機器學習< Assignment #5 – DNN>

姓名：翁佳煌　　學號：409430030

要求 1.

在要求 1 中，都是採用一開始 sample code 的模型(如下圖 1)，只需修改 model_nblocks 做觀察即可，詳細的觀察敘述將會放在後面問題與討論的部分，在此先大略描述更改的內容和一些截圖。

```python
def _block(in_features, out_features, drop_rate):
    return nn.Sequential(
        nn.Linear(in_features, out_features),      #全連接層
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate)
    )

class FOGModel(nn.Module):
    def __init__(self, p=cfg.model_dropout, dim=cfg.model_hidden, nblocks=cfg.model_nblocks):
        super(FOGModel, self).__init__()
        self.dropout = nn.Dropout(p)
        self.in_layer = nn.Linear(cfg.window_size*3, dim)
        self.blocks = nn.Sequential(*[_block(dim, dim, p) for _ in range(nblocks)])
        self.out_layer = nn.Linear(dim, 3)

                                                # dim=32
    def forward(self, x):                       # x.Size([1024,32,3])
        x = x.view(-1, cfg.window_size*3)       # x.Size([1024,96])
        x = self.in_layer(x)                    # x.Size([1024,32])
        for block in self.blocks:
            x = block(x)                        # x.Size([1024,32])
        x = self.out_layer(x)                   # x.Size([1024,3])
        return x
```

▲圖 1

model_nblocks=1:

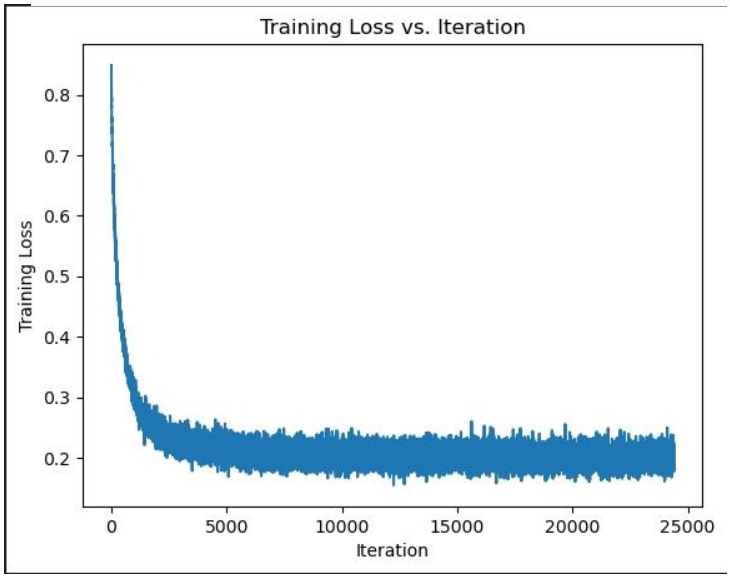
```python
class Config:
    train_dir1 = "/kaggle/input/tlvmc-parkinsons-freezing-gait-prediction/train/defog"
    train_dir2 = "/kaggle/input/tlvmc-parkinsons-freezing-gait-prediction/train/tdcsfog"

    batch_size = 1024                                   #batch size大小
    window_size = 32                                    #每次Input的總時序資料長度
    window_future = 8                                   #取目標時間點後多少為Input
    window_past = window_size - window_future           #取目標時間點前多少為Input

    wx = 8                                              #資料padding長度的參數

    optimizer_name = "Adam"                             #optimizer
    loss_function = "BCEWithLogitsLoss"                 #loss function

    model_dropout = 0.2                                 #dropout機率
    model_hidden = 32                                   #每層linear layer的神經元數
    model_nblocks = 1                                   #model內的block數

    lr = 0.00015                                        #learning rate
    num_epochs = 5                                      #訓練的epochs數
    device = 'cuda:0' if torch.cuda.is_available() else 'cpu'   #裝置(GPU)設定

    feature_list = ['AccV', 'AccML', 'AccAP']
    label_list = ['StartHesitation', 'Turn', 'Walking']


cfg = Config()
```

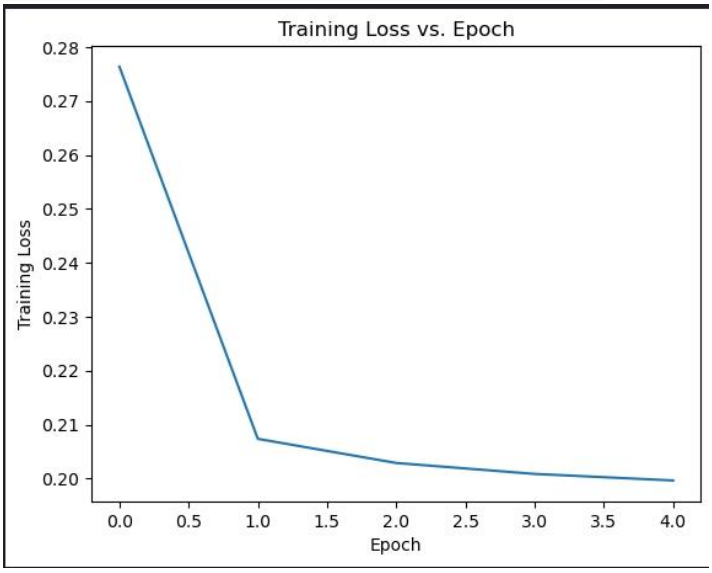
▲圖 2

ML_Parkinson's Freezing of Gait Prediction 8b8e11 - Version 5

Succeeded · 42m ago · Notebook ML_Parkinson's Freezing of Gait Prediction 8b8e11 | Block=1

0.244



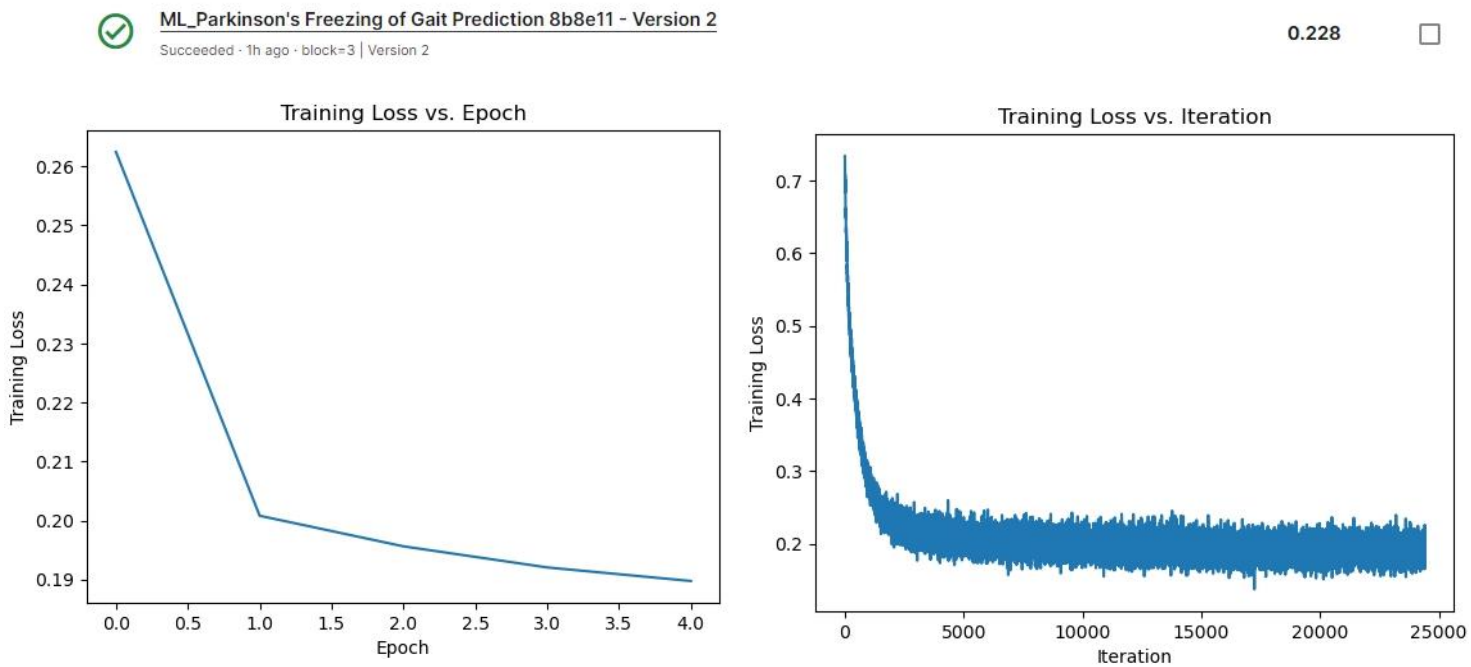


▲圖 3

model_nblocks=3:

```python
class Config:
    train_dir1 = "/kaggle/input/tlvmc-parkinsons-freezing-gait-prediction/train/defog"
    train_dir2 = "/kaggle/input/tlvmc-parkinsons-freezing-gait-prediction/train/tdcsfog"

    batch_size = 1024                                   #batch size大小
    window_size = 32                                    #每次Input的總時序資料長度
    window_future = 8                                   #取目標時間點後多少為Input
    window_past = window_size - window_future           #取目標時間點前多少為Input

    wx = 8                                              #資料padding長度的參數

    optimizer_name = "Adam"                             #optimizer
    loss_function = "BCEWithLogitsLoss"                 #loss function

    model_dropout = 0.2                                 #dropout機率
    model_hidden = 32                                   #每層linear layer的神經元數
    model_nblocks = 3                                   #model內的block數

    lr = 0.00015                                        #learning rate
    num_epochs = 5                                      #訓練的epochs數
    device = 'cuda:0' if torch.cuda.is_available() else 'cpu'   #裝置(GPU)設定

    feature_list = ['AccV', 'AccML', 'AccAP']
    label_list = ['StartHesitation', 'Turn', 'Walking']


cfg = Config()
```

▲圖 4

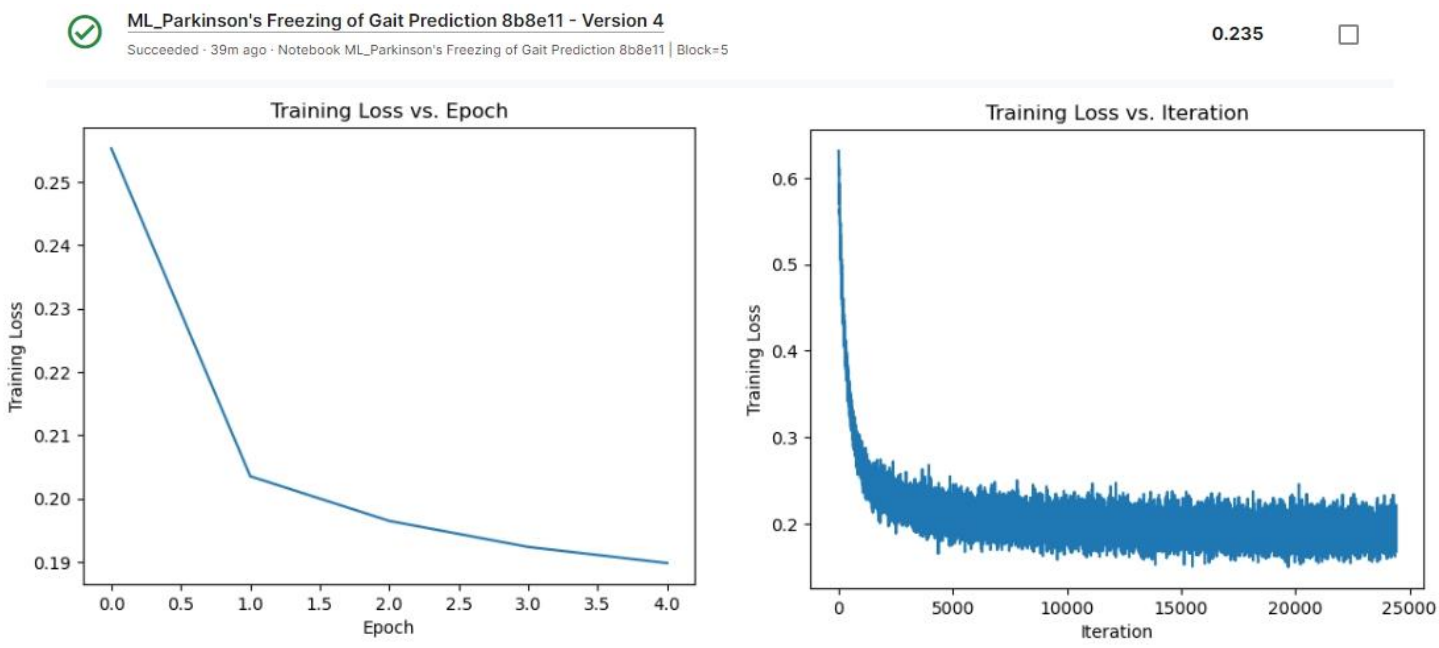


▲圖 5

model_nblocks=5:

```python
class Config:
    train_dir1 = "/kaggle/input/tlvmc-parkinsons-freezing-gait-prediction/train/defog"
    train_dir2 = "/kaggle/input/tlvmc-parkinsons-freezing-gait-prediction/train/tdcsfog"

    batch_size = 1024                                   #batch size大小
    window_size = 32                                    #每次Input的總時序資料長度
    window_future = 8                                   #取目標時間點後多少為Input
    window_past = window_size - window_future           #取目標時間點前多少為Input

    wx = 8                                              #資料padding長度的參數

    optimizer_name = "Adam"                             #optimizer
    loss_function = "BCEWithLogitsLoss"                 #loss function

    model_dropout = 0.2                                 #dropout機率
    model_hidden = 32                                   #每層linear layer的神經元數
    model_nblocks = 5                                   #model內的block數

    lr = 0.00015                                        #learning rate
    num_epochs = 5                                      #訓練的epochs數
    device = 'cuda:0' if torch.cuda.is_available() else 'cpu'   #裝置(GPU)設定

    feature_list = ['AccV', 'AccML', 'AccAP']
    label_list = ['StartHesitation', 'Turn', 'Walking']


cfg = Config()
```

▲圖 6

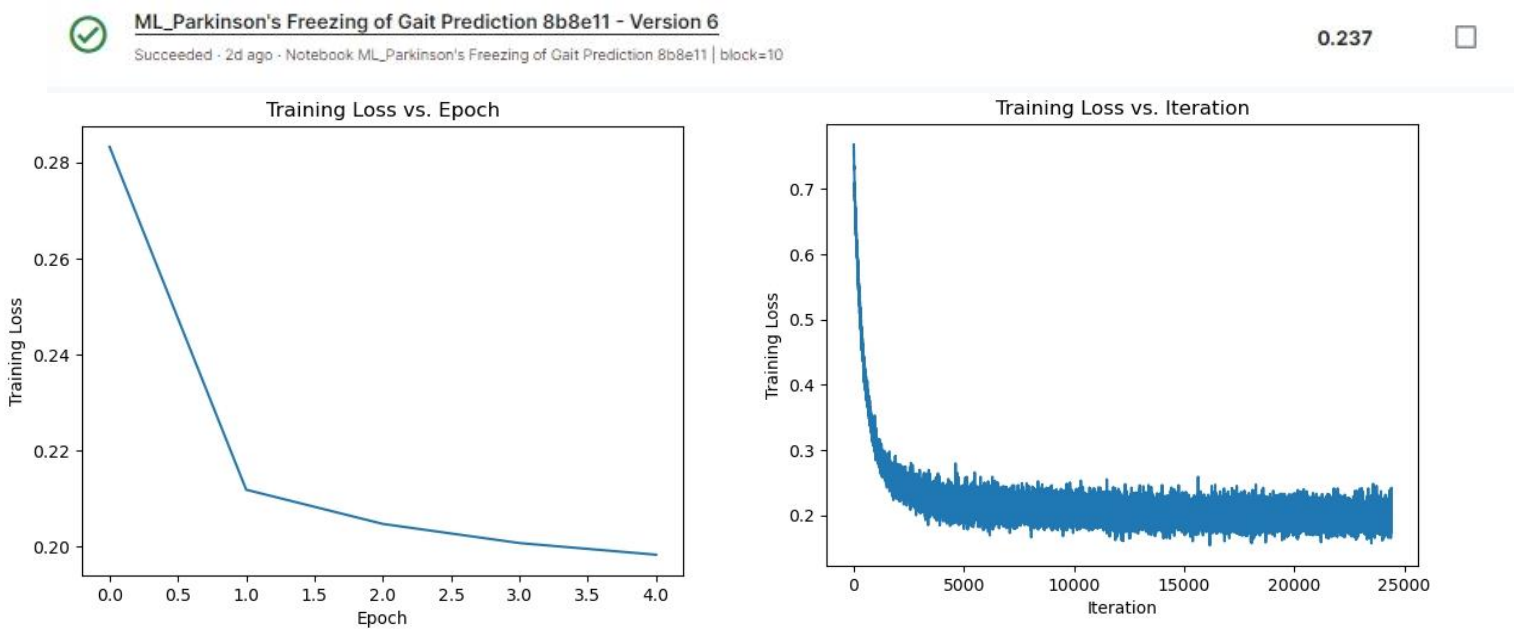


▲圖 7

model_nblocks=10:

```python
class Config:
    train_dir1 = "/kaggle/input/tlvmc-parkinsons-freezing-gait-prediction/train/defog"
    train_dir2 = "/kaggle/input/tlvmc-parkinsons-freezing-gait-prediction/train/tdcsfog"

    batch_size = 1024                                   #batch size大小
    window_size = 32                                    #每次Input的總時序資料長度
    window_future = 8                                   #取目標時間點後多少為Input
    window_past = window_size - window_future           #取目標時間點前多少為Input

    wx = 8                                              #資料padding長度的參數

    optimizer_name = "Adam"                             #optimizer
    loss_function = "BCEWithLogitsLoss"                 #loss function

    model_dropout = 0.2                                 #dropout機率
    model_hidden = 32                                   #每層linear layer的神經元數
    model_nblocks = 10                                   #model內的block數

    lr = 0.00015                                        #learning rate
    num_epochs = 5                                      #訓練的epochs數
    device = 'cuda:0' if torch.cuda.is_available() else 'cpu'   #裝置(GPU)設定

    feature_list = ['AccV', 'AccML', 'AccAP']
    label_list = ['StartHesitation', 'Turn', 'Walking']


cfg = Config()
```

▲圖 8

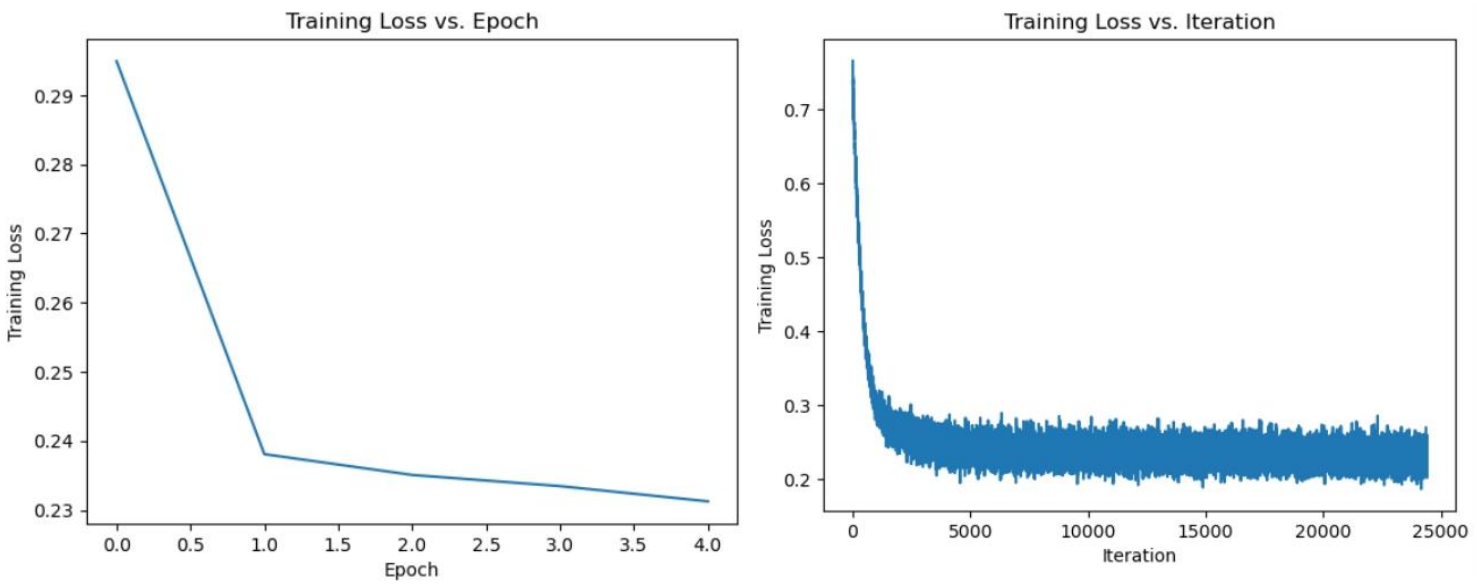


▲圖 9

model_nblocks=20:

model_nblocks 改成 20 後發現 training_loss 和 val_score 都比前幾次的結果都來的糟糕，這可能是由於模型的複雜性增加而導致的優化問題、超參數調整不當或其他因素。

因為 Kaggle 上繳限制的緣故，這裡就沒有再上傳這個較差版本的結果上去測試正確率。



▲圖 10

要求 2.

再綜合觀察正確率、val_loss 和 val_score 的數據後，我發現 model_nblocks=10 在原本的模型架構上(上圖 1)的表現是最好的，因此要求 2 一開始先採用 model_nblocks=10 的前提下，去把模型**多添加了額外的全連接層**(如下圖 11)。

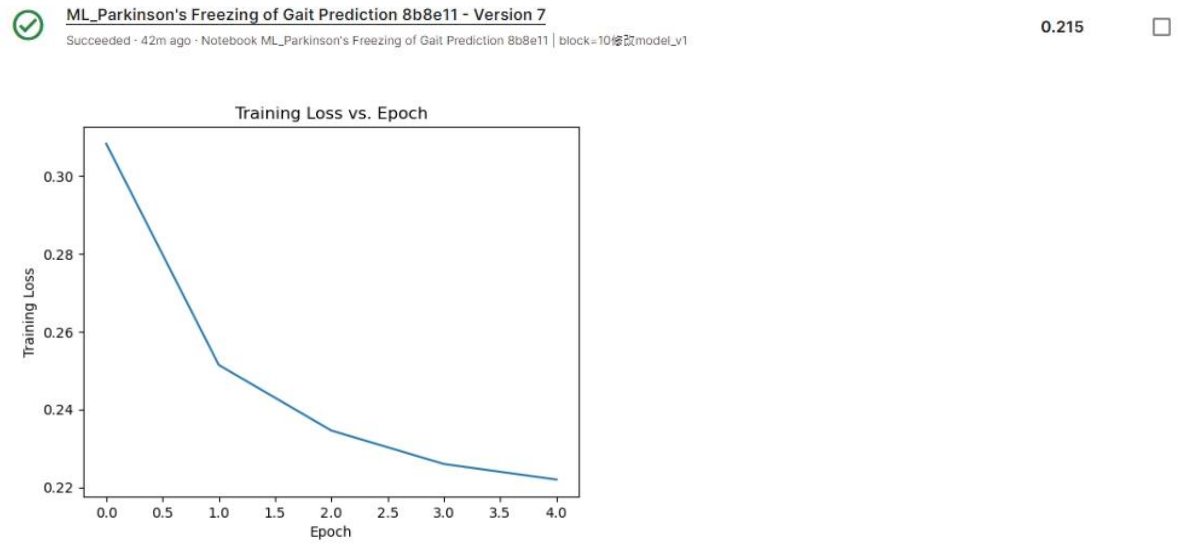
```python
def _block(in_features, out_features, drop_rate):
    return nn.Sequential(
        nn.Linear(in_features, out_features),      # 全連階層
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate),
        nn.Linear(out_features, out_features),      # 添加額外的全連階層
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate)
    )

class FOGModel(nn.Module):
    def __init__(self, p=cfg.model_dropout, dim=cfg.model_hidden, nblocks=cfg.model_nblocks):
        super(FOGModel, self).__init__()
        self.dropout = nn.Dropout(p)
        self.in_layer = nn.Linear(cfg.window_size*3, dim)
        self.blocks = nn.Sequential(*[_block(dim, dim, p) for _ in range(nblocks)])  # 增加block層數
        self.out_layer = nn.Linear(dim, 3)

                                                # dim=32
    def forward(self, x):                       # x.Size([1024,32,3])
        x = x.view(-1, cfg.window_size*3)       # x.Size([1024,96])
        x = self.in_layer(x)                    # x.Size([1024,32])
        for block in self.blocks:
            x = block(x)                        # x.Size([1024,32])
        x = self.out_layer(x)                   # x.Size([1024,3])
        return x
```

▲圖 11

```
--------------------------------------------------
Epoch: 4

100%|██████████| 4883/4883 [03:12<00:00, 25.43it/s]

Train Loss: 0.1983

100%|██████████| 4868/4868 [02:21<00:00, 34.28it/s]

Validation Loss: 0.0891, Validation Score: 0.220, ClassWise: 0.040,0.542,0.079
==================================================
```

▲圖 12

上繳的正確率和 loss 結果如下圖 13。

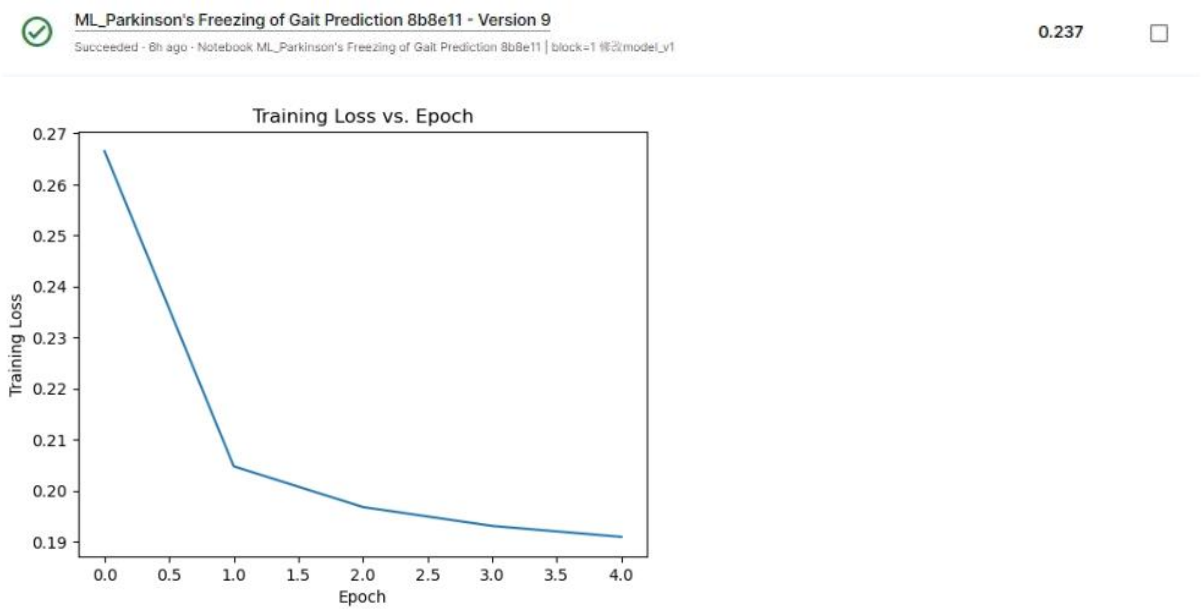


▲圖 13

我發現正確率反而不如預期，於是我再次測試一次 model_nblocks=1 和 model_nblocks=5，對同樣兩層全連接層的模型(上圖 11)再去跑一次，並觀察哪一個對修改後模型的正確率較好。

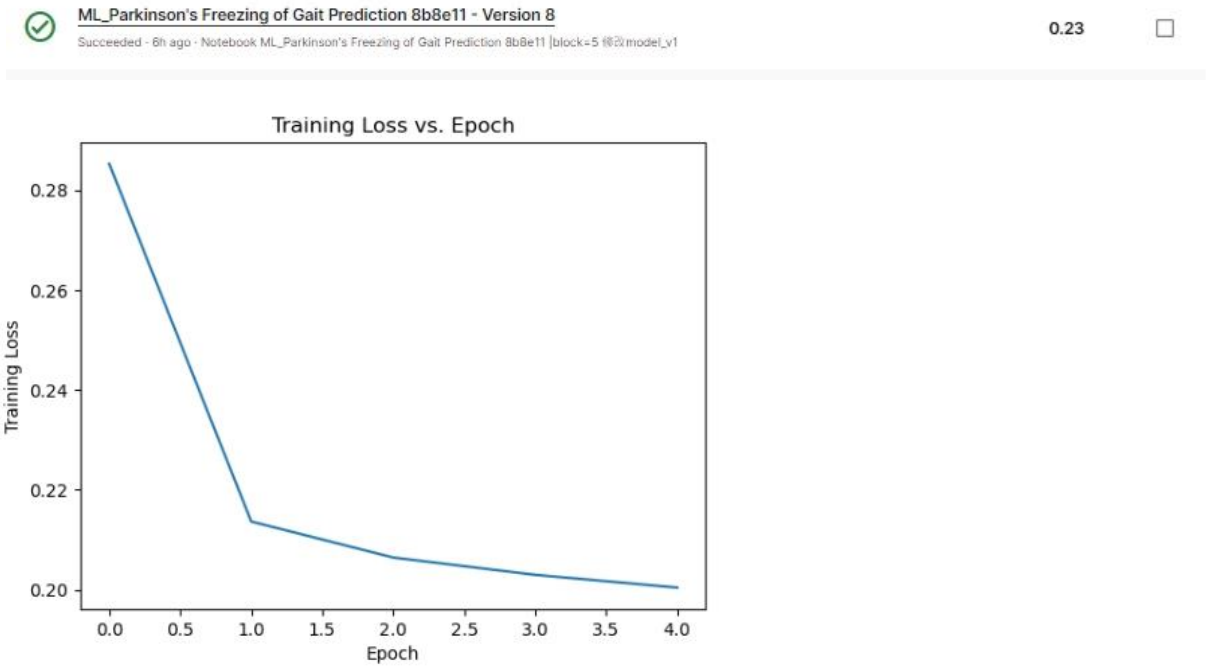
上繳的正確率結果如下圖 14 和圖 15。

model_nblocks=1:



▲圖 14

model_nblocks=5:



▲圖 15

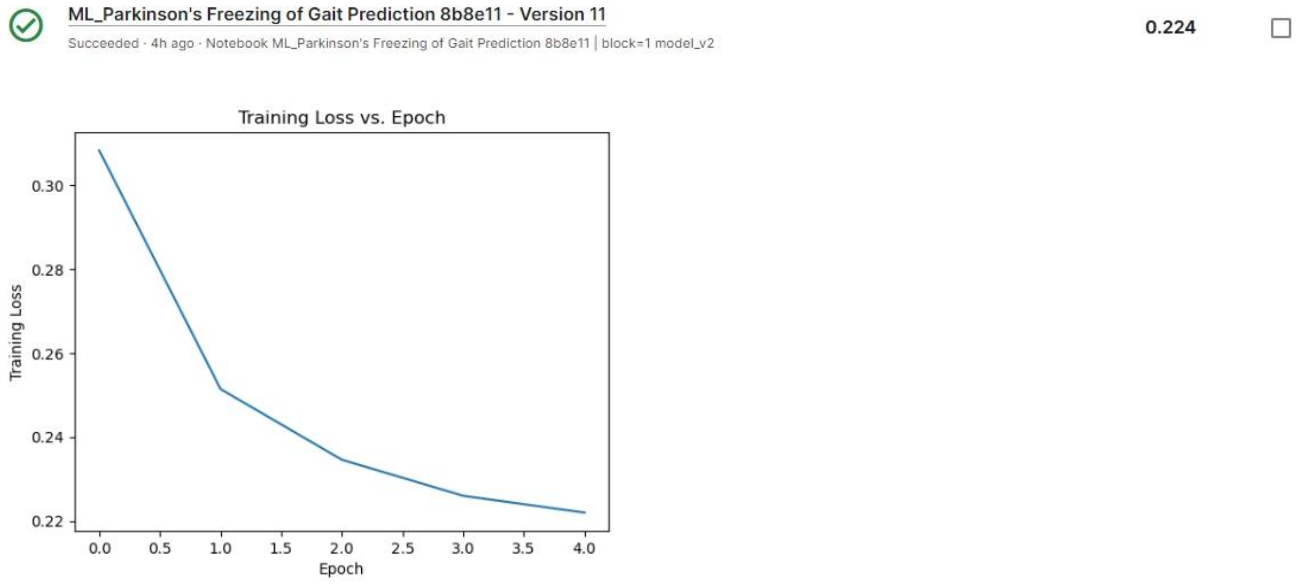
降低 model_nblocks 後準確率明顯提升一些，於是我決定再**多加一層** model 內 block 層數後，再去測試 model_nblocks=1 和 model_nblocks=5 對於下圖 16 模型的結果。

```python
def _block(in_features, out_features, drop_rate):
    return nn.Sequential(
        nn.Linear(in_features, out_features),      # 全連階層
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate),

        nn.Linear(out_features, out_features),      # 添加第二個全連階層
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate),

        nn.Linear(out_features, out_features),      # 添加第三個全連階層
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate)

    )

class FOGModel(nn.Module):
    def __init__(self, p=cfg.model_dropout, dim=cfg.model_hidden, nblocks=cfg.model_nblocks):
        super(FOGModel, self).__init__()
        self.dropout = nn.Dropout(p)
        self.in_layer = nn.Linear(cfg.window_size*3, dim)
        self.blocks = nn.Sequential(*[_block(dim, dim, p) for _ in range(nblocks)])  # 增加block層數
        self.out_layer = nn.Linear(dim, 3)

                                                # dim=32
    def forward(self, x):                       # x.Size([1024,32,3])
        x = x.view(-1, cfg.window_size*3)       # x.Size([1024,96])
        x = self.in_layer(x)                    # x.Size([1024,32])
        for block in self.blocks:
            x = block(x)                        # x.Size([1024,32])
        x = self.out_layer(x)                   # x.Size([1024,3])
        return x
```

▲圖 16

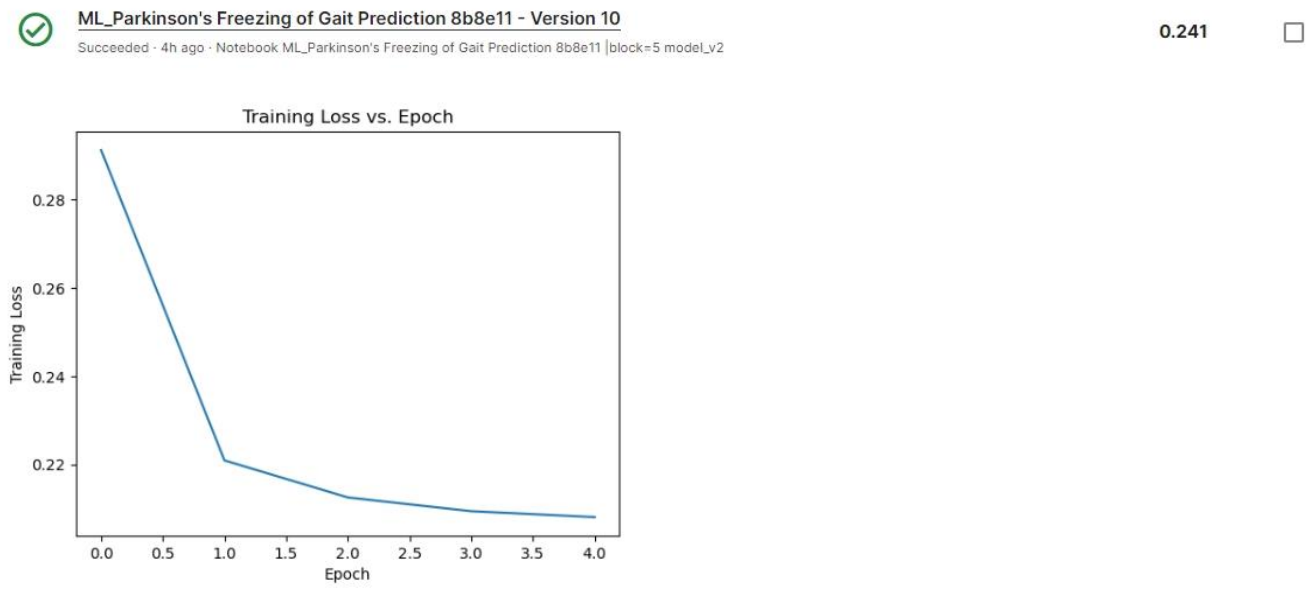
從下圖 17 和圖 18 的結果發現，model_nblocks=5 採用**三層全連接層**的正確率較好，因此接下來都使用這個 model_nblocks 數量去增加/減少模型的深度。

model_nblocks=1:



▲圖 17

model_nblocks=5:



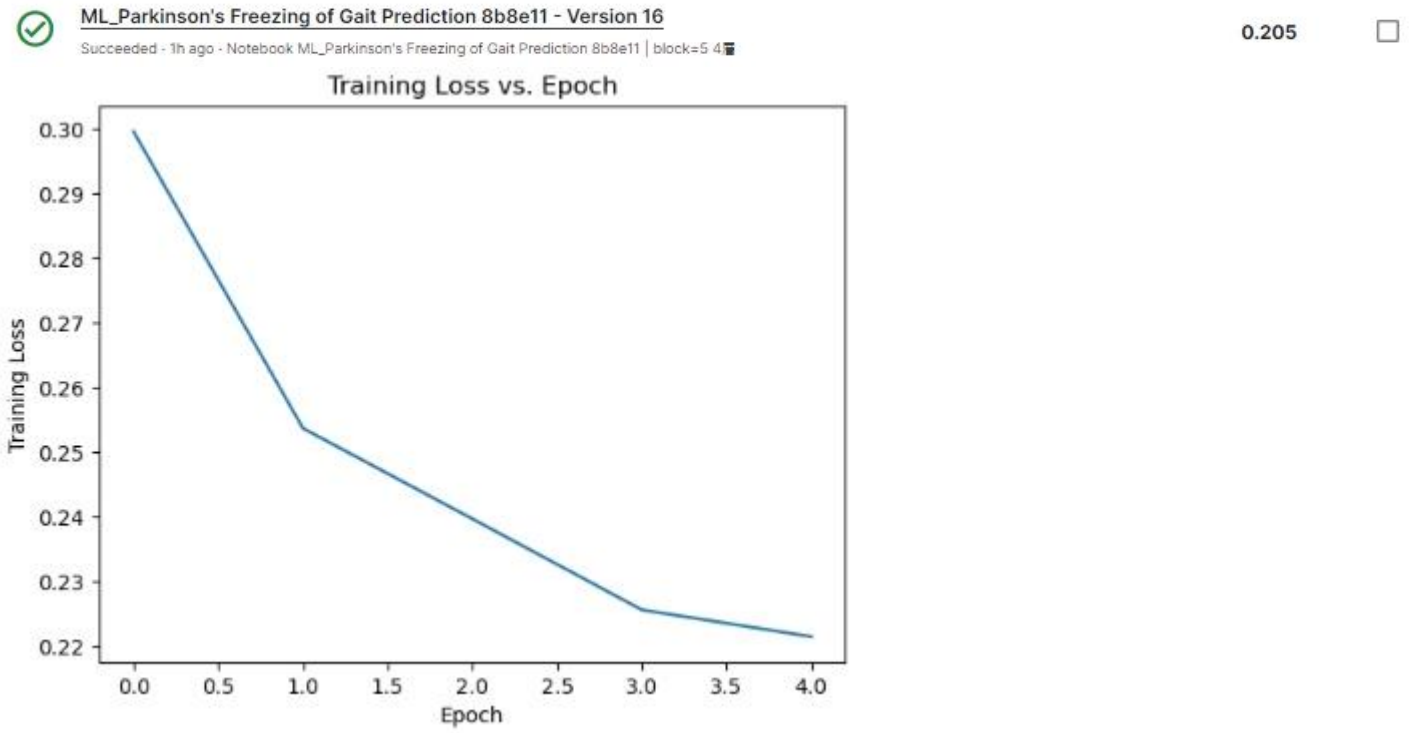
▲圖 18

為了更加提升準確率，我又**再次多加了一層全連接層**去測試，也就是目前有**四層**全連接層，如下圖 19。

```
def _block(in_features, out_features, drop_rate):
    return nn.Sequential(
        nn.Linear(in_features, out_features),       # 全連階層
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate),

        nn.Linear(out_features, out_features),      # 添加第二個全連階層
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate),

        nn.Linear(out_features, out_features),      # 添加第三個全連階層
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate),

        nn.Linear(out_features, out_features),      # 添加第四個全連階層
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate)
    )
```

▲圖 19

發現增加為四層全連接層反而導致正確率下降，我發現當增加全連接層的層數時，並不一定意味著模型的性能會提高，而是可能會導致過度擬合或增加訓練難度。



▲圖 20

問題與討論

要求 1:

在要求 1 發現當設 block=1 的時候上傳的正確率最高為 0.244，然後 block=3 的時候最低 0.228，block=5 為 0.235，block=10 是 0.237。
但再綜合觀察 val_loss 和 val_score 的數據後，我發現 block=10 的表現是最好的，因此要求 2 就決定用 block=10 的前提下，去更改模型。

我認為造成這個問題的可能原因如下:
模型複雜性不匹配： 增加 model_nblocks 將會增加模型的複雜性和容量，可以提供更多的表示能力。但是，如果數據集的規模相對較小，或者訓練數據不足以支撐更複雜的模型，那麼增加 model_nblocks 可能會導致過擬合，從而降低在驗證集上的性能。

特定數據集特徵： 不同的數據集具有不同的特徵和模式。在某些情況下，使用較簡單的模型可能能夠更好地擬合數據集的特定模式。在您的情況下，將 model_nblocks 設置為 1 可能剛好能夠適應數據集中的模式，而增加 model_nblocks 後，模型變得過於復雜，不再適應數據集的模式，從而導致性能下降。

超參數調整： 增加 model_nblocks 時，可能需要重新調整其他超參數，如學習率、正則化等。如果您只是增加了 model_nblocks 而沒有相應地調整其他超參數，那麼模型的性能可能會受到影響。

要求 2:

增加模型中神經網路架構的深度。每個額外的區塊都會添加更多層到模型中，從而增強模型學習複雜關係的能力。透過增加全連接層的層數，模型的容量和表達能力得到提升。每個全連接層都可以學習到不同的特徵轉換和表示。透過多個全連接層的組合，模型可以更深入地學習到輸入資料中的抽象特徵和模式，從而更好地進行預測和分類任務。
然而，需要注意的是，過於深的模型也可能導致問題，例如過度擬合或訓練困難。增加全連接層的層數和 nblocks 數量應該根據特定任務和數據集的性質進行調整和優化。通常，通過實驗和模型評估來找到最適合的模型架構，並避免過度複雜或過於簡單的模型。