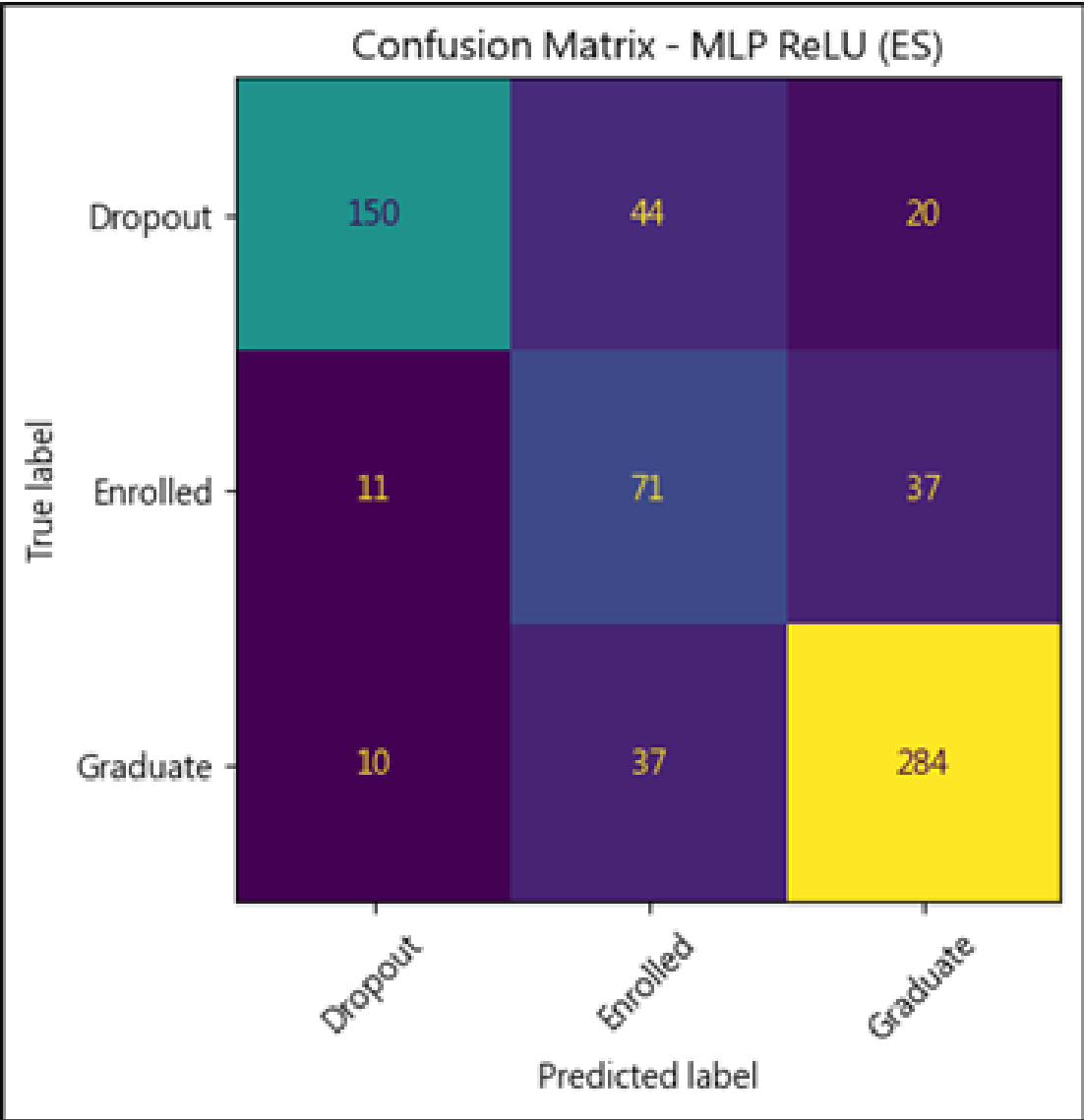
```
=== ReLU-5N (Early Stopping) 模型最終結果 ===  
Train Loss: 0.6777  
Val Loss: 0.7378  
Train Acc: 0.7377  
Val Acc: 0.7018
```

```
=== ReLU (Early Stopping) 模型最終結果 ===  
Train Loss: 0.7161  
Val Loss: 0.7406  
Train Acc: 0.7364  
Val Acc: 0.7048
```

測試

IR=1e-3(0.001)

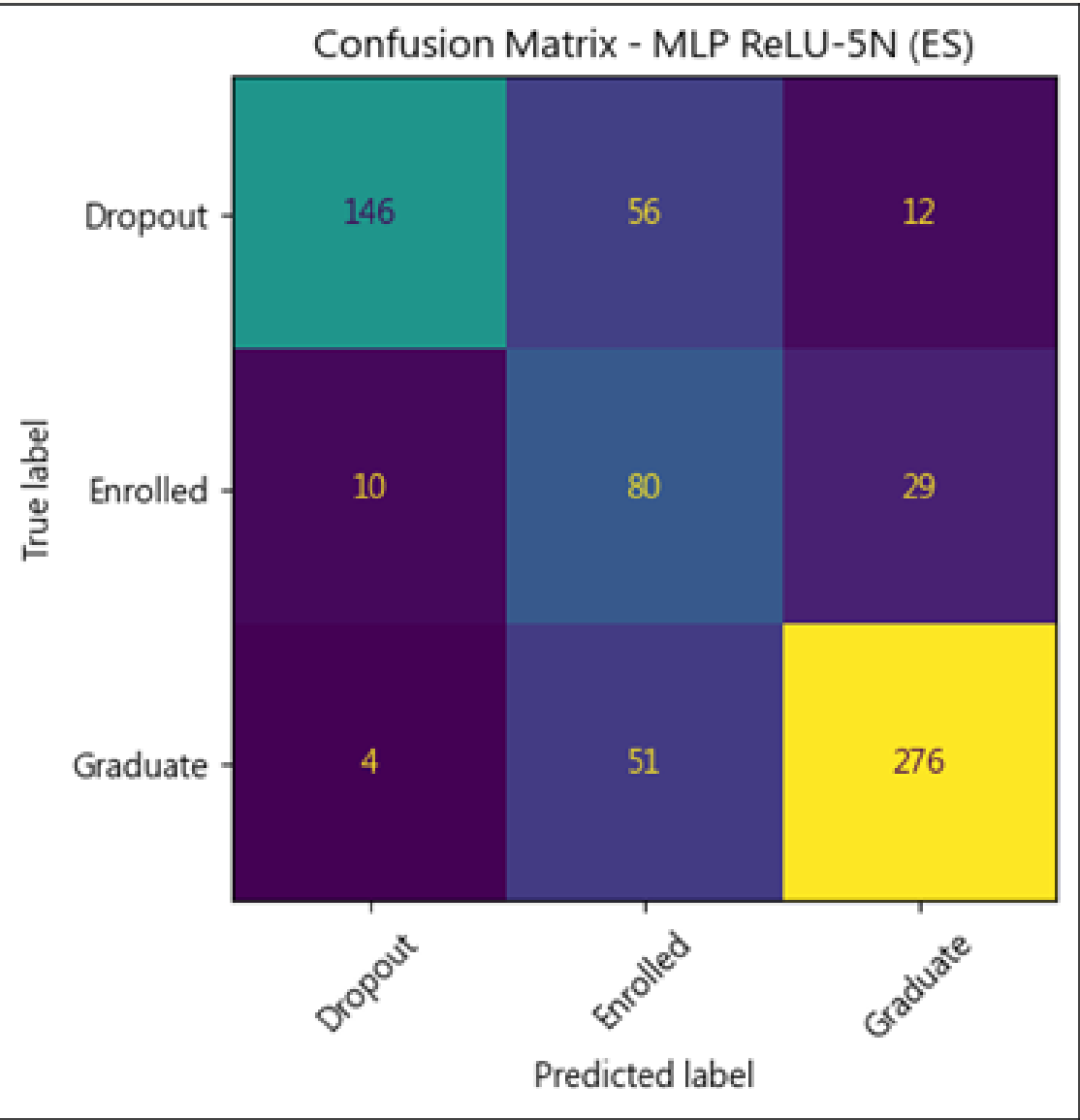
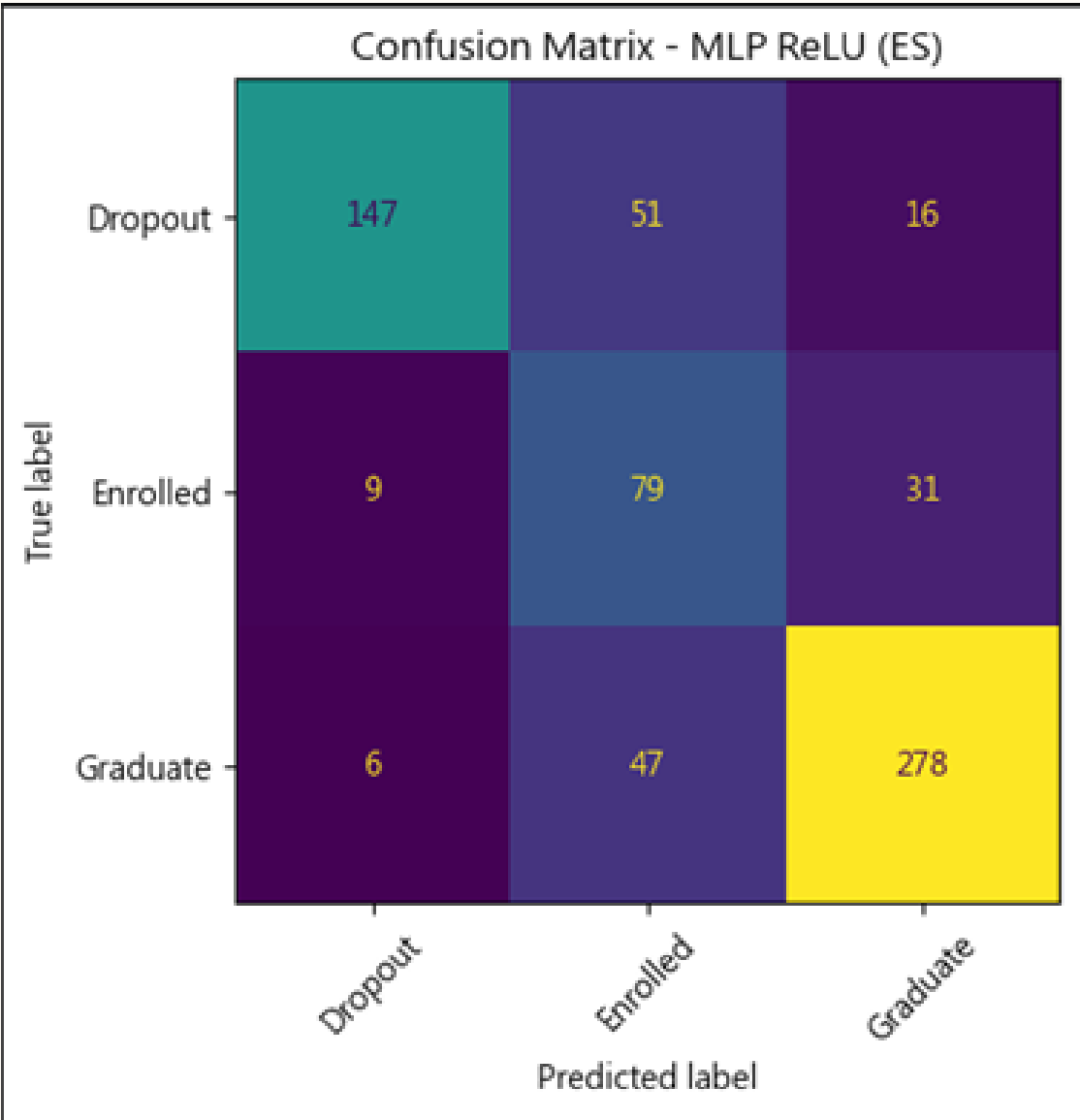
2	MLP ReLU (ES)	0.763554	0.802031	0.763554	0.774270
3	MLP ReLU-5N (ES)	0.737952	0.807275	0.737952	0.756731



測試

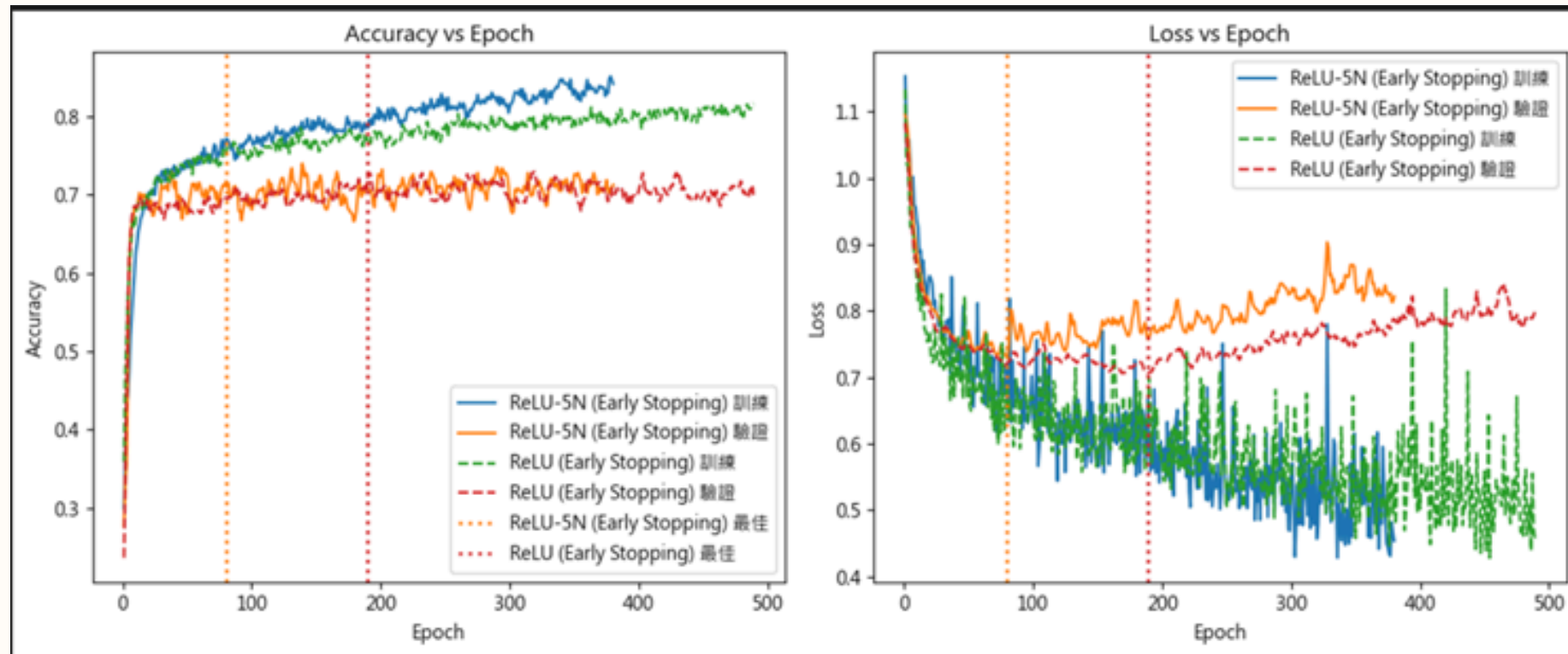
IR=3e-4(0.0003)

2	MLP ReLU (ES)	0.759036	0.798841	0.759036	0.770170
3	MLP ReLU-5N (ES)	0.756024	0.804779	0.756024	0.769978

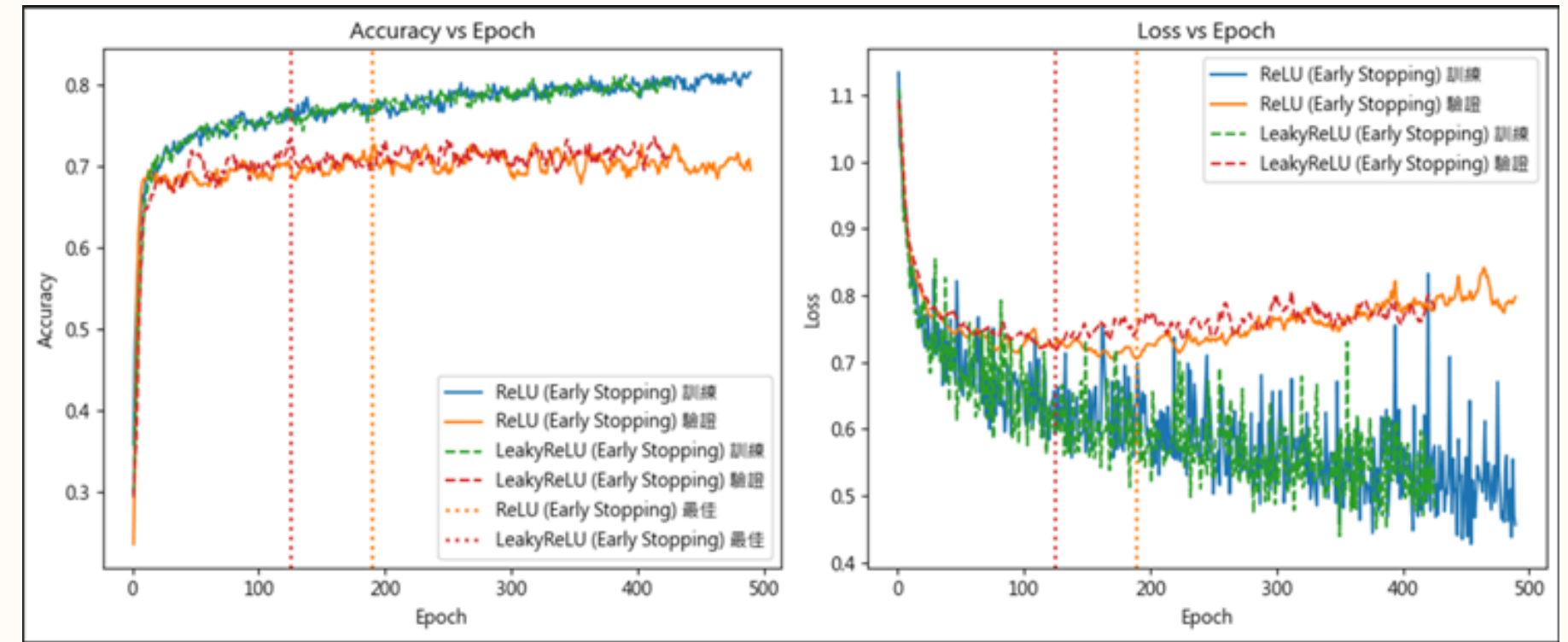


- 最佳參數組合
 - 驗證、測試_batch_size=256
 - 訓練batch_size =1024
 - 優化器:Adam
 - LR=1e-3
 - weight_decay:L2正則
 - 用驗證停止(30次無進步停止、最高300次)，
- 激活函數加上 Leaky_ReLU(0.01) 比較 以及機器學習模型 XGboosts 比

ReLU_2層 VS ReLU_5層



ReLU_2層 VS LeakyReLU

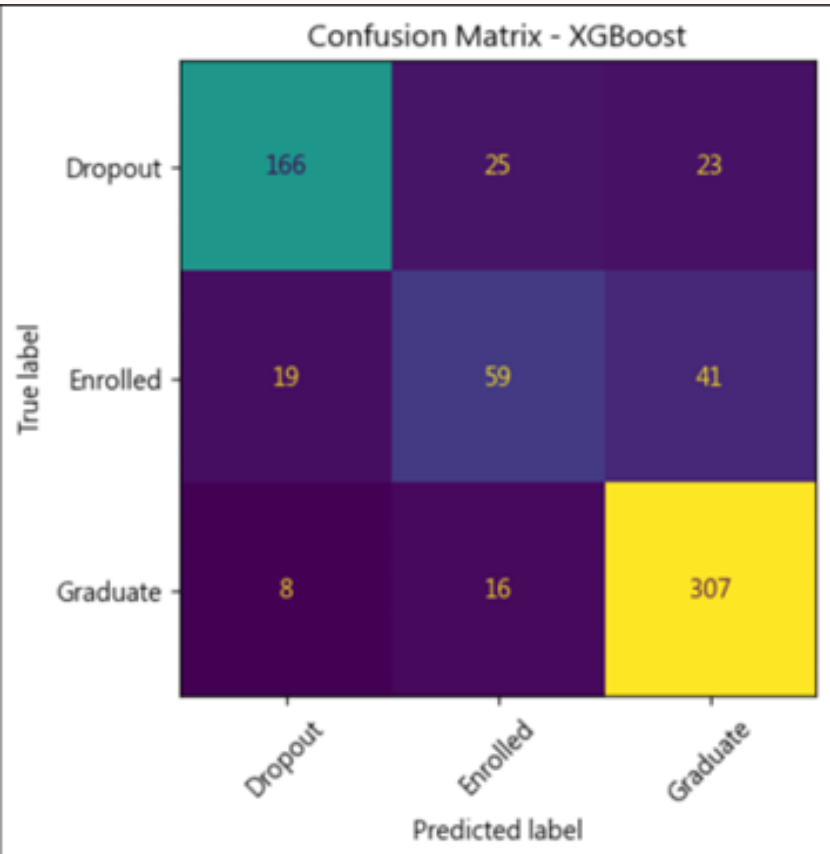


```
=== ReLU-5N (Early Stopping) 模型最終結果 ===  
Train Loss: 0.7015  
Val Loss: 0.7524  
Train Acc: 0.7432  
Val Acc: 0.7093  
  
=== ReLU (Early Stopping) 模型最終結果 ===  
Train Loss: 0.6755  
Val Loss: 0.7515  
Train Acc: 0.7384  
Val Acc: 0.6988  
  
=== LeakyReLU (Early Stopping) 模型最終結果 ===  
Train Loss: 0.6773  
Val Loss: 0.7506  
Train Acc: 0.7183  
Val Acc: 0.6807
```

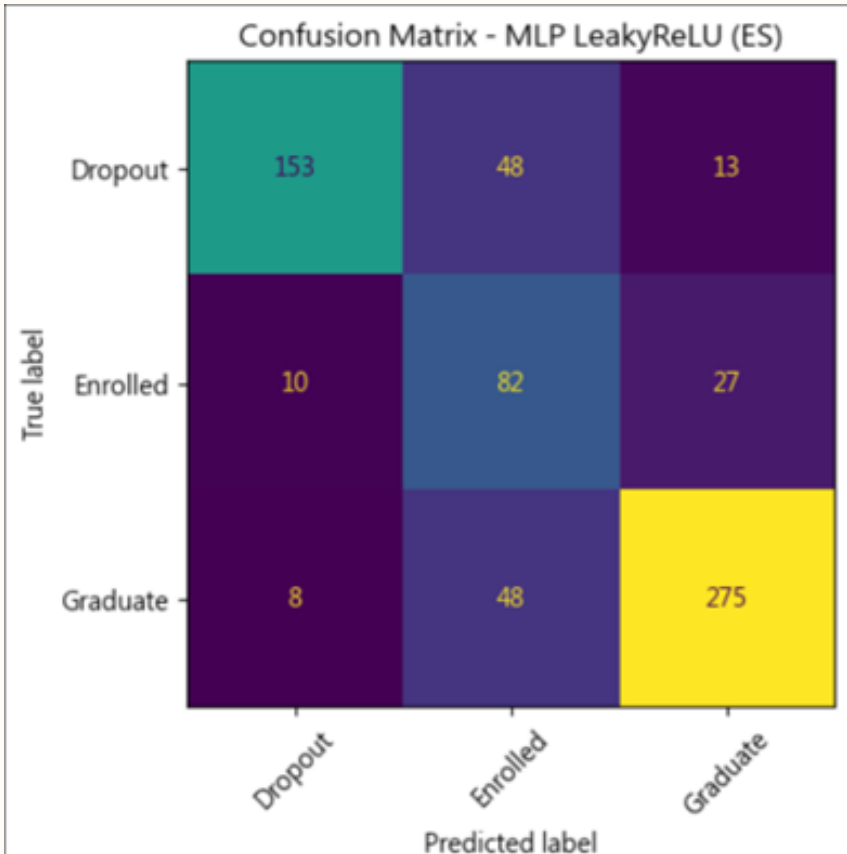
測試

	Model	Accuracy	Precision(w)	Recall(w)	F1(w)
0	XGBoost	0.801205	0.795440	0.801205	0.795468
1	MLP LeakyReLU (ES)	0.768072	0.806118	0.768072	0.779533
2	MLP ReLU (ES)	0.733434	0.798545	0.733434	0.752466
3	MLP ReLU-5N (ES)	0.765060	0.804169	0.765060	0.775272

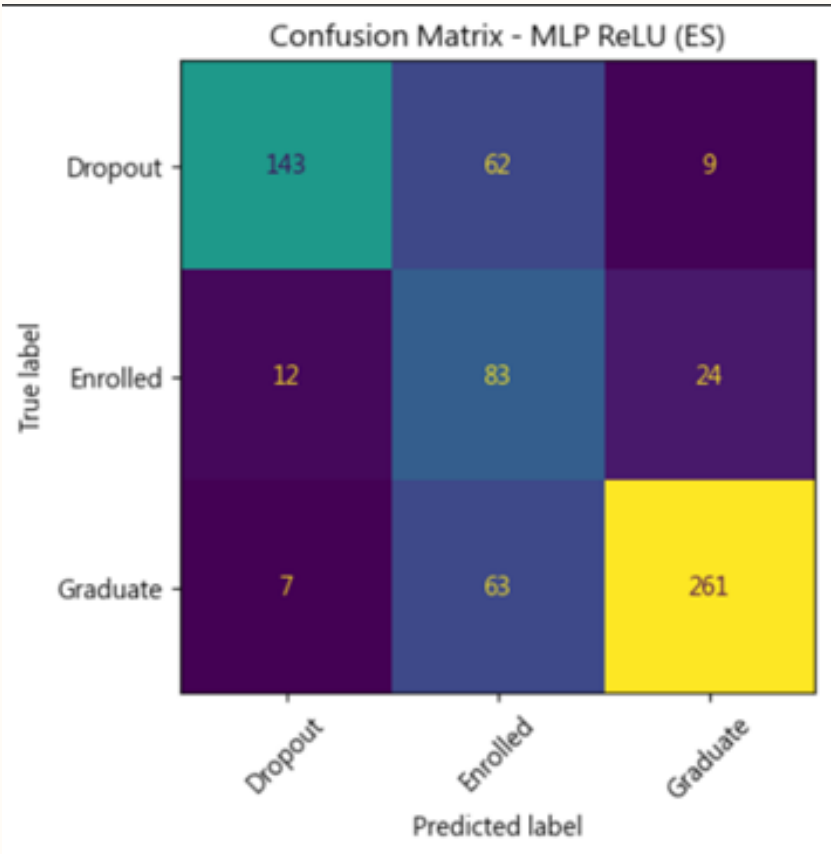
XGBoost



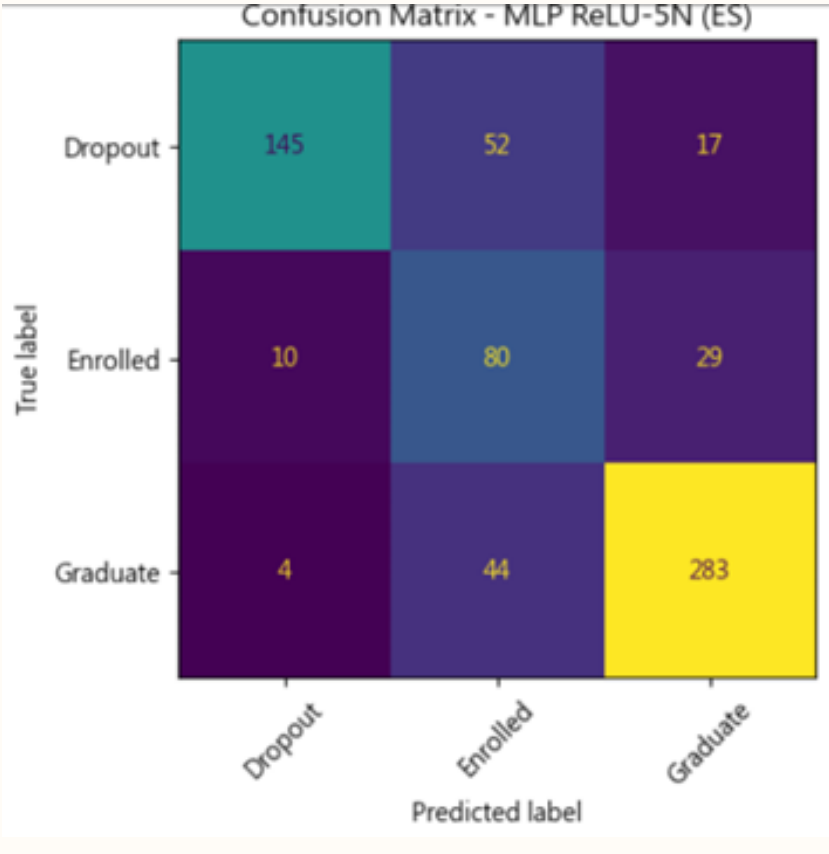
LeakyReLU



ReLU_2層



ReLU_5層



結論

- MLP 模型在不同超參數設定下（層數、Batch Size、學習率、激活函數）表現差異明顯。
- 兩層隱藏層（Relu_2N）整體表現最佳，但在 Batch Size = 1024 時，Relu_5N 偶爾會更好，表示深層網路在大批次訓練下能學得更充分。
- 學習率 $1e-3$ 表現最佳，收斂快又穩定；ReLU 激活函數效果也比 LeakyReLU 好。

謝謝大家