DLP_Lab2_313553024_蘇柏叡

1. Overview

本次Lab主要是實作SCCNet(**Fig 1**)，包含兩個卷積層**分別提取空間**、**時間與空間之特徵**，並使用平均池化層對時間做平滑處理，最終分類成四種類別(以Softmax進行分類)。並且分別使用**SD**、**LOSO**、**LOSO_FT**三種方法針對不同資料集對模型進行評出與測試。其中SD、LOSO、LOSO_FT的方法比較整理如下表(含實驗最佳準確度):

方法/準確度	方法內容與敘述	優點	缺點與潛在風險
SD/61.46%	模型使用了**包括**測試對象的部份數據進行訓練。	有助於提高對該測試對象的分類準確度(**個體適應性強**)。	模型對於其他受試者有較差的準確度，意即模型難以泛化。
LOSO/60.42%	使用**除了**測試對象的其他所有對象的數據進行訓練。	可以**保有泛化能力**，不會只學習到部分特徵，也不至於過擬合。	通常準確度較不高，因為若是個體落差過大會無法適應該受試者差異較大的特徵。
LOSO_FT/73.26%	使用**除了**測試對象的其他所有對象的數據進行訓練之外，並**使用測試對象的部分數據**對模型進行**微調**	保有泛化能力並透過Finetuning提高對測試者的個體適應性。	還是存在過擬合的問題，尤其是資料筆數較少之情況。

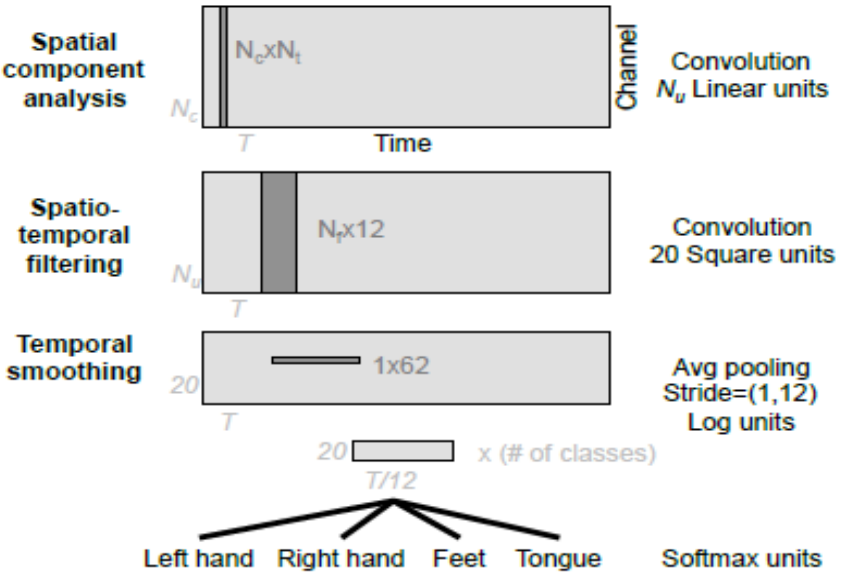


Fig. 1. The example architecture of SCCNet.

2. Implementation Details

- Details of training and testing code

a. trainer.py程式碼如下:

```python
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from Dataloader import MIBCI2aDataset  # Ensure the data loader matches your project structure
from model.SCCNet import SCCNet        # Ensure the model path is correct
from utils import plot_metrics         # Ensure this function exists in utils.py


# Data Augmentation
def data_augmentation(data):
    # Randomly add Gaussian noise
    noise = torch.randn_like(data) * 0.02
    data = data + noise

    # Random time shift
    shift = np.random.randint(-5, 5)
    data = torch.roll(data, shifts=shift, dims=-1)

    return data
```

說明: 引入相關之套件，並且實作資料擴增(**實際訓練並未採用**)

```python
# Training function
def train(model, device, train_loader, valid_loader, optimizer, criterion, scheduler, epochs=300):
    best_accuracy = 0.0
    train_losses, valid_losses, train_accuracies, valid_accuracies = [], [], [], []
    for epoch in range(epochs):
        model.train()
        total_loss, total, correct = 0, 0, 0
        for data, target in train_loader:
            data, target = data.to(device), target.to(device)
            data = data_augmentation(data)
            optimizer.zero_grad() #清除之前的梯度
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step() #update model parameters

            total_loss += loss.item()
            _, predicted = output.max(1)#輸出機率最高的class
            total += target.size(0)
            correct += predicted.eq(target).sum().item()
```

```
    train_loss = total_loss / len(train_loader)
    train_accuracy = 100. * correct / total
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)

    valid_loss, valid_accuracy = validate(model, device, valid_loader, criterion)
    valid_losses.append(valid_loss)
    valid_accuracies.append(valid_accuracy)
    print(f'Epoch {epoch+1}: Train Loss: {train_loss:.4f}, Valid Loss: {valid_loss:.4f},
          Train Acc: {train_accuracy:.2f}%, Valid Acc: {valid_accuracy:.2f}%,
          LR: {scheduler.optimizer.param_groups[0]["lr"]:.6f}')
    if valid_accuracy > best_accuracy:
        best_accuracy = valid_accuracy
        # Save the best model based on accuracy
        #save_model(model, './best_sccnet_model_SD.pth')
        save_model(model, './best_sccnet_model_LOSO.pth')
        #save_model(model, './best_sccnet_model_LOSO_FT.pth')
        print(f'Saved Best Model with Accuracy: {best_accuracy:.2f}%')

    scheduler.step(valid_loss)

plot_metrics(train_losses, valid_losses, train_accuracies, valid_accuracies)
```

說明： 計算train、valid的loss、accuracy，並且保留訓練過程中最高的valid_accuracy那個Epoch存為權重。

```python
# Validation function
def validate(model, device, valid_loader, criterion):
    model.eval()
    valid_loss, total, correct = 0, 0, 0
    with torch.no_grad():
        for data, target in valid_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            valid_loss += criterion(output, target).item()
            _, predicted = output.max(1)
            total += target.size(0)
            correct += predicted.eq(target).sum().item()

    accuracy = 100. * correct / total
    return valid_loss / len(valid_loader), accuracy

# Function to save the model
def save_model(model, path):
    torch.save(model.state_dict(), path)
    print(f'Model saved to {path}')
```

說明：這邊使用的是對應的測試資料集(和tester.py的test內容相同)之所以不使用梯度是因為在評估模型時，不必更新參數。然後底下是儲存模型權重檔案的函式。

```python
def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # Instantiate datasets and data loaders
    train_dataset = MIBCI2aDataset(mode='train')
    valid_dataset = MIBCI2aDataset(mode='test')  # Assume using test set as validation set
    train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
    valid_loader = DataLoader(valid_dataset, batch_size=64, shuffle=False)

    # Initialize model
    model = SCCNet(num_classes=4, input_channels=22, Nu=22, Nt=1, dropout_rate=0.5,weight_decay=1e-4)
    model = model.to(device)

    criterion = nn.CrossEntropyLoss()
    #criterion = nn.MSELoss()
    optimizer = optim.Adadelta(model.parameters(),lr=1,rho=0.9, eps=1e-6)  # Use AdaDelta optimizer
    #optimizer = optim.Adam(model.parameters(),lr=0.0001,weight_decay=1e-5)
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=10)

    print("Training Model")
    train(model, device, train_loader, valid_loader, optimizer, criterion, scheduler, epochs=300)

if __name__ == '__main__':
    main()
```

說明：使用Cross Entropy計算損失函數，並且使用AdaDelta做為優化器，rho用於調節梯度平方的移動平均的衰減率，eps為一個很小的常數，加到分母上以防止在執行參數更新時出現除以零的情況。其中設置scheduler當連續十個epoch，val_loss沒有下降時會*factor調低學習率。

b. trainer.py程式碼如下：

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from Dataloader import MIBCI2aDataset
from model.SCCNet import SCCNet

def test(model, device, test_loader, criterion):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += criterion(output, target).item()
            _, predicted = output.max(1)
            correct += predicted.eq(target).sum().item()

    test_loss /= len(test_loader)
    accuracy = 100. * correct / len(test_loader.dataset)
    print(f'Test set: Average loss: {test_loss:.4f}, Accuracy: {correct}/{len(test_loader.dataset)} ({accuracy:.2f}%)')
    return test_loss, accuracy
```

說明：之所以不使用梯度是因為在評估模型時，不必更新參數。然後底下是儲存模型權重檔案的函式。

```python
def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    criterion = nn.CrossEntropyLoss()

    print("Select the model to test:")
    print("1. Subject Dependent (SD)")
    print("2. Leave-One-Subject-Out (LOSO)")
    print("3. Fine-Tuning (FT)")
    choice = input("Enter your choice (1, 2, or 3): ")

    if choice == '1':
        sd_test_dataset = MIBCI2aDataset(mode='test')
        sd_test_loader = DataLoader(sd_test_dataset, batch_size=64, shuffle=False)

        model_sd = SCCNet(num_classes=4, input_channels=22, Nu=22, Nt=1, dropout_rate=0.5, weight_decay = 1e-4)
        model_sd.load_state_dict(torch.load('./best_sccnet_model_SD_6146.pth', map_location=device))
        model_sd = model_sd.to(device)

        print("Testing Subject Dependent Model")
        test(model_sd, device, sd_test_loader, criterion)
```

```
    elif choice == '2':
        loso_test_dataset = MIBCI2aDataset(mode='test')
        loso_test_loader = DataLoader(loso_test_dataset, batch_size=64, shuffle=False)

        model_loso = SCCNet(num_classes=4, input_channels=22, Nu=22, Nt=1, dropout_rate=0.5,weight_decay = 1e-4)
        model_loso.load_state_dict(torch.load('./best_sccnet_model_LOSO(60.42new).pth', map_location=device))
        model_loso = model_loso.to(device)

        print("Testing LOSO Model")
        test(model_loso, device, loso_test_loader, criterion)

    elif choice == '3':
        ft_test_dataset = MIBCI2aDataset(mode='test')
        ft_test_loader = DataLoader(ft_test_dataset, batch_size=64, shuffle=False)

        model_ft = SCCNet(num_classes=4, input_channels=22, Nu=22, Nt=1, dropout_rate=0.5,weight_decay = 1e-4)
        model_ft.load_state_dict(torch.load('./best_sccnet_model_LOSO_FT(73.26new).pth', map_location=device))
        model_ft = model_ft.to(device)

        print("Testing Finetune Model")
        test(model_ft, device, ft_test_loader, criterion)

    else:
        print("Invalid choice. Please run the program again and select a valid option.")

if __name__ == '__main__':
    main()
```

說明： 這裡主函式可以選擇要測試的三種方法任一，load進來的都是train好且保存的最佳權重檔案(已在後面標示準確度以方便demo)。

- Details of the SCCNet

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class SquareActivation(nn.Module):
    def forward(self, x):
        return x ** 2

class SCCNet(nn.Module):
    def __init__(self, num_classes = 4, input_channels = 22, Nu = 22, Nt = 1, dropout_rate = 0.5, weight_decay = 1e-4):
        super(SCCNet, self).__init__()
        self.num_classes = num_classes

        # First convolution block for spatial component analysis
        '''
        - input channel : 1
        - output channel : Nu
        - (input_channels, Nt) : conv kernel，前者為spatial filter，後者(Nt)為時間dimension上kernel大小
        - apply 0 padding : 因為Nt = 1所以沒做padding沒差
        '''
        self.conv1 = nn.Conv2d(1, Nu, (input_channels, Nt), padding=(0, 0), bias=True)
        self.batchnorm1 = nn.BatchNorm2d(Nu)#對每個feature map做normalize
        self.dropout1 = nn.Dropout(dropout_rate)

```

本圖為 Square Activation、SCCNet基本建構、第一個Convolution Block(for spatial component analysis)介紹:

1. 首先建立Square Activation，即將輸入的訊號(x)去平方。

2. 接著架SCCNet，建構的需要放**num_classes(4個類別)**，**輸入的channel數量**，Nu、Nt，以及論文提到的**drop out設0.5**，以及**L2正則化設1e-4**。

3. 模型的第一個convolution layer輸入的有(**input channel = 1**、**Nu**、**conv kernel(前者是空間濾波，後者是時間維度的kernel大小)**、padding部分因為第一層在空間和時間維度上**皆是1，所以不用做padding**、此外我們也將bias項打開來)

4. 接著對每個feature map做正規化，並且Drop out。

```
        # Second convolution block for spatio-temporal filtering
        '''
        - input channel : Nu
        - output channel : 20 square units
        - (1,12) : conv kernel，其實是Nt * 12
        - apply 0 padding : 因為空間濾波Nt = 1所以沒做padding沒差。
        後者因為kernel大小為12，合理的padding是(12-1) / 2 = 5.5 => 6
        - 在這邊使用square activation
        '''
        self.conv2 = nn.Conv2d(Nu, 20, (1, 12), padding = (0, 6), bias = True)
        self.batchnorm2 = nn.BatchNorm2d(20)
        self.square_activation = SquareActivation()
        self.dropout2 = nn.Dropout(dropout_rate)

        # Pooling block for temporal smoothing
        self.avg_pool = nn.AvgPool2d((1, 62), stride=(1, 12))
```

本圖為第二個Convolution Block (for spatio-Temporal Filtering)、使用Average Pooling之介紹:

1.模型的第二個convolution layer輸入的有(**Nu(即上層輸出項轉input)**、**output_channel論文設定20**、**conv kernel**(前者是空間濾波，後者是時間維度的kernel大小，分別為1,12)、**padding**部分因為第二層在空間濾波部分***Nt = 1(不用padding)，時間維度12，根據公式至少要zero Padding 左右各6***、此外我們也將bias項打開來)

2. 接著將其作正規化、使用Square Activation將提取的訊號值平方並Drop out。

3.最後，根據論文內容使用平均池化層大小為(1,62)，步長為(1,12)進行實現。

```
        # Compute the number of features to input to the fully connected layer
        '''
        - dummy_input(batch_size,sample channel = 1, input_channel, time_steps)
        - time_steps: 125Hz * (4-0.5)s = 437.5 => 438
        - self.num_features: 計算全連接層的features數量
        - nn.Linear(input_feature, output_feature) 前者好理解就是輸入的特徵，後者就是分成四類。
        '''
        dummy_input = torch.zeros(64,1, input_channels, 438)
        self.num_features = self._get_fc_input_features(dummy_input)
        self.fc = nn.Linear(self.num_features, num_classes)

        self.weight_decay = weight_decay
```

本圖為使用dummy input測試模型介紹:

1. time_steps根據論文內容是0.5-4s間以125Hz頻率，故會有437.5=>438個時間點。

2. 透過_get_fc_input_features計算需要的輸入特徵數量並用self.num_features去接。接著就是拿輸入的特徵分成論文說的四個類別: Left hand、Right hand、Feet、Tongue

```
    def _get_fc_input_features(self, x):
        x = self.conv1(x)
        x = self.batchnorm1(x)
        x = F.elu(x)
        x = self.dropout1(x)
        x = self.conv2(x)
        x = self.batchnorm2(x)
        x = self.square_activation(x)
        x = self.dropout2(x)
        x = self.avg_pool(x)
        x = torch.flatten(x, start_dim=1) #x = (batch_size, channels, height, width)把後三者推平成一個vector
        return x.size(1)#輸出即為c,h,w被flatten後的東東
```

本圖說明:

1. 訊號輸入到第一個捲積層之後再對其特徵圖做normalization接著使用elu激活函數(因為比較relu、tanh、leaky relu後發現elu效果較佳，但原論文未提起使用何者)，之後做Drop out。接下來經過第二個捲積層再對其特徵圖做做normalization，接著使用Square Activation再經過Drop out輸出。

2. 使用平均池化層並且flatten，把c,h,w推平成一個vector，最終再輸出。

```
67
68      def forward(self, x):
69          x = self.batchnorm1(self.conv1(x))
70          x = F.elu(x)
71          x = self.dropout1(x)
72          x = self.batchnorm2(self.conv2(x))
73          x = self.square_activation(x)
74          x = self.dropout2(x)
75          x = self.avg_pool(x)
76          x = torch.flatten(x, start_dim=1)
77          x = self.fc(x)
78          return F.log_softmax(x, dim=1)
```

本圖說明:

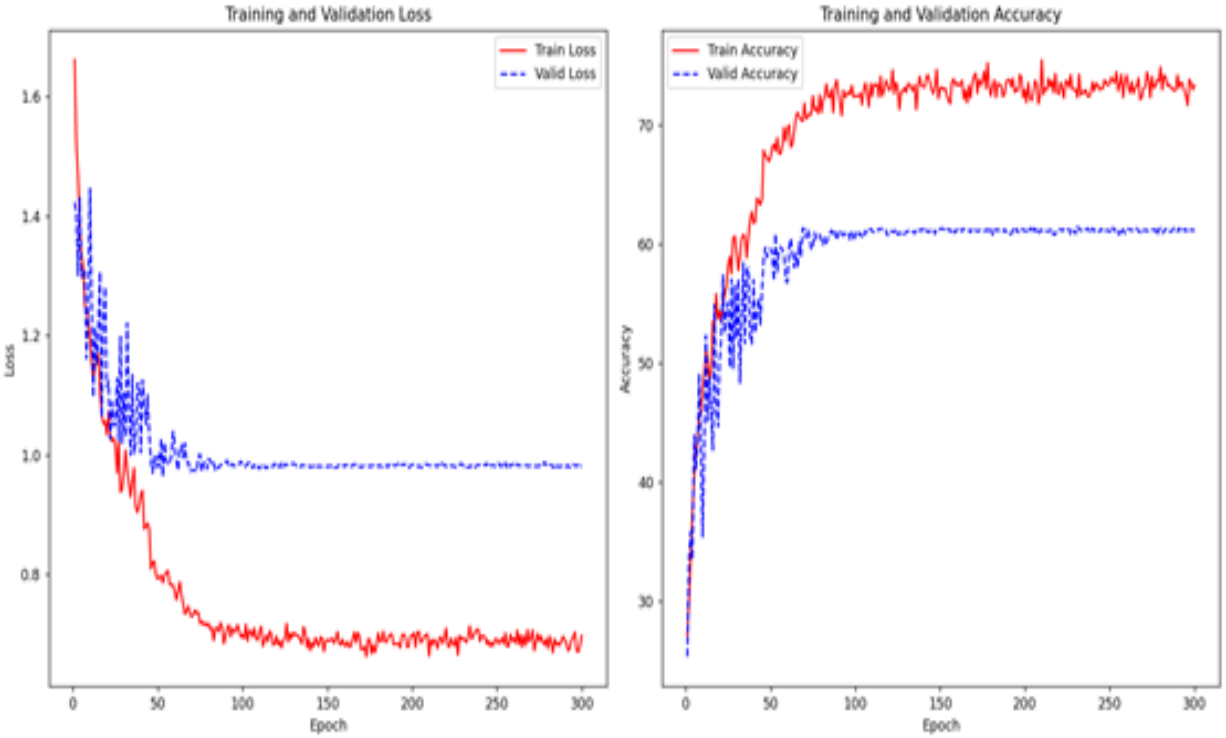
為前項傳遞當batchnorm1收到從第一個捲積層輸出的東西後開始傳遞，接著就和上面的順序一樣(不再多做說明)，最後是使用Log_softmax進行分類。

3. Analyze on the experiment results

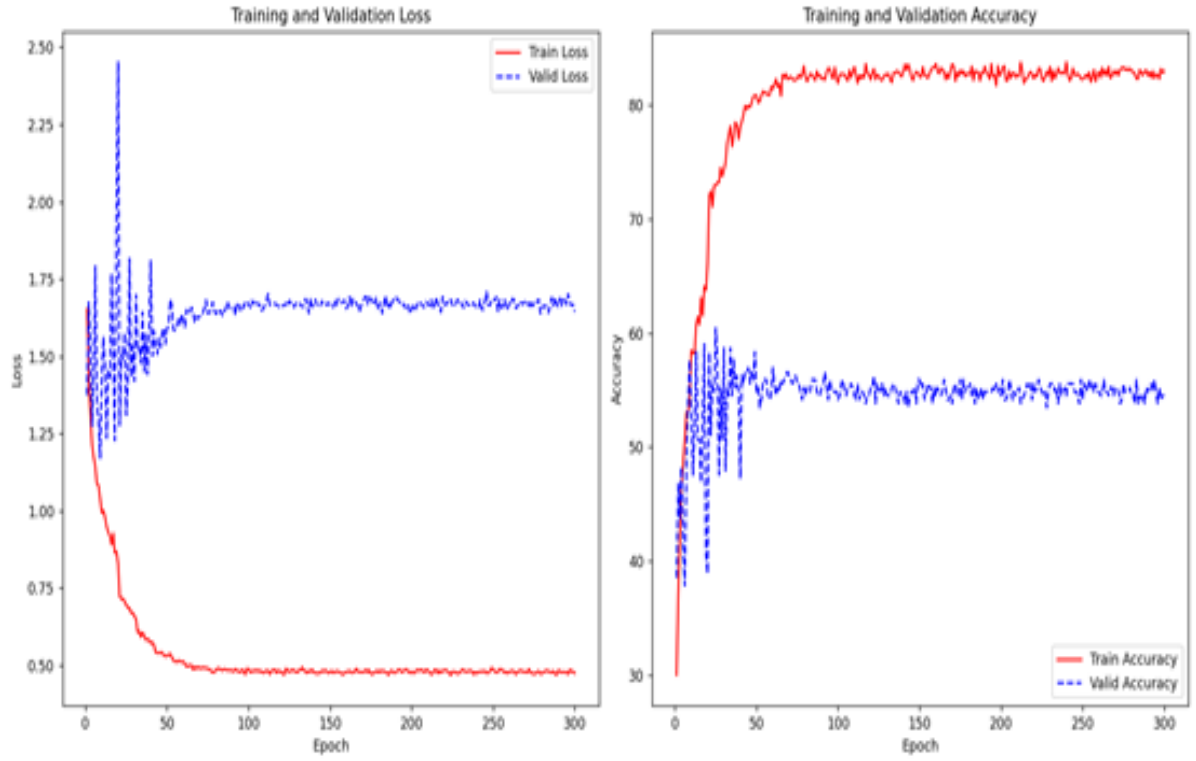
▪ Discover during the training process

1. 羅列三種方法的loss、accuracy curve

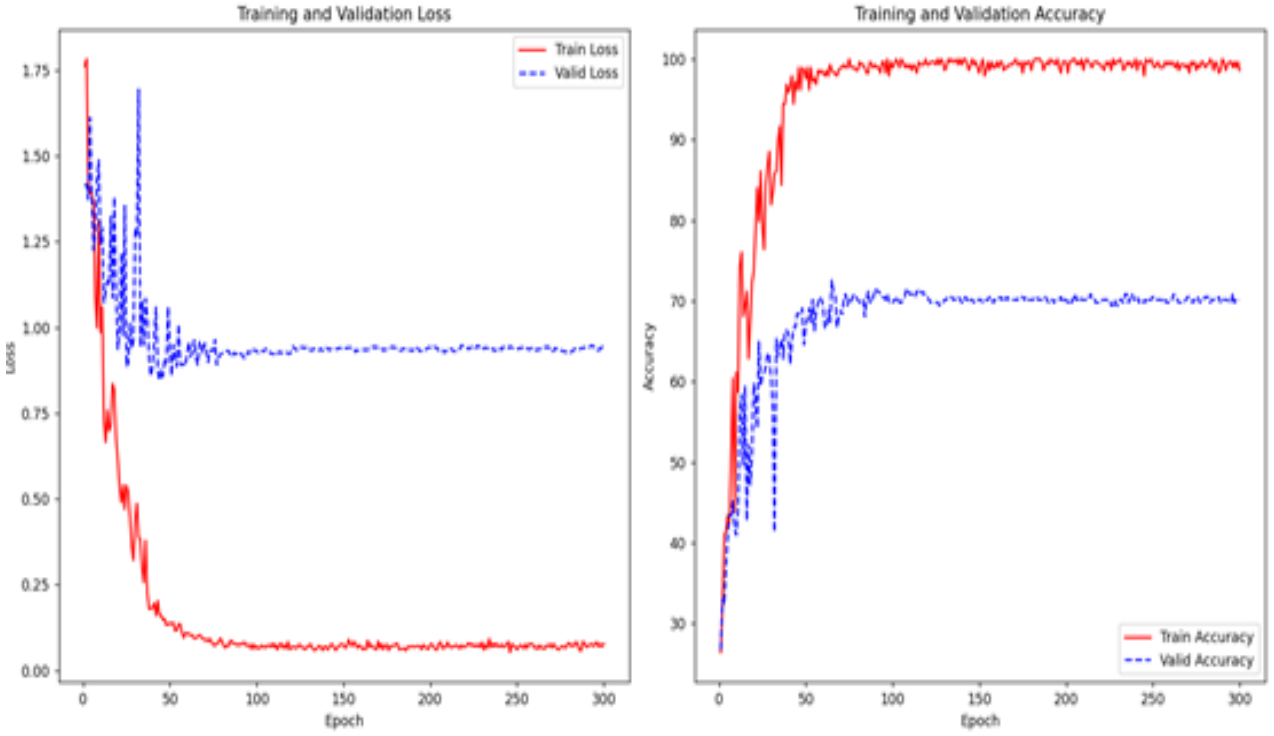
a. SD:



b. LOSO:



c. LOSO_FT:



2. 發現:

(1) **以測試集的準確度而言**，當訓練**Epoch大約落在50-100之間**時，**SD、LOSO_FT在測試時有最高的**Accuracy，而LOSO方法則是約莫在**Epoch接近50時有最高的**Accuracy。隨後準確度會略為下降並且不再提升。

(2) **以訓練集的準確度而言**，當訓練Epoch大約落在50-100之間時，三種方法皆會逐漸達到較高值，分別是75%、83%、99%，且有逐步緩慢上升之趨勢(雖有震盪)，這也顯示出當訓練集的準確度仍有提升但測試集準確度不再提升，意即有出現OverFitting之情形。

(3) **以損失函數曲線圖來看**，SD、LOSO_FT在Epoch50-100時，會逐漸趨於平緩不再有幅度地降低(而是會微幅震盪)，然而在LOSO方法中測試集的Loss值在Epoch前期較低(但震盪幅度也相當大)，而在Epoch = 50左右時，Loss value往上回彈並趨於平滑，這代表**模型較不穩定**。然而，在三種方法中在訓練集的部分則是皆能有效的降低Loss值。

(4) 小結論: 在三種方法中，**LOSO_FT是呈現overfitting最為明顯的，因為訓練與測試集資料筆數均只有288，故容易過早造成過擬合**，使得訓練集準確度極高，但移到測試集時就有明顯落差(雖然測試集準確度可以來到73.26%)。而在LOSO方法中，測試集最佳準確度有站上60%，但隨著後期Loss value回彈，準確度在後期大部分在55%左右徘徊。另外，在SD方法中，訓練期數在100左右後，準確度在60、61%來回微幅震盪。

▪ Comparison between the three training method

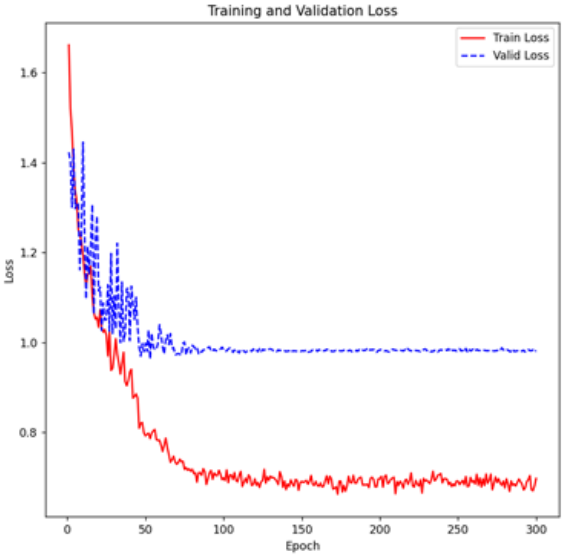
(1) SD

a. Testing Result:

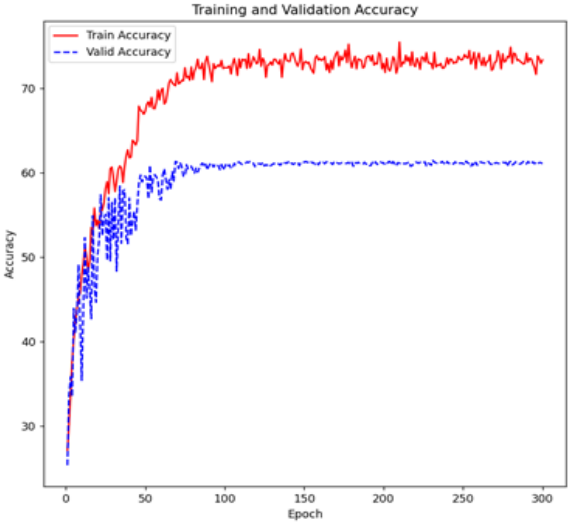
Testing Loss(average): 0.9793、Testing Accuracy: 61.46 %

```
Lab2_313553024_蘇柏叡/lab2/tester.py
Select the model to test:
1. Subject Dependent (SD)
2. Leave-One-Subject-Out (LOSO)
3. Fine-Tuning (FT)
Enter your choice (1, 2, or 3): 1
Testing Subject Dependent Model
Test set: Average loss: 0.9793, Accuracy: 1416/2304 (61.46%)
```

b. Loss Plot(Includes training and validation)



c. Accuracy Plot(Includes training and validation)



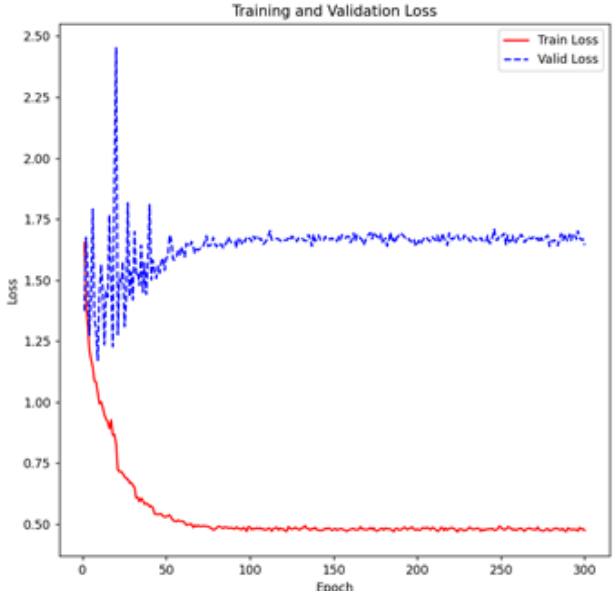
(2) LOSO

a. Testing Result:

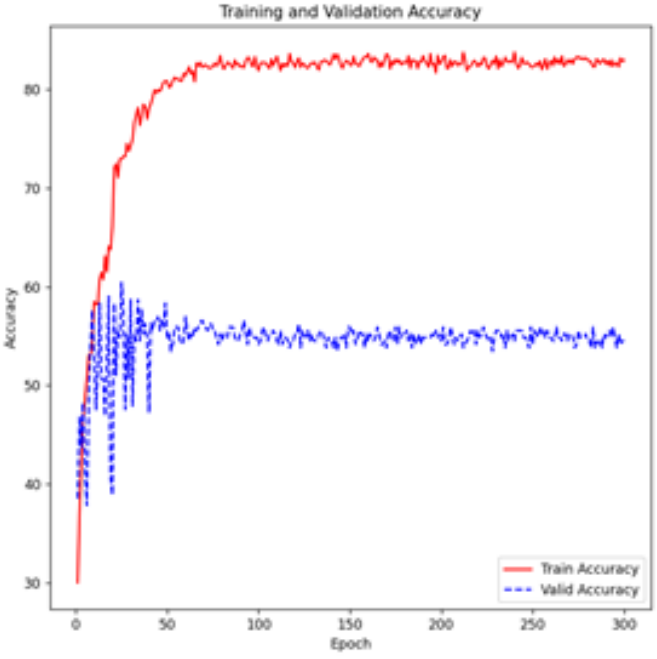
Testing Loss(average): 1.1741、Testing Accuracy: 60.42 %

```
Lab2_313553024_蘇柏叡/lab2/tester.py
Select the model to test:
1. Subject Dependent (SD)
2. Leave-One-Subject-Out (LOSO)
3. Fine-Tuning (FT)
Enter your choice (1, 2, or 3): 2
Testing LOSO Model
Test set: Average loss: 1.1741, Accuracy: 174/288 (60.42%)
```

b. Loss Plot(Includes training and validation)



c. Accuracy Plot(Includes training and validation)



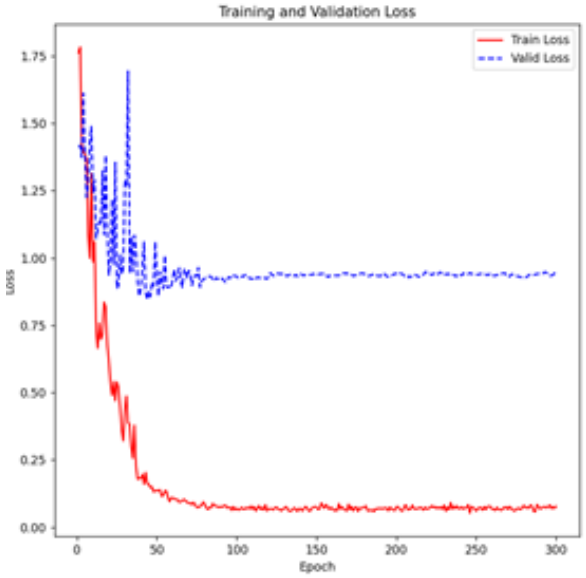
(3)LOSO_FT

a. Testing Result:

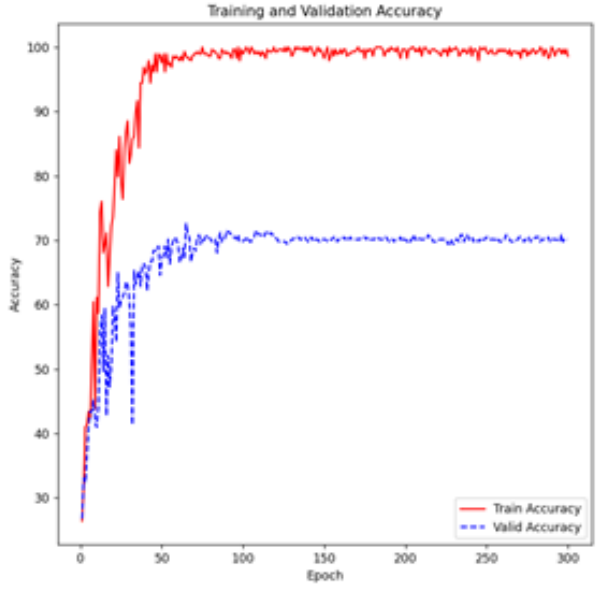
Testing Loss(average): 0.9761、Testing Accuracy: 73.26 %

```
Lab2_313553024_蘇柏叡/lab2/tester.py
Select the model to test:
1. Subject Dependent (SD)
2. Leave-One-Subject-Out (LOSO)
3. Fine-Tuning (FT)
Enter your choice (1, 2, or 3): 3
Testing Finetune Model
Test set: Average loss: 0.9761, Accuracy: 211/288 (73.26%)
```

b. Loss Plot(Includes training and validation)



c. Accuracy Plot(Includes training and validation)



4. Discussion

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

(1) 訊號資料本質上就**比較複雜且非靜態的**(且需考量時間維度)，因此要有效地從訊號中提取特徵本質上就不是那麼容易，所以在分類上有一定的困難性(**如右手、左手可能就不太好分**)。

(2) 不同受試者之間的訊號存在顯著差異，這種個體間和個體內的變異性使得模型要能泛化具有相當程度的困難。

(3) 模型學習上容易過擬合，且無法在測試集的地方有效持續降低損失值並提高其準確度。

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

(1) Data Augmentation

前者為在資料添加高斯噪音，讓模型學習到更多元的特徵，以提高模型泛化能力，減少過擬合的可能性。

```python
# Data Augmentation
def data_augmentation(data):
    # Randomly add Gaussian noise
    noise = torch.randn_like(data) * 0.02
    data = data + noise

    # Random time shift
    shift = np.random.randint(-5, 5)
    data = torch.roll(data, shifts=shift, dims=-1)

    return data
```

(2) 再調參，可以嘗試調整不同參數，如:

a. Drop_out、weight_decay可以進行調整

b. 增加、調整模型複雜度(嘗試不同激活函數tanh、relu、leaky relu等)、使用不同尺寸、步伐大小的平均池化層。

c. 嘗試使用不同優化器，如SGD、NAdam、AdamW…等，並且更靈活的在訓練過程中調整學習率。