NYCU DL Lab2 – Motor Imagery Classification

Student ID:313553014 Name:廖怡誠

1. Overview

In lab2, we are required to implement the SCCNet designed in the paper “Spatial component-wise convolutional network (SCCNet) for motor-imagery EEG classification.”. Additionally, we need to employ three methods from the paper, namely SD, LOSO, and Fine-tuned LOSO, training the model with the BCI Competition IV 2a dataset, which consists of four classes of EEG signals, to achieve the specified accuracy and analyze the experimental results.

2. Implementation Details

A. Details of training and testing code

Figure 1 illustrates the preprocessing steps prior to training. Initially, the train and test data loaders are obtained, and the SCCNet is created with the specified hyperparameters. Once the loss function and optimizer are set, the training begins. Figure 2 demonstrates the process training, which includes forward pass, backward pass, and weight updating. Additionally, I utilized the zero_grad() function before the forward pass to prevent gradients from mixing and accumulating with other batches. Finally, I record the accuracy and loss during training for visualization purposes.

```python
def train(args):

    # check current device is cpu or gpu
    device = get_device()
    # processing dataset to dataloader
    train_dataloader = get_dataloader(method= args.method, phase= "train", batch_size= args.batch_size)
    test_dataloader = get_dataloader(method= args.method, phase= "test", batch_size= args.batch_size)

    model = SCCNet(C= args.channel, Nu= args.Nu, Nc= args.Nc, Nt= args.Nt, dropoutRate= args.dropoutRate).to(device)
    # pre-trained model
    if args.pre_trained != "":
        model.load_state_dict(torch.load(args.pre_trained, weights_only= True))
    # check model total parameters
    show_model(model)

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr= args.lr, weight_decay= args.l2)

    best_acc = 0
    history = {"train_loss": [], "train_acc": [], "test_loss": [], "test_acc": []}
```

Fig1. Preprocessing before training

```python
for epoch in range(args.epochs):

    model.train()
    batch_train_loss = []
    batch_train_acc = []

    for batch in tqdm(train_dataloader):

        features, labels = batch
        features = features.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        y_pred = model(features)

        loss = criterion(y_pred, labels.long())
        # back propagation
        loss.backward()
        # update weight
        optimizer.step()

        acc = (y_pred.argmax(dim= -1) == labels).float().mean()

        batch_train_acc.append(acc)
        batch_train_loss.append(loss.item())

    train_acc = sum(batch_train_acc) / len(batch_train_acc)
    train_loss = sum(batch_train_loss) / len(batch_train_loss)
    history["train_acc"].append(train_acc)
    history["train_loss"].append(train_loss)
```

Fig2. training code

Figure 3 illustrates the testing phase. The most crucial step is loading the weights of the pre-trained model and ensuring that the hyperparameter of the model are identical to those of the training model, followed by inferencing the test data. Notably, the testing phase does not involve a backward pass and weight updating in the testing phase, as it does not utilize gradients.

```python
def test(args):

    device = get_device()
    # get dataloader
    test_dataloader = get_dataloader(method= args.method, phase= "test", batch_size= args.batch_size)
    model = SCCNet(C= args.channel, Nu= args.Nu, Nc= args.Nc, Nt= args.Nt, dropoutRate= args.dropoutRate).to(device)
    # load pre-trained model
    if args.pre_trained != "":
        model.load_state_dict(torch.load(args.pre_trained))
    model.eval()

    with torch.no_grad():

        criterion = nn.CrossEntropyLoss()
        batch_test_acc = []
        batch_test_loss = []

        for batch in tqdm(test_dataloader):
            features, labels = batch
            features = features.to(device)
            labels = labels.to(device)
            y_pred = model(features)

            loss = criterion(y_pred, labels.long())
            acc = (y_pred.argmax(dim= -1) == labels).float().mean()

            batch_test_acc.append(acc)
            batch_test_loss.append(loss.item())
        test_acc = sum(batch_test_acc) / len(batch_test_acc)
        test_loss = sum(batch_test_loss) / len(batch_test_loss)

        print(f"[ Test ] loss = {test_loss:.5f}, acc = {test_acc:.5f}")
```

Fig3. testing code

B. Details of the SCCNet

The architecture of SCCNet consists of four blocks: the first convolution block, the second convolution block, the pooling block, and the softmax block. The implemented code is shown in figure 4 and the architecture of the model is shown in figure 5.

In the first convolution block, also known as spatial component analysis, the EEG data is decomposed from the channel domain to a component domain. In other words, the multi-channel EEG data are transferred into multiple EEG spatial components. I ensure that Nt is set to 1 in the experiment to achieve the objective outlined in the paper. After the first convolution operation, the 2D EEG data become 3D EEG data as shown in figure5. The second convolution, also known as the spatial-temporal filter, operates on the spatial component and temporal domain. Due to the constraints specified in the paper, which sets the kernel to (Nu, 12) and applies zero padding, I utilize padding of (0, 6) to maintain the original size of the time slot. Furthermore, the input size of the second convolutional layer is (22 - C + 1, Nu, 438), where Nu matches the kernel size (Nu,12) of the second convolutional layer. This means that the size of the second output dimension will reduce to 1 as shown in figure5. Following the two convolutional blocks, I apply an average pooling layer of size (1, 62) and stride of (1, 12) to perform smoothing in the temporal domain. Finally, I apply a dense layer. Since the loss function is cross entropy, adding a softmax layer after the dense layer is unnecessary.

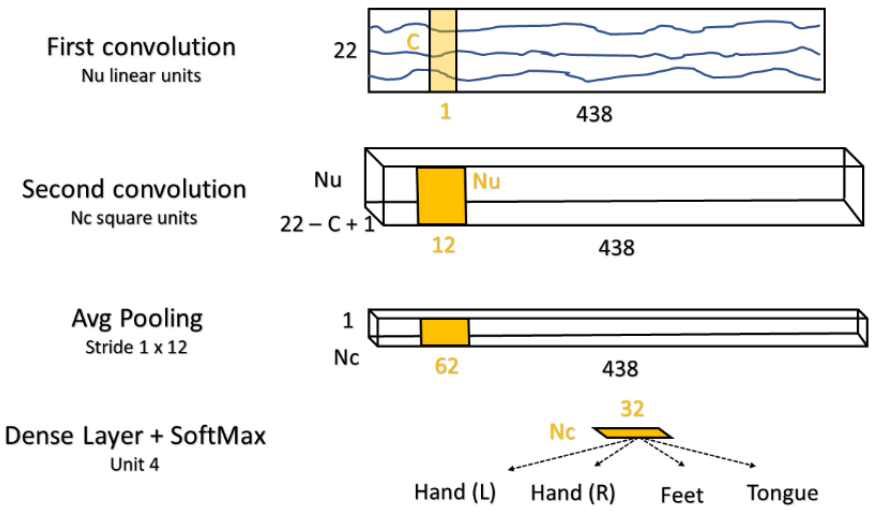


Fig4. The architecture of SCCNet

```python
def __init__(self, numClasses= 4, timeSample= 438, C= 22, Nu= 22, Nc= 20, Nt= 1, dropoutRate= 0.5):
    super(SCCNet, self).__init__()

    # spatial component analysis
    self.first_conv2d_block = nn.Sequential(
        nn.Conv2d(in_channels= 1, out_channels= Nu, kernel_size= (C, Nt)),
        nn.BatchNorm2d(num_features= Nu),
        nn.Dropout(p= dropoutRate),
    )

    self.second_conv2d_block = nn.Sequential(
        nn.Conv2d(in_channels= 22 - C + 1, out_channels= Nc, kernel_size= (Nu, 12), padding= (0, 6)),
        nn.BatchNorm2d(num_features= Nc),
        SquareLayer(),
        nn.Dropout(p= dropoutRate),
    )

    self.pooling_block = nn.Sequential(
        nn.AvgPool2d(kernel_size= (1, 62), stride= (1,12)),
    )

    self.softmax_block = nn.Sequential(
        nn.Linear(Nc * math.ceil((timeSample- 62 + 1) / 12), numClasses)
    )
```

Fig5. SCCNet code

3. Analyze on the experiment results

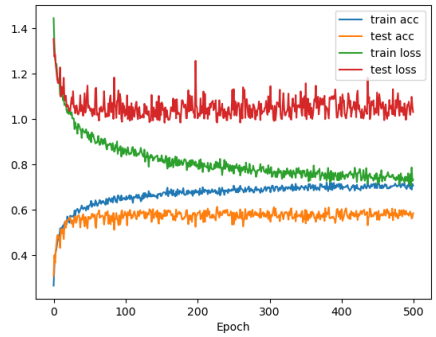
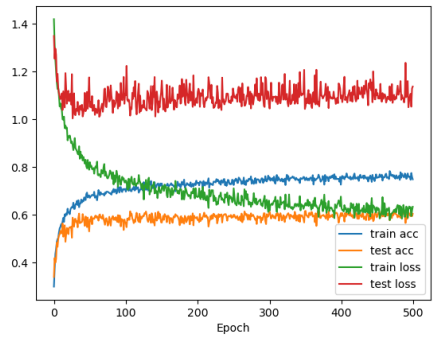
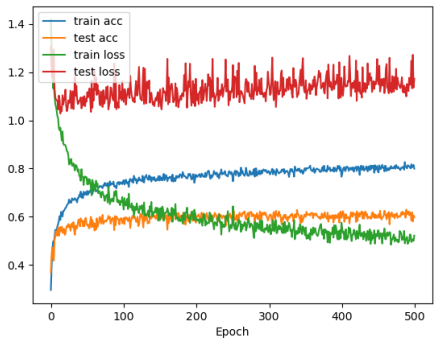
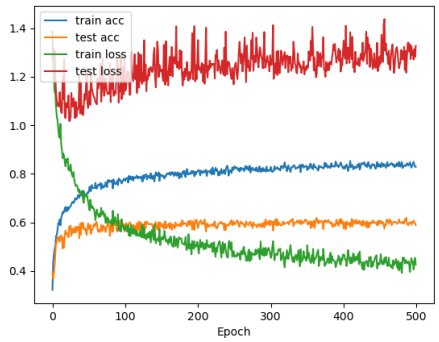
The final training results of three methods are as following. The accuracy of three method are beyond the experimental results in the paper. Unfortunately, it does not achieve the 100% standard specified in the Lab2.

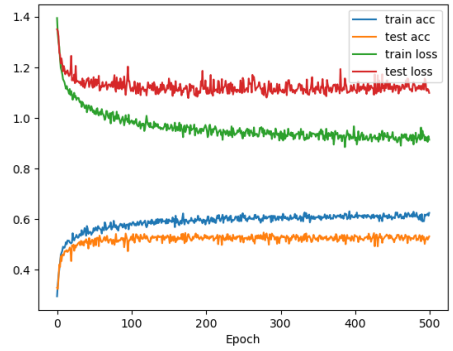
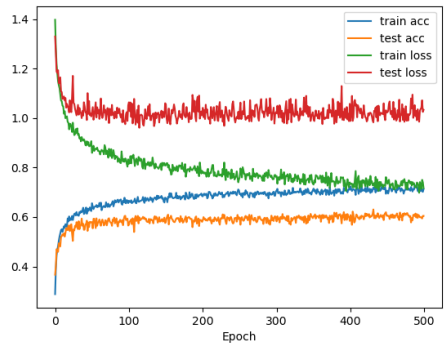
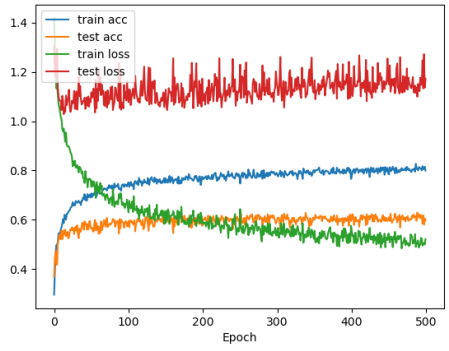
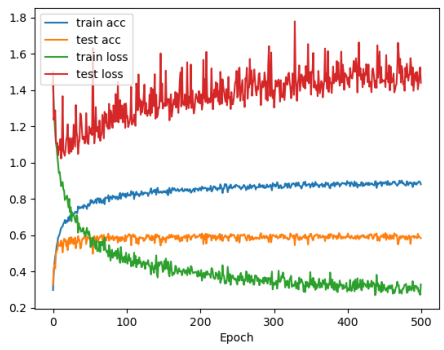
Method	SD	LOSO	Finetuned LOSO
Accuracy	0.62891	0.6250	0.7847

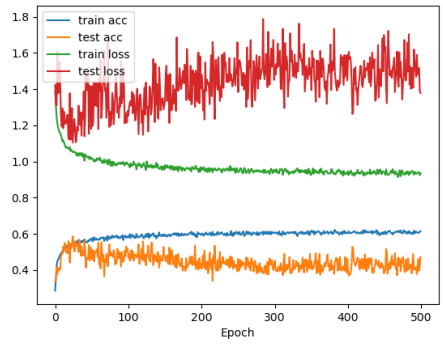
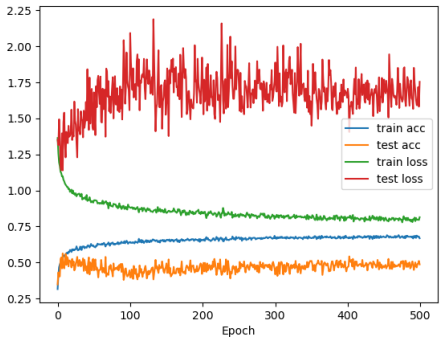
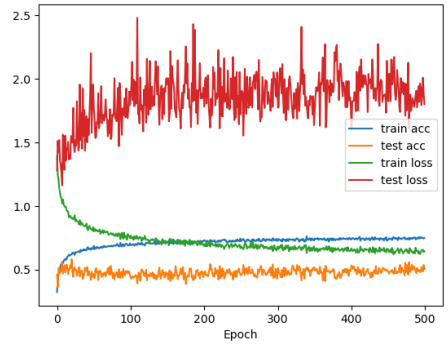
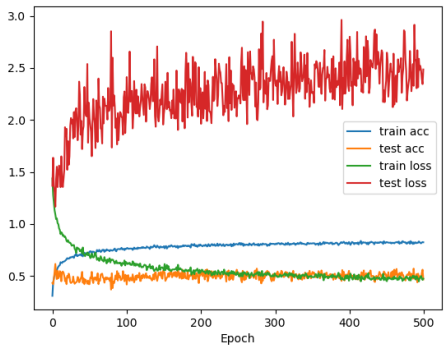
A. Discover during the training process

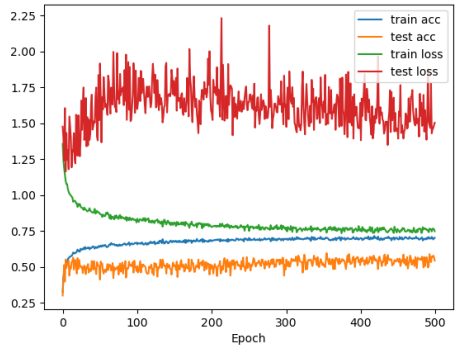
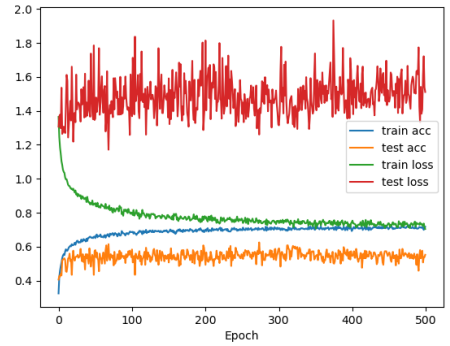
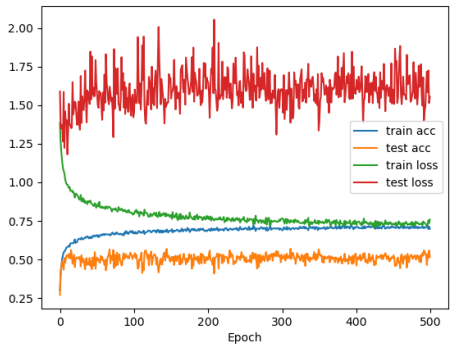
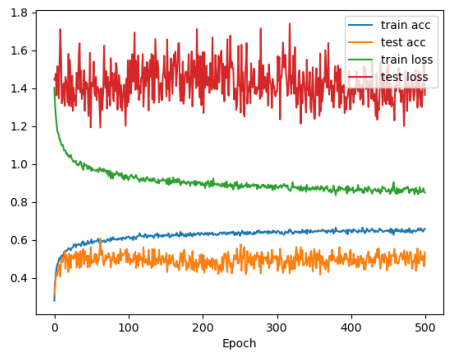
Given that the model architecture cannot be modified, the analysis is conducted by adjusting the model parameters. In the subsequent experiments, I fixed the number of epochs to 500, set the learning rate of the Adam optimizer to 0.001, the batch size to 32, and Nt to 1, and adjusted the parameters: C, Nc, and Nu.

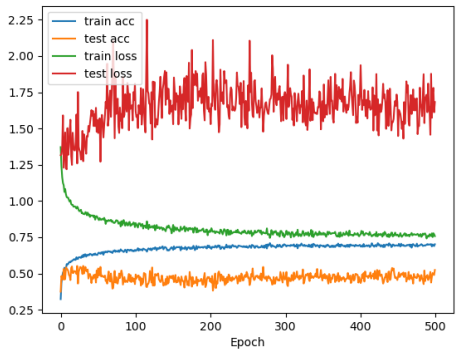
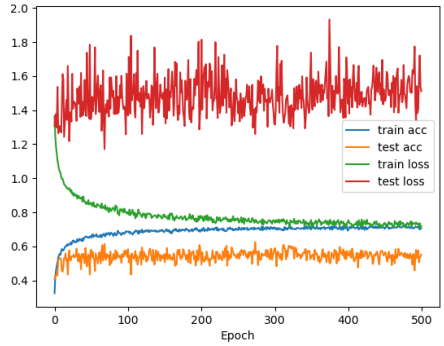
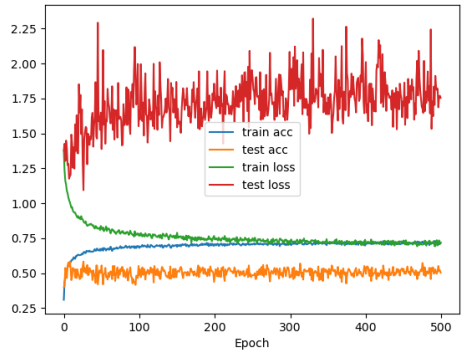
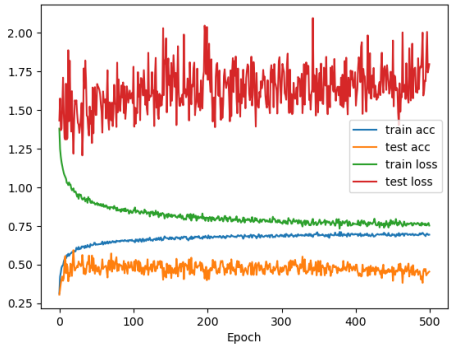
During the training process, the most significant issue encountered is overfitting. Regardless of how I reduce the parameters, overfitting occurs at a certain epoch, as evidenced by the red line (representing test loss) in the graph consistently being higher than the green line (representing train loss). Consequently, the improvement in testing accuracy is also limited. However, when the parameters C, Nc or Nu are increased, the training accuracy improves. In summary, I hypothesize that the training data is not strongly correlated with the testing data, resulting in a scenario where training continues improve, but testing does not change significantly.

Method = SD, C = 22, Nc = 20, Nt = 1	
	
Nu = 11, Accuracy = 0.6137	Nu = 22, Accuracy = 0.6280
	
Nu = 44, Accuracy = 0.6289	Nu = 88, Accuracy = 0.6189

Method = SD, C = 22, Nu = 44, Nt = 1	
	
Nc = 5, Accuracy = 0.5477	Nc= 10, Accuracy = 0.6304
	
Nc= 20, Accuracy = 0.6289	Nc= 40, Accuracy = 0.6198

Method = LOSO, C = 22, Nu = 44, Nt = 1	
	
Nc = 5, Accuracy = 0.5861	Nc= 10, Accuracy = 0.5477
	
Nc= 20, Accuracy = 0.5382	Nc= 40, Accuracy = 0.6155

Method = LOSO, Nc= 10, Nu = 22, Nt = 1	
	
C = 5, Accuracy = 0.5868	C= 11, Accuracy = 0.6250
	
C= 15, Accuracy = 0.5684	C= 22, Accuracy = 0.60764

Method = LOSO, C = 11, Nc = 10, Nt = 1	
	
Nu = 11, Accuracy = 0.5521	Nu = 22, Accuracy = 0.6250
	
Nu = 44, Accuracy = 0.5763	Nu = 88, Accuracy = 0.5938

B. Comparison between the three training methods

The paper presents three training methods: Subject-Dependent (SD), Leave-One-Subject-Out (LOSO), and Finetuned LOSO. The SD method trains the model using the first session of all 8 subjects as the training data and tests on the second session of the test subject. The LOSO method, similar to cross-validation, trains with the first and second sessions of 7 subjects and tests with the second session of the last subject. The finetuned LOSO method is based on LOSO pre-trained model, further trained with the first session of the last subject.

The main advantage of LOSO method is that it can train the model without data from a specific test subject. This allows the model to make predictions over a broader range and may be more generalizable. However, since it does not consider the individual differences of each subject, it may not fully utilize the specific information of each subject. Compared to LOSO, the SD method includes some data from the test subject in the training process, which may allow the model to better adapt and predict the behavior of that subject. However, a potential drawback of this method is that if the test subject's data greatly differs from the training data, the performance of the model may

decline. Finetuned LOSO method combines the advantages of LOSO and SD. Firstly, it trains an LOSO model using all the data from the other 8 subjects, then fine-tunes this model using the data from the test subject. This method may better utilize individual traits and may provide better performance in some cases.

Method	LOSO	SD	Finetuned LOSO
advantage	more generalizable	better adapt and predict the behavior of that subject	better utilize individual traits and may provide better performance
drawback	not consider the individual differences of each subject	testing data may greatly differ from the training data	Had the same problem with SD and LOSO

4. Discussion

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

As mentioned in the paper, EEG data have strong variability between subjects and even within a single subject. Therefore, increasing the size of training data by combining data from various subjects may not necessarily improve the performance of the model and could potentially be destructive, as evidenced by the experimental results. Although the accuracy improves during training, the testing result do not perform well and exhibit a high loss. Of course, it is also a problem that the model architecture does not have the capabilities and strategies are not good enough, resulting in the inability to classify EEG data. The following three figures represent the signals from subject1, subject2, and subject3, all of which are labels as “left hand”. These figures clearly demonstrate the strong variability between subjects.

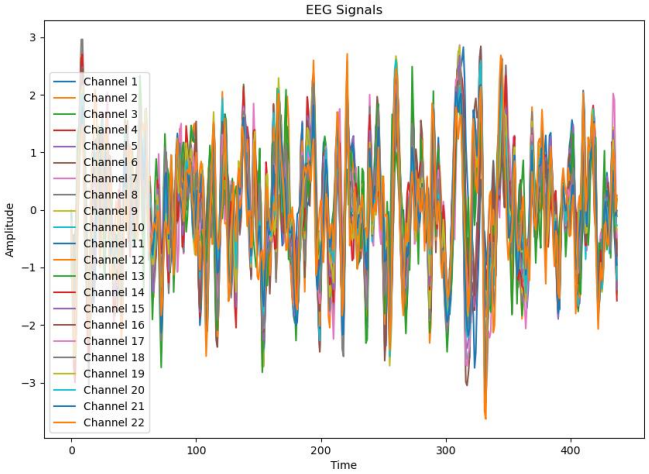


Fig6. Subject 1

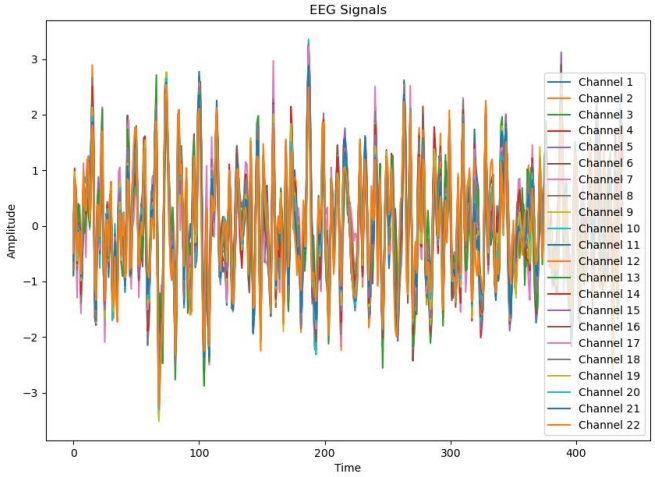


Fig7. Subject 2

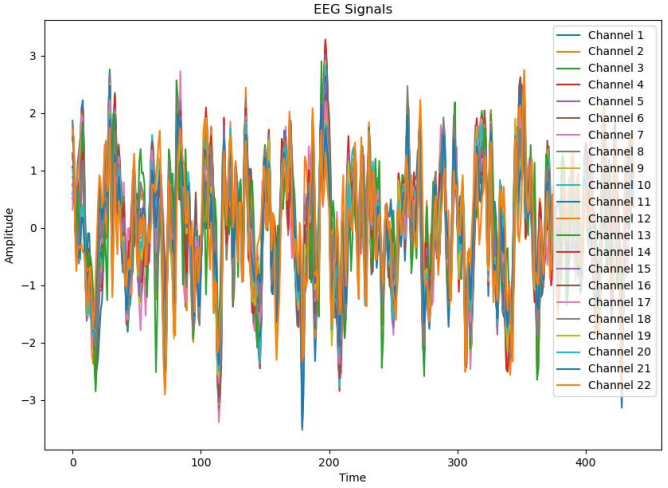


Fig8. Subject 3

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

The first method is akin to the subject-independent (SD) method described in the paper. Given the variability of EEG data across different subjects, training models on data from a single subject can yield better performance. The second method is a transfer-learning approach, similar to the fine-tuned LOSO method. We can train a primary model using data from multiple subjects. When we receive new subject data, we simply fine-tune the primary model. As we train with more subjects, the primary model become stronger. These two methods are based on solutions that do not modify the model architecture. However, if the model architecture can be modified, I believe we could add LSTM or RNN layers to address temporal issues. By considering temporal factors, the model can more comprehensively learn the complete features of the EEG data from different subjects to solve the problem of variability.