Overview

在本次的實驗報告中，利用 PyTorch 實作了 SCCNet 的架構，透過論文中可以調整的參數，在三個訓練分類中，記錄了超過 20 萬組的訓練次數，最終 SD 拿到最高的準確度為 65.23%、LOSO 為 63.54%、LOSO+FT 為 80.90%。

本實驗報告首先會簡單介紹 Trainer 與 Tester 實作了哪些功能，接著講解 SCCNet 的架構，和簡單講解實作方法。接著是在訓練階段根據訓練給的參數做比較，如何調整參數才能達到更佳的準確度。

在本實驗報告的最後一小節則是討論 EEG 資料集的訓練難度和透過這次實驗能如何去提升它的準確度。

Implementation Details

Details of training and test code

在 training.py 中設計了一個 Trainer 的 Class，在建構子帶入的參數有。

- numClasses：最後輸出的類別數量
- timeSample：每一個時間段(single trials)的特徵值
- Nu：第一層 Conv 的輸出通道數(Out_Channel)
- C：EEG 資料的 C(22) 個檢測點
- Nc：第二層 Conv 的輸出通道數
- Nt：第一層 Conv 的 feature map 的 W 大小
- dropout_rate：設定 SCCNet 中 Dropout 的大小
- weight_decay：設定 L2 regularization 大小
- learning_rate：設定 model 的學習率
- optimizer_name：設定 optimizer 的類型

對於這次實驗報告可以調配的參數有 Nu、Nt、learning_rate、optimizer 等，之外也可以對輸入的訓練集大小做 mini-batch 的訓練，因此 batch_size 也是一個可以調整的參數之一。

此外 Trainer 的 class 設計了加載訓練與測試資料集的函數，在加載資料時會需要設定每次在訓練時取資料的 batch 大小，而在訓練資料集上匯兌資料做亂數打亂的動作，而測試資料集上則不會做打亂的動作。

在訓練時會同時顯示測試資料上的準確度(accuracy)和訓練及本身的單次訓練的 loss。

本實驗所使用的 loss function 為 nn.CrossEntropyLoss()，因為此實驗是一個分類問題，且在 SCCNet 中的最後一層使用的是 softmax，將結果轉為機率分布，因此 Loss function 上採用 CrossEntropy 的方式。

Trainer 同時提供了 Train 和 Evaluate 的功能，但是對於 Tester 只能使用 Evaluate，Trainer 在 train 函數的設計上可以將資料儲存到資料庫，做統一的儲存(功能寫至 utils.py)，後續要將資料提出或者過濾上利用 SQL 語法都能很快的抓資料出來(在後續取資料環節都是使用)。

在模型權重的儲存上則是會根據可調配的參數，將會儲存在

{train_method}/{model_name}/Nu={Nu}-Nt={Nt}-LR={lr}-Opt={opti_name}-BZ={batch_size}/epoch={epoch}.pt

其中 train_method 是指本次作業中提到的 SD/LOSO/LOSO+FT，model_name 則是額外制定的變數名稱，用來辨識不同的 model。

在 Tester class 的撰寫上都跟 Trainer 差不多，只是擷取部分功能，能直接抓取特定位置的權重紀錄檔。

Details of the SCCNet

SCCNet 的架構分成四層。

1. Conv2D
2. Conv2D
3. Pooling
4. Softmax

在第一層的 Conv2D，使用N_u個大小為(C, N_t)的 feature map，這一層的 Conv2D 做完之後還會做 Batch Normalization 和 Zero-padding。

在第二層的 Conv2D，使用Nc個大小為$(1,12)$的 feature map，這一層的 Conv2D 做完之後還會做 Batch Normalization 和 Zero-padding，這邊的 Zero-padding 會特別拉到一樣大小的 size，但是因為 kernel map 是偶數，在 padding 時必定會多一格出來。

在第二層結束後會做 SquareLayer 去放大特徵值，接著做 dropout 防止 training 時出現 overfitting 的問題。

在第三層的 pooling 則是使用 AvgPool，pooling 完畢之後會再做一個 Log (Activation) Layer。

做完 Log Layer 之後會將資料攤平，然後進入 fully connect 層。

Fully connect 層出來之後會得到 4 個 classes 的輸出，接著進到最後一層 softmax，實驗上使用 F.log_softmax()，提升在 training 時的速度。

在 SCCNet 的實作上，因為考慮到最後的 Fully Connect 層需要先去計算可能輸入的大小，因此在 SSCNet 上還有寫計算 Conv 的輸出和 pooling 層的輸出和最後要給 Fully Connect，以下為三者的計算方式。

Conv Layer：

$$H_{out} = \frac{\mathrm{H_{in}} - K_h + 2 \times P_h}{\mathrm{S_h}} + 1 \text{、} W_{out} = \frac{W_{in} - K_w + 2 \times P_w}{S_w} + 1$$

H_{in}、W_{in}、H_{out}、W_{out}：Conv Layer 的輸入高和寬與輸出高和寬

K_h、K_w：Conv Layer 的 feature map 大小 (K_h, K_w)

S_h、S_w：Conv Layer 的 stride 大小 (S_h, S_w)

P_h、P_w：Conv Layer 的 padding 大小 (P_h, P_w)

Pooling Layer：

$$H_{out} = \left\lfloor \frac{H_{in} - K_h + 2 \times P_h}{S_h} \right\rfloor + 1 \text{、} W_{out} = \left\lfloor \frac{W_{in} - K_w + 2 \times P_w}{S_w} \right\rfloor + 1$$

H_{in}、W_{in}、H_{out}、W_{out}：Pooling Layer 的輸入高和寬與輸出高和寬

K_h、K_w：Pooling Layer 的 feature map 大小 (K_h, K_w)

S_h、S_w：Pooling Layer 的 stride 大小 (S_h, S_w)

P_h、P_w：Pooling Layer 的 padding 大小 (P_h, P_w)

FC Layer：

$$C_{in} = C \times N$$

C、N：對應到 Flatten 之後要算進入 FC 的大小

Anything you want to mention

在 SCCNet 的實作階段，有發現於論文中有提到，第一層 Conv 做完之後需要做 permute 的動作，但是在程式碼的實作上直接將第一層輸出對應到第二層的輸入，而在對第二層 Conv 的 kernel size 做轉換也能達到同樣訓練的效果，並且針對第二層的 feature map，去計算的話兩者的參數量是一致的。

$$Parameters = \big((Kernel\ Height) \times (Kernel\ Width) \times (Input\ Channel) + Bias\big) \times (Output\ Channels)$$

Analyze on the experiment results

Discover during the training process

在訓練時 SD 時，發現到不同 learning rate、Nu、Nt、optimizer、batch_size 取用下的差異。

- 不同的 learning rate：

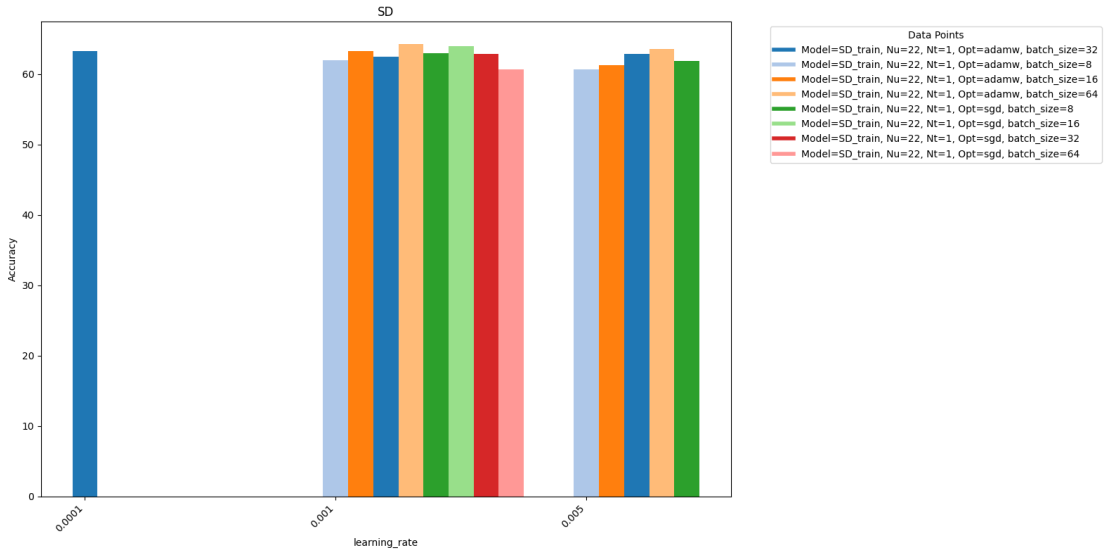


圖.1：Nu=22、Nt=1 下不同 learning_rate 的最高準確度

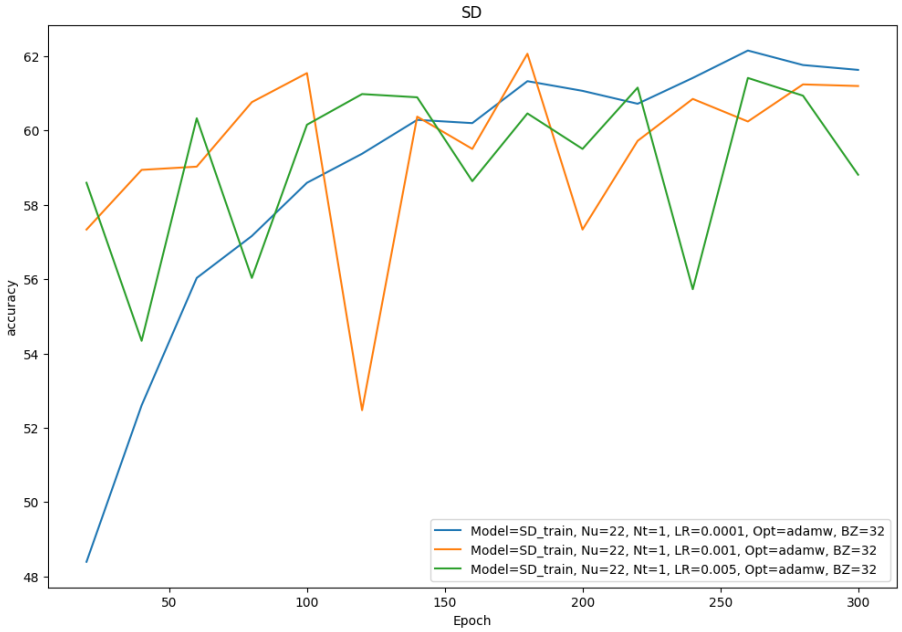


圖.2：Nu=22、Nt=1 下 optimizer 為 adamw 的不同 learning_rate 的準確度

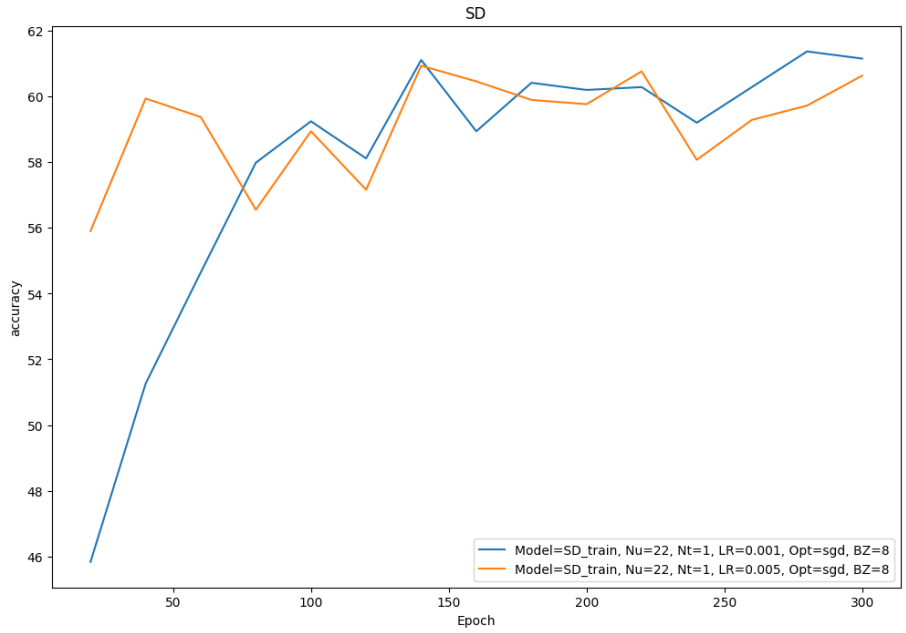


圖.3：Nu=22、Nt=1 下 optimizer 為 SGD 的不同 learning_rate 的準確度

Learning rate 越小的 model 訓練起來準確度越高、越穩定，而 adamw 對於 learning rate 很敏感，如果 learning rate 太高從圖.2 中可以看到 0.001 與 0.005 的波動是比圖.3 的 SGD 還要來的大，且不穩定的。此外從圖.1 來看，參數調整為 Nu=22、Nt=1 時不同 learning rate 的準確度都位於 60%上下。

- 不同的 Optimizer

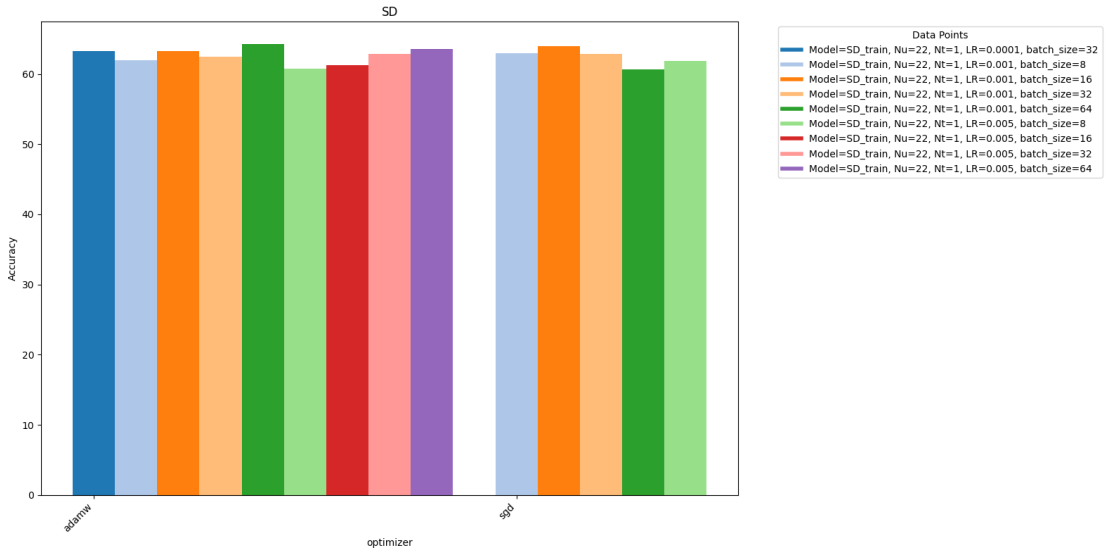


圖.4：Nu=22、Nt=1 下不同 optimizer 的最高準確度

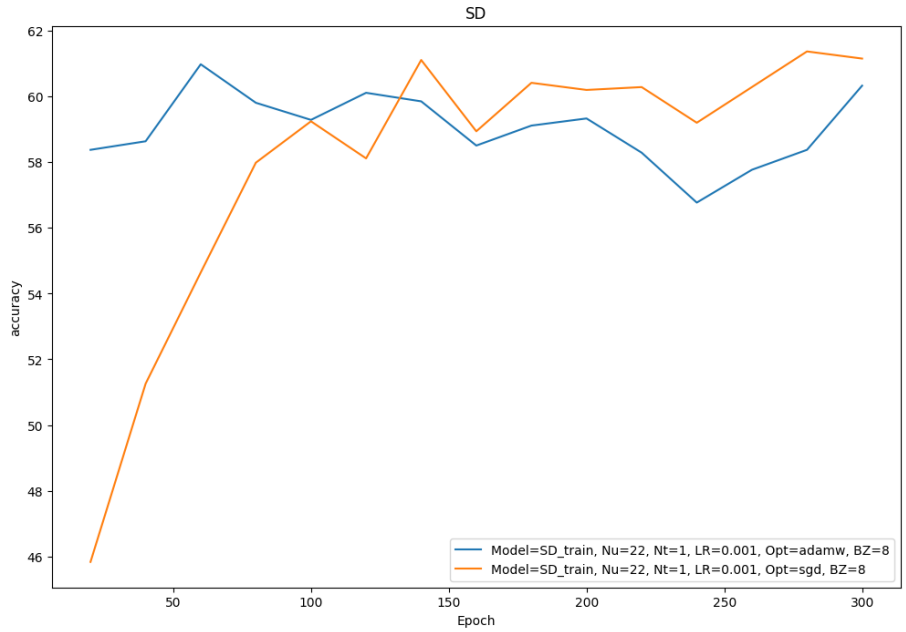


圖.5：Nu=22、Nt=1 下訓練時不同 optimizer 的準確度

在圖.5 中可以看到 adamw 的 optimizer 一開始的訓練效果就能達到 60 上下，但是因為 learning rate 太大的關係，adamw 沒法穩定的保持在準確度 60 以上。由圖.4 也可以看出在同樣的參數下訓練出來的結果都差不多

- 不同的 batch 大小

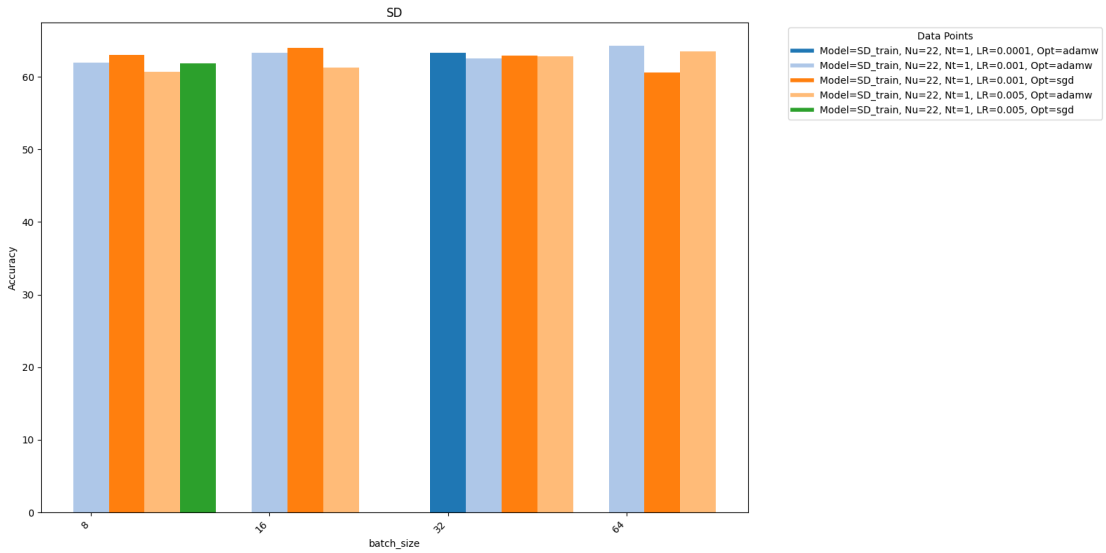


圖.6：Nu=22、Nt=1 時不同 batch_size 的最高準確度

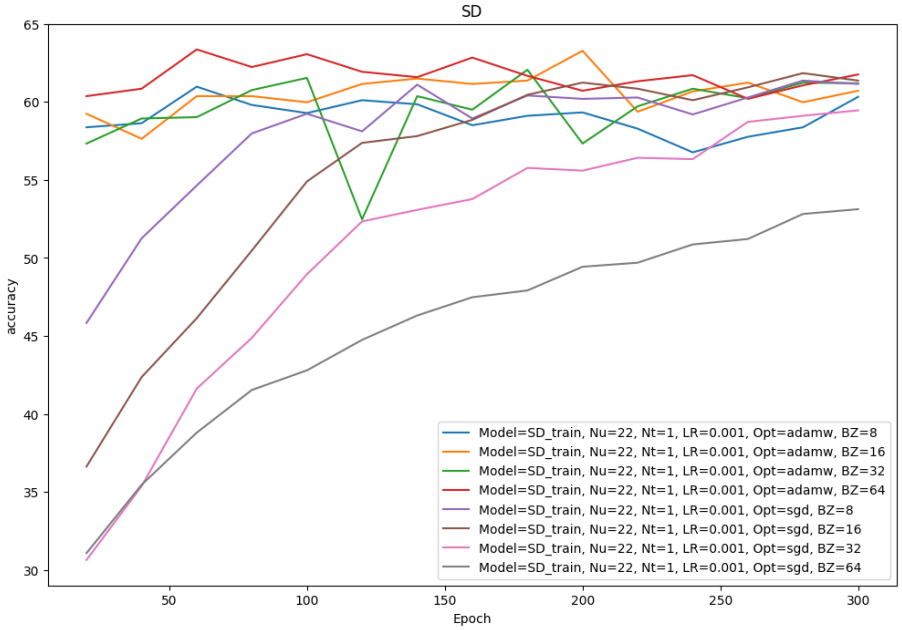


圖.7：Nu=22、Nt=1 下訓練時不同 batch_size 的準確度

由上圖.6 和圖.7 中可以看到，adamw 在 batch 越大時表現的效果越好，相反的 sgd 在 batch 大小越小時，準確率越高。

- 不同的 Nu

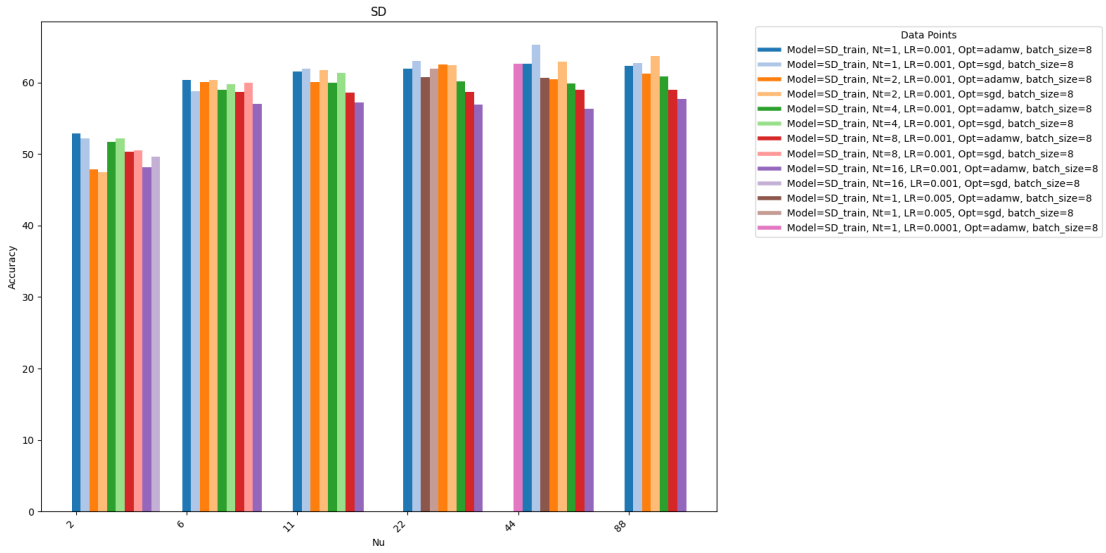


圖.8：batch_size=8 時不同參數的準確度

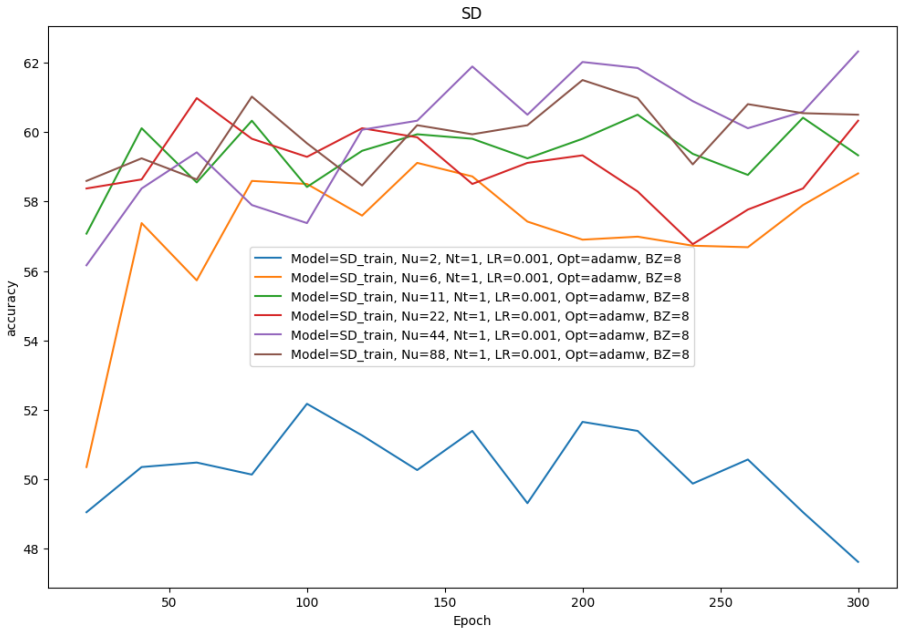


圖.9：batch_size=8、Nt=1、optimizer=adamw、learning_rate=0.001 在訓練時不同 Nu 的準確度

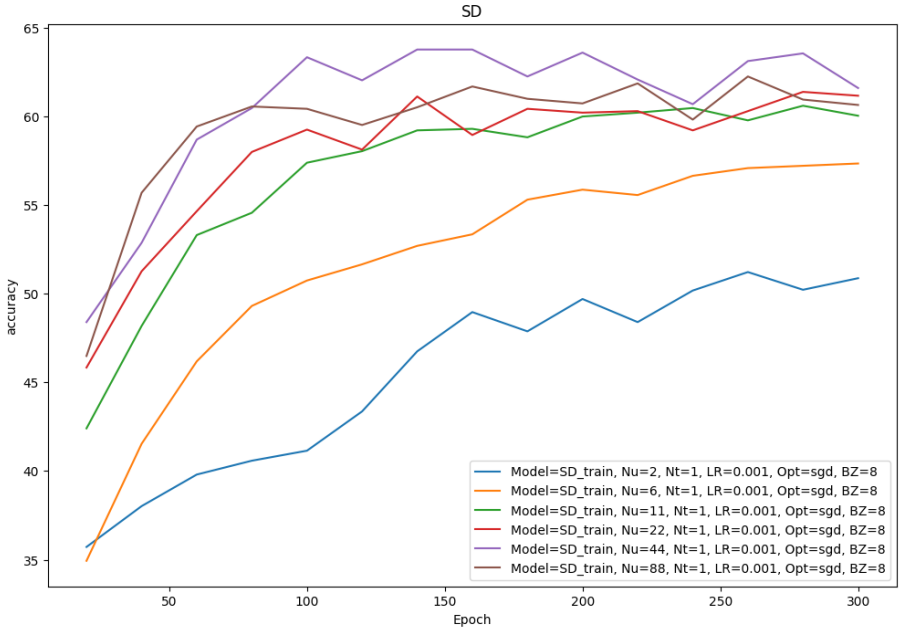


圖.10：batch_size=8、Nt=1、optimizer=SGD、learning_rate=0.001 在訓練時不同 Nu 的準確度

在圖.9 和圖.10 中可以看到，Nu 在一定的數量之後訓練到的結果都落在準確度 60 上下，但唯獨 Nu 為 2 時的效果不佳(如圖.8 所示)，此外在圖.8 和圖.10 也有發現在使用 SGD 作為 optimizer 的情況下 Nu 為 44、Nt 為 1 時訓練結果是最好的。

- 不同的 Nt

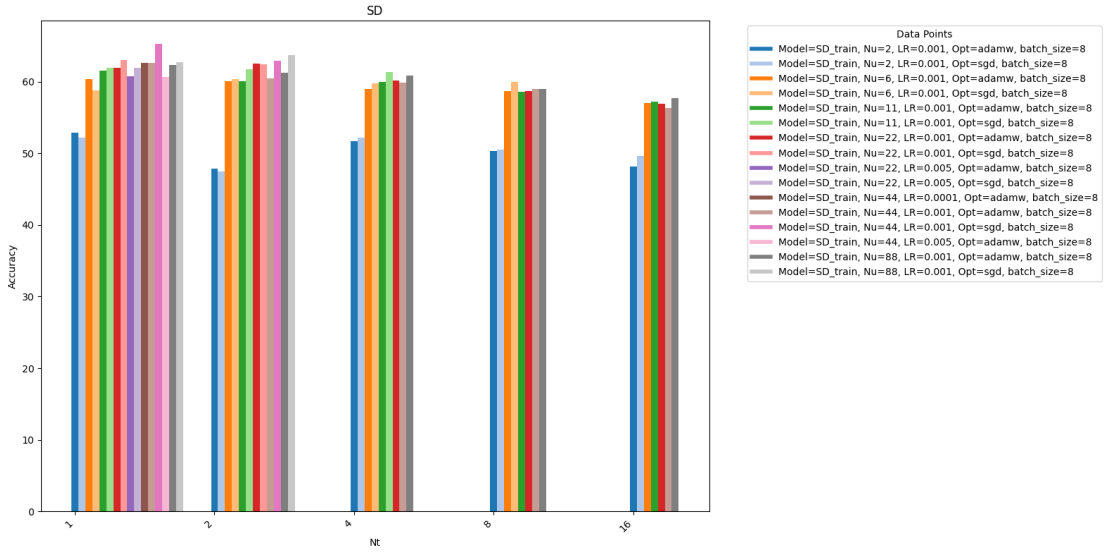


圖.11：batch_size=8 時不同參數的準確度

圖.12：batch_size=8、Nu=22、optimizer=adamw、learning_rate=0.001 在訓練時不同 Nt 的準確度

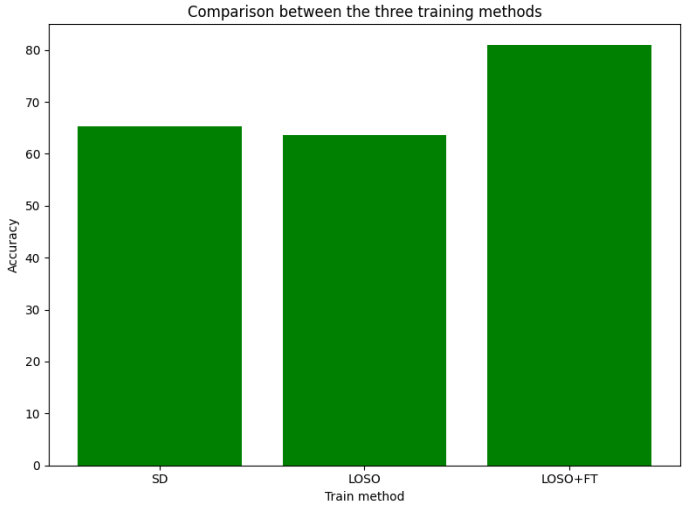
在上圖.12 中，Nt 為 1 或 2 時的準確度非常相近，但是隨著 Nt 數量的增加準確度也跟著一起下降。

對於 SD 訓練的總結，在 optimizer 的選擇上使用 SGD 時 batch 大小越低越好，反之選擇 adamw 的 batch 大小則是越高越好，在 learning rate 的比較下 learning rate 對 adamw 的影響是比較大的。Nu 和 Nt 的比較下，Nu 超過 11 之後的準確度都是差不多的，而 Nt 則是 1 和 2 測試出來的準確度

都差不多，但是隨著後續數字增加，訓練出來的效果就會越來越差。

Comparison between the three training methods

以下是根據 20 多萬組 epoch 的數據總結下來的三個訓練結果，其中 SD 有 7.2 萬組、LOSO 有 2.5 萬組、LOSO+FT 有 10 萬組，透過不斷調整上述有提到的參數得到的以下結果。



其中 SD 拿到最高的準確度為 65.23%、LOSO 為 63.54%、LOSO+FT 為 80.90%。

SD 使用的參數為 Nu=44、Nt=1、batch_size=8、learning_rate=0.001、optimizer=SGD。

LOSO 使用的參數為 Nu=22、Nt=1、batch_size=8、learning_rate=0.001、optimizer=AdamW

LOSO+FT 使用的參數為 Nu=22、Nt=1、batch_size=32、learning_rate=0.001、optimizer=adamw 挑選的 LOSO 原先訓練集為 Nu=22、Nt=1、batch_size=16、learning_rate=0.005、optimizer=SGD，在訓練一定準確度的基礎上轉移訓練，改成使用大的 batch size 和 adamw 的 optmizier。

Anything you want to mention

在做 LOSO-FT 訓練時，挑選在 training 時 loss 較高的會比較容易 train 起來，但是如果太高或者準確率太低，會導致最終準確率突破不了 80%，換句話說，挑選(針對測試集)準確度高的但是(訓練集)尚未收斂(loss 高)的記錄檔進行 finetune 的轉移訓練，效果會比已經收斂的記錄檔還要來的好。

Discussion

What is the reason to make the task hard to achieve high accuracy

根據 EEG 資料集的論文，輸入的資料有兩個 session，s1T 和 s1E，且這兩個 session 都在不同天去蒐集資料，以 subject 1 為例(s1T/s1E)，T 為 training 資料，E 則是 Evaluation 資料。每一個 session 中包含了 6 個偵測回合，每一回合(run)都需要分別偵測 4 種不同的動作(classes)：左手、右手、雙腳、舌頭，而每一個種類的動作都需要偵測 12 次試驗(trail)， 6 次 48 個試驗，總共是 288 個試驗。為了採集這些動作的特徵點，使用了 22 個電極貼片，每個貼片隨機採樣 438 個時間點的特徵值。

而每個受試者(subject)可能都有所不同，因此導致很難一次訓練出一個每個受試者都能準確偵測的模型，外在干擾的因素過多(包含兩個 session 於不同天的偵測問題)。

What can you do to improve the accuracy of this task?

透過 finetune 的方式先從訓練好的資料再次訓練，但是使用其他訓練集，再次訓練會達到更好得效果。

在訓練中，透過不同的調整參數，發現透過調整 Nu 的大小可以提升 model 對不同 subject 之間的辨識度，訓練得到的結果 Nu=22 比 Nu=2 時還要來的好。同樣的 Nt 在數值越小的情況訓練的效果越好。

也可以透過選擇不同的 optimizer 讓整體的準確度更高，也可以調整 learning rate 讓他訓練時嘗試收斂在更好的地方。