NYCU DL Lab2 – EEG classification

313552049 鄭博涵

1.Overview :

在本次 lab 中, 實作了 SCCNet 這個特殊的 CNN 架構.

SCCNet 是一種針對時間序列數據，特別是腦電波(EEG)數據進行分析和分類的深度學習模型。該模型旨在有效地從 EEG 記錄中提取時空特徵，以用於實時腦機接口系統和其他神經科學應用

2.implementation Details :

A. Details of training and testing code:

Training code:

```python
def train_model(model, train_loader, val_loader, epochs, criterion, optimizer, scheduler, device, save_path, model_save_path):
    best_val_loss = float('inf')  # 初始化最佳驗證損失
    train_losses = []  # 儲存每個 epoch 的訓練損失
    val_losses = []  # 儲存每個 epoch 的驗證損失
    for epoch in range(epochs):
        model.train()
        total_loss = 0
        for data, target in train_loader:
            data, target = data.to(device), target.to(device)
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        scheduler.step()

        # 打印訓練損失
        avg_train_loss = total_loss / len(train_loader)
        train_losses.append(avg_train_loss)  # 加入到train list中

        # 計算驗證損失
        val_loss, val_accuracy = validate_model(model, val_loader, criterion, device)
        val_losses.append(val_loss)  # 加入驗證損失

        print(f"Epoch {epoch + 1},Training Loss: {avg_train_loss},  Validation Loss: {val_loss}, Validation Accuracy: {val_accuracy:.2f}%")
        # 保存驗證損失最低時的模型
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            save_model(model, model_save_path)

    # 獲取最終測試準確率
    final_accuracy = test_model(model, test_loader, device)
    print(f"FT's Final accuracy: {final_accuracy:.2f}%")
    PlotLoss(train_losses, val_losses, save_path, final_accuracy)
```

- best_val_loss: 用於記錄最低的驗證損失。
- train_losses, val_losses: 用於儲存每個 epoch 的 train, validation loss的list
- For every batch of data, 將輸入和目標移動到設備上，並進行梯度歸零、前向傳播、計算損失、反向傳播和參數更新。
- 每個 epoch 後，更新學習率調度器(scheduler)
- validation_model 在驗證集上評估模型，計算損失和準確率。
- 如果當前的驗證損失低於最佳驗證損失，則保存模型。
- 並在train結束後，在test dataset上測試最終準確率。

Testing code:

```python
def test_model(model, test_loader, device):
    correct = 0
    total = 0
    with torch.no_grad():   # torch.no_grad: 表示不需計算梯度
        for data in test_loader:    # 逐批處理dataloader集 中的所有數據
            images, labels = data
            images, labels = images.to(device), labels.to(device)   # set to 指定的device
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)   # 1指的是在每一個輸出的向量中尋找最大值
            total += labels.size(0) # calculate總樣本數
            correct += (predicted == labels).sum().item()   # calculate正確預測數, sum(): 計算此陣列中True的總數, item(): tensor -> 轉成數字
    accuracy = 100 * correct / total
    # print(f"Accuracy of the model on the test images: {accuracy} % ")
    return accuracy
```

•correct 和 total 用於計算正確預測的數量和總樣本數。

•從test_loader提取測試數據，並移動到指定設備(CPU or GPU)

•模型進行前向傳播，獲取輸出。

•使用 torch.max 函數獲取每個樣本的預測標籤。

•累計總樣本數和正確預測的數量，最後計算準確率。

Set Parameter:

```python
def save_model(model, path):
    torch.save(model.state_dict(), path)

num_classes = 4
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_save_path = "./model/SCCNet_model.pth"
save_path = "./loss_plot.jpg"

model = load_model(model_save_path)
model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.88, weight_decay=1e-4)
# StepLR每隔10個epoch將學習率減半
scheduler = StepLR(optimizer, step_size=4, gamma=0.95)

train_dataset = MIBCI2aDataset(mode='FT_train')
val_dataset = MIBCI2aDataset(mode='FT_test')
test_dataset = MIBCI2aDataset(mode='FT_test')

train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=256, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=256, shuffle=False)



epochs = 300

train_model(model, train_loader, val_loader, epochs, criterion, optimizer, scheduler, device, save_path, model_save_path)
```

•以上這段代碼設置了模型訓練的所有必要參數

並啟動了模型的訓練過程

B. Details of the SCCNet:

```python
class SCCNet(nn.Module):
    def __init__(self, numClasses=4, timeSample=438, Nu=44, C=22, Nc=1, Nt=1, dropoutRate=0.5):
        super(SCCNet, self).__init__()

        Nf = Nu     # Nf是一個用來設定第二個卷積層的參數，論文上說 Nf 跟Nu相同
        # Spatio
        self.conv1 = nn.Conv2d(1, Nu, (C, Nt))  # 將輸入通道數 (1) 轉換為 Nu 個特徵圖，卷積核大小為 (C, Nt)
        self.bn1 = nn.BatchNorm2d(Nu)   # 因為conv1 輸出有Nu個
        self.drop = nn.Dropout(dropoutRate)
        # Spatio-temporal
        self.conv2 = nn.Conv2d(1, 20, (Nf, 12))  # 因為我們有permute, 所以不是(Nu, 20, (1, 12))
        self.bn2 = nn.BatchNorm2d(20)
        self.square = SquareLayer()

        # Temporal Smoothing
        self.drop2 = nn.Dropout(dropoutRate)
        self.avg_pooling = nn.AvgPool2d((1, 62), (1, 12))

        # Classifier
        self.fc = nn.Linear(620, numClasses, bias=True)
        # self.softmax = nn.Softmax(dim=1)


    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        # x = F.relu(x)
        x = self.drop(x)
        x = x.permute(0, 2, 1, 3)

        x = self.conv2(x)
        x = self.bn2(x)
        x = self.square(x)
        x = self.drop2(x)

        x = self.avg_pooling(x)
        x = torch.log(x + 1e-8)  # 避免 log(0)
        x = x.flatten(start_dim=1)

        x = self.fc(x)
        # x = F.softmax(x)
        return x
```

- 定義了模型的各個層，包括卷積層, 批次正規化層, Dropout 層, 自定義平方層, 平均池化層和全連接層
- 再透過Forward pass 依次通過以上所述的層
- 最後返回輸出結果

3. Analyze on the experiment results :

A. Discover during the training process

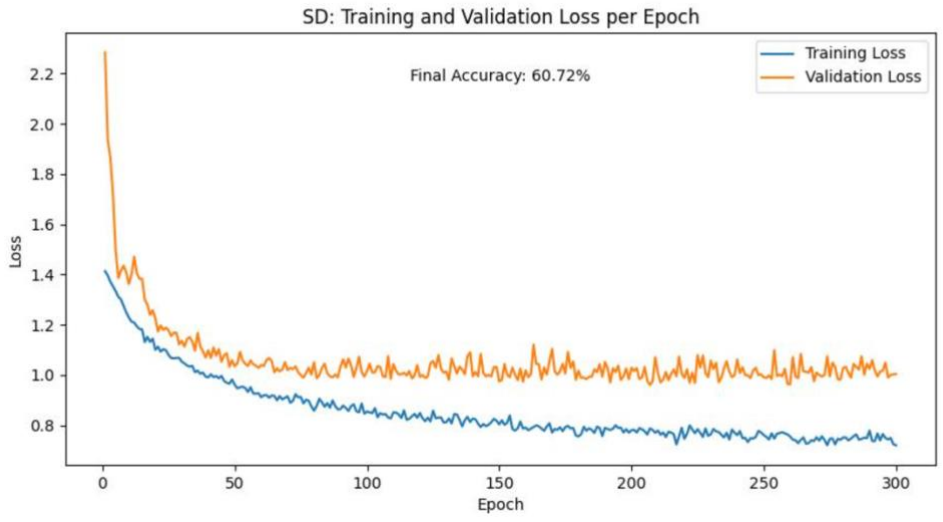
在 training 的過程中,會發現 FT_LOSO 比 LOSO 略快一些,應是因為 dataset 較小，然後發現到大概 300 epochs 時 loss 就無法再下降 (FT_LOSO 除外) 我覺得應該是因為資料集過於龐大，導致要提取的特徵太多，所以容易失真

另外，也很容易發生 training loss 有下降，但 validation loss 不會下降的 overfitting 的情況，應該是因為 EEG 資料具有高度變異性或資料龐大的關係

B. Comparison between the three training methods

SD:

```
Epoch 293,Training Loss: 0.7674913340144687,  Validation Loss: 0.9938572413391538, Validation Accuracy: 61.11%
Epoch 294,Training Loss: 0.7397429545720419,  Validation Loss: 1.0141334980726242, Validation Accuracy: 60.68%
Epoch 295,Training Loss: 0.7651759816540612,  Validation Loss: 1.0171751578648884, Validation Accuracy: 61.07%
Epoch 296,Training Loss: 0.7497952613565657,  Validation Loss: 1.049874362018373, Validation Accuracy: 60.07%
Epoch 297,Training Loss: 0.7438059581650628,  Validation Loss: 0.9898705830176672, Validation Accuracy: 60.59%
Epoch 298,Training Loss: 0.7493921087847816,  Validation Loss: 1.0010250293546252, Validation Accuracy: 60.46%
Epoch 299,Training Loss: 0.7245782216389974,  Validation Loss: 1.0022651553153992, Validation Accuracy: 61.46%
Epoch 300,Training Loss: 0.7203771902455224,  Validation Loss: 1.0038651261064742, Validation Accuracy: 60.72%
SD's Final accuracy: 60.72%
```

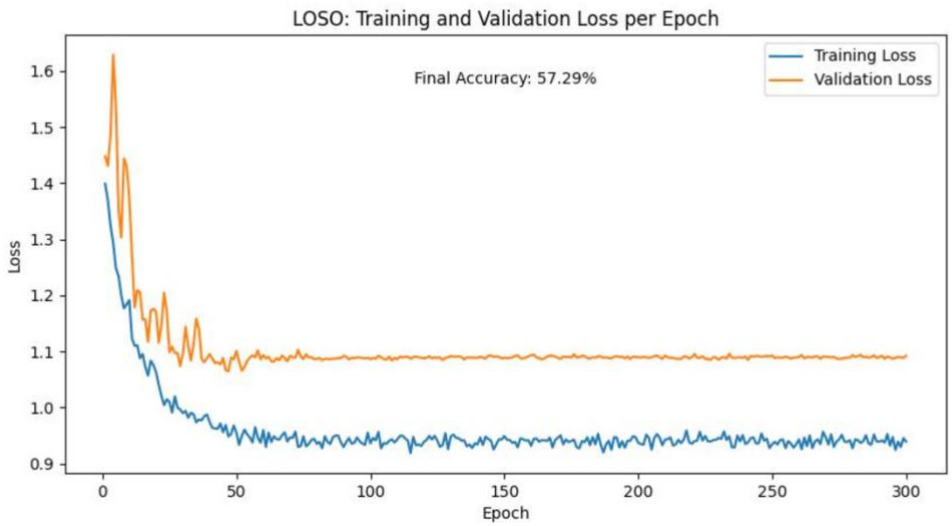


訓練數據和測試數據來自同一個數據集，但有不同的分割

結果：訓練損失和驗證損失的差距不大，表明模型沒有嚴重的過擬合或欠擬合

LOSO:

```
Epoch 293,Training Loss: 0.9528183247894049,  Validation Loss: 1.0909703572591145, Validation Accuracy: 57.99%
Epoch 294,Training Loss: 0.932375593110919,  Validation Loss: 1.0895335872968037, Validation Accuracy: 57.64%
Epoch 295,Training Loss: 0.9476926792412996,  Validation Loss: 1.0868925253550212, Validation Accuracy: 57.64%
Epoch 296,Training Loss: 0.9244999960064888,  Validation Loss: 1.0888315439224243, Validation Accuracy: 57.99%
Epoch 297,Training Loss: 0.9393489640206099,  Validation Loss: 1.0900355378786724, Validation Accuracy: 57.64%
Epoch 298,Training Loss: 0.9300669599324465,  Validation Loss: 1.0894986788431804, Validation Accuracy: 57.64%
Epoch 299,Training Loss: 0.9454845860600471,  Validation Loss: 1.0885838667551677, Validation Accuracy: 57.64%
Epoch 300,Training Loss: 0.9394580963999033,  Validation Loss: 1.0926541686058044, Validation Accuracy: 57.29%
LOSO's Final accuracy: 57.29%
```

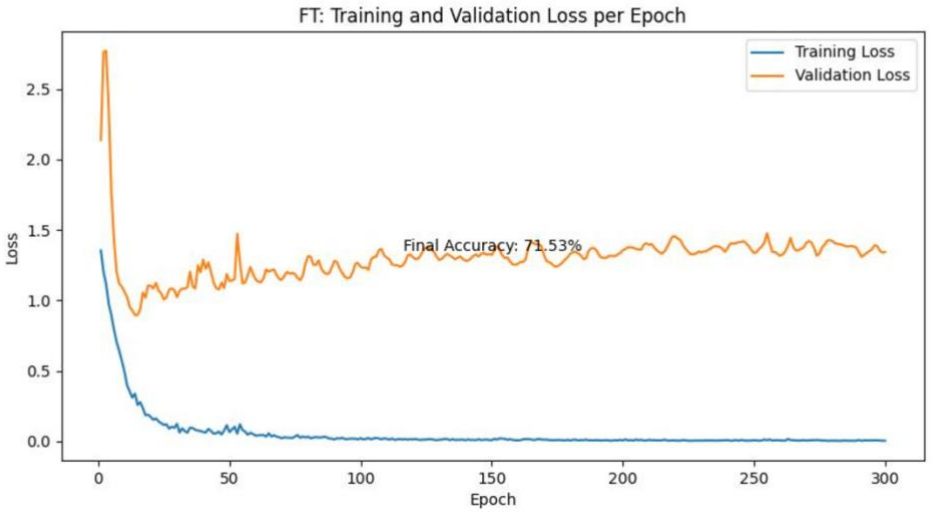


LOSO 是一種交叉驗證技術，每次訓練時，將一個受試者的數據用作測試集，其他受試者的數據用作訓練集

結果：　驗證損失顯示出一些波動，表明模型在不同受試者之間的一致性可能有所不同

FT_LOSO:

```
Epoch 293,Training Loss: 0.005748959568639596,  Validation Loss: 1.3356823921203613, Validation Accuracy: 69.44%
Epoch 294,Training Loss: 0.0054252474413563805,  Validation Loss: 1.349038044611613, Validation Accuracy: 69.44%
Epoch 295,Training Loss: 0.006808707645783822,  Validation Loss: 1.3621454238891602, Validation Accuracy: 68.75%
Epoch 296,Training Loss: 0.006051863543689251,  Validation Loss: 1.3898215691248577, Validation Accuracy: 70.14%
Epoch 297,Training Loss: 0.007088072209929426,  Validation Loss: 1.3867474993069966, Validation Accuracy: 70.49%
Epoch 298,Training Loss: 0.005717223820586999,  Validation Loss: 1.3530832926432292, Validation Accuracy: 70.49%
Epoch 299,Training Loss: 0.0036919547710567713,  Validation Loss: 1.3382391333580017, Validation Accuracy: 70.83%
Epoch 300,Training Loss: 0.004002504671613376,  Validation Loss: 1.3434767723083496, Validation Accuracy: 71.53%
FT's Final accuracy: 71.53%
```



使用一個預訓練的模型，並在特定任務上進行微調。這通常有助於提高模型在新數據上的性能

結果: 訓練損失很低, 但驗證損失高很多，-> overfitting 情況較嚴重

4.Discussion :

A.What is the reason to make the task hard to achieve high accuracy?

我覺得是因為 EEG 資料具有高度變異性 和 複雜性

如果這些資料是來自不同受試者, 在執行 BCI 任務時的表現

差異顯著，而這種跨受試者的變異性使得模型很難在所有受試者上都保持高準確度，即便是同一個體，在不同時間點的資料也可能會有所不同

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

Validation loss 跟 training loss 要降下來，最後的 accuracy 才會高，我透過更改學習率大小，將靜態學習率改為動態學習率，嘗試不同的 optimizer, batch_size 的大小，且將論文所提到的最後一層要加 softmax 給拔掉，也嘗試過是否加 ReLU，利用以上這些方法來有效降低我的 loss

並透過調 drop_rate，增加一層的 drop layer，
加入 L2 正則化(weight_decay) 來防止 overfitting 的現象