```
## Importing the necessary packages


import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, TensorDataset, random_split
from torchvision import transforms, utils
import time

# get the device type of machine
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# device = 'cpu'
print(device)
```

```
cuda
```

```
from google.colab import drive
drive.mount('/content/drive')
% cd 'drive/My Drive/ECE C147'
% cd 'project'
% ls
```

```
Mounted at /content/drive
/content/drive/My Drive/ECE C147
/content/drive/My Drive/ECE C147/project
EEG_loading.ipynb  person_train_valid.npy  X_train_valid.npy  y_train_valid.npy
person_test.npy    X_test.npy              y_test.npy
```

Loading the dataset and creating a windowed version

```
## Loading the numpy arrays

X_test = np.load("X_test.npy")
y_test = np.load("y_test.npy")
person_train_valid = np.load("person_train_valid.npy")
X_train_valid = np.load("X_train_valid.npy")
y_train_valid = np.load("y_train_valid.npy")
person_test = np.load("person_test.npy")

## Adjusting the labels so that

# Cue onset left - 0
# Cue onset right - 1
# Cue onset foot - 2
# Cue onset tongue - 3
```

```
y_train_valid -= 769
y_test -= 769


print ('Training/Valid data shape: {}'.format(X_train_valid.shape))
print ('Test data shape: {}'.format(X_test.shape))
print ('Training/Valid target shape: {}'.format(y_train_valid.shape))
print ('Test target shape: {}'.format(y_test.shape))
print ('Person train/valid shape: {}'.format(person_train_valid.shape))
print ('Person test shape: {}'.format(person_test.shape))
```

```
Training/Valid data shape: (2115, 22, 1000)
Test data shape: (443, 22, 1000)
Training/Valid target shape: (2115,)
Test target shape: (443,)
Person train/valid shape: (2115, 1)
Person test shape: (443, 1)
```

```
def make_steps(samples,samples_per_frame,stride):
    '''
    in:
    samples - number of samples in the session
    samples_per_frame - number of samples in the frame
    stride - the gap between succesive frames
    out: list of tuple ranges
    '''

    i = 0
    intervals = []
    while i+samples_per_frame <= samples:
        intervals.append((i,i+samples_per_frame))
        i = i + stride
    return intervals

def make_win_data_pipeline(data_arr,label_arr,num_samples_frame,stride):
    '''
    in:
    data_arr - original data array without windowing
    label_arr - labels of the data array without windowing
    num_samples_frame - number of samples in the frame
    stride - the gap between succesive frames

    out:
    data_win_arr - windowed data array
    label_win_arr - labels of the windowed data array

    '''
```

```
    num_trials = data_arr.shape[0]
    num_channels = data_arr.shape[1]
    num_samples = data_arr.shape[2]

    steps_list = make_steps(num_samples,num_samples_frame,stride)
    num_windows = len(steps_list)

    data_win_arr = np.zeros((num_trials*num_windows,num_channels,num_samples_frame))
    label_win_arr = []
    k = 0

    for i in range(num_trials):

        trial_label = label_arr[i]
        trial_data = data_arr[i,:,:]

        for m,n in enumerate(steps_list):
            start_ind = n[0]
            end_ind = n[1]

            win_data = trial_data[:,start_ind:end_ind]
            data_win_arr[k,:,:] = win_data
            label_win_arr.append(trial_label)
            k = k+1

    label_win_arr = np.asarray(label_win_arr)
    return data_win_arr, label_win_arr
```

Creating the custom dataset for torch

```
# Creating the custom dataset

class EEGDataset(Dataset):

    """EEG dataset"""
    def __init__(self, subset, transform=None):

        'Initialization'

        self.subset = subset
        self.transform = transform

    def __getitem__(self, index):

        'Generates one sample of data'

        x, y = self.subset[index]
        if self.transform:
```

```
          pass
            # x = self.transform(x)
            # y = self.transform(y)
        return x, y

    def __len__(self):

        'Denotes the total number of samples'
        return len(self.subset)
```

Defining the models

```
# Defining the shallow conv net


class ShallowConv(nn.Module):

    # Defining the building blocks of shallow conv net

    def __init__(self, in_channels, num_conv_filters, num_samples_frame, num_eeg_channels,cla

        # Defining as a subclass
        super(ShallowConv, self).__init__()

        self.num_samples_frame = num_samples_frame
        self.num_conv_filters = num_conv_filters
        self.num_eeg_channels = num_eeg_channels

        # Define the convolution layer, https://pytorch.org/docs/stable/generated/torch.nn.Co
        self.conv1 = nn.Conv2d(in_channels, self.num_conv_filters, (1, 25), stride=1)
        self.conv_output_width =  int(self.num_samples_frame - (25-1) - 1 + 1)

        # Define the 2d batchnorm layer
        self.bnorm2d = nn.BatchNorm2d(self.num_conv_filters)

        # Define the 1d batchnorm layer
        self.bnorm1d = nn.BatchNorm1d(self.num_conv_filters)


        # Define the fc layer, https://pytorch.org/docs/stable/generated/torch.nn.Linear.html
        self.fc1 = nn.Linear(self.num_eeg_channels*self.num_conv_filters, self.num_conv_filte

        # Define the elu activation
        self.elu = nn.ELU(0.2)

        # Define the avg pooling layer
        self.avgpool = nn.AvgPool1d(75, stride=15)
```

```
        self.num_features_linear = int(np.floor(((self.conv_output_width - 75)/15)+1))



        # Define the fc layer for generating the scores for classes
        self.fc2 = nn.Linear(self.num_features_linear*self.num_conv_filters, classes)



      # Defining the connections of shallow conv net

    def forward(self, x):

        # Reshaping the input for 2-D convolution (B,22,num_samples_frame) -> (B,1,22,num_sam

        x = x.view(-1, 1, 22, self.num_samples_frame)

        # Performing the 2-D convolution (B,1,22,300) -> (B,40,22,x_shape_4dim)

        x = self.conv1(x)
        x_shape_4dim = x.shape[3]

        # ELU activation

        x = self.elu(x)

        # 2d Batch normalization

        x = self.bnorm2d(x)


        # Reshaping the input to dense layer (B,40,22,x_shape_4dim) -> (B,x_shape_4dim,880)

        x = x.permute(0,3,1,2) # (B,40,22,x_shape_4dim) -> (B,x_shape_4dim,40,22)
        x = x.view(-1,x_shape_4dim,880)

        # Passing through the dense layer (B,x_shape_4dim,880) -> (B,x_shape_4dim,40)

        x = self.fc1(x)

        # ELU activation

        x = self.elu(x)

        # Square activation

        x = torch.square(x)

        # Reshaping the input for average pooling layer (B,x_shape_4dim,40) -> (B,40,x_shape_

        x = x.permute(0,2,1)
```

```
        x = x.permute(0,2,1)

        # Passing through the average pooling layer (B,40,x_shape_4dim) -> (B,40,x_pool_3dim)

        x = self.avgpool(x)
        x_pool_3dim = x.shape[2]

        # Log activation

        x = torch.log(x)

        # 1D Batch normalization

        x = self.bnorm1d(x)
        #print(x.shape)

        # Reshaping the input to dense layer (B,40,x_pool_3dim) -> (B,40*x_pool_3dim)

        x = x.reshape(-1, 40*x_pool_3dim)

        # Passing through the dense layer (B,40*x_pool_3dim) -> (B,classes)

        x = self.fc2(x)

        # Passing through the softmax layer



        return x
```

Defining the training and validation of the model

```
## Defining the training and validation function

def train_val(model,optimizer,criterion,num_epochs):



    train_loss = []
    train_acc = []
    val_loss = []
    val_acc = []

    for epoch in range(num_epochs):

        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)
```

```
        for phase in ['train','val']:



            #Initializing the losses and accuracy

            training_loss = 0
            correct_train_preds = 0
            total_train_preds = 0
            batch_train_idx = 0

            validation_loss = 0
            correct_val_preds = 0
            total_val_preds = 0
            batch_val_idx = 0


            # Implementing the training phase

            if phase == 'train':

                # setting the model to training mode

                model.train()

                # Loading the training dataset in batches

                for inputs, labels in dataloaders['train']:

                    # Transfer input data and labels to device

                    inputs = inputs.to(device)
                    labels = labels.to(device)

                    # Incrementing the batch counter

                    batch_train_idx += 1

                    # Zeroing the gradient buffer

                    optimizer.zero_grad()

                    # Perform the forward pass

                    outputs = model(inputs)

                    # Compute loss

                    loss = criterion(outputs,labels)
```

```
            # Perform the backward pass

            loss.backward()

            # Perform optimization step

            optimizer.step()

            # Compute training statistics

            training_loss += loss.item()
            _, predicted = outputs.max(1)
            total_train_preds += labels.size(0)
            correct_train_preds += predicted.eq(labels).sum().item()


        train_loss.append(training_loss)
        t_acc = correct_train_preds/total_train_preds
        train_acc.append(t_acc)
        print('Training loss:',training_loss)
        print('Training accuracy:',t_acc)


    else:



        # setting the model to evaluation mode

        model.eval()

        # Disable gradient computation

        with torch.no_grad():

            # Loading the training dataset in batches

            for val_inputs, val_labels in dataloaders['val']:


                # Transfer input data and labels to device

                val_inputs = val_inputs.to(device)
                val_labels = val_labels.to(device)

                # Incrementing the batch counter

                batch_val_idx += 1

                # Perform forward pass
```

```
                    val_outputs = model(val_inputs)

                    # Compute loss

                    valid_loss = criterion(val_outputs,val_labels)


                    # Compute validation statistics

                    validation_loss += valid_loss.item()
                    _, val_predicted = val_outputs.max(1)
                    total_val_preds += val_labels.size(0)
                    correct_val_preds += val_predicted.eq(val_labels).sum().item()

                val_loss.append(validation_loss)
                v_acc = correct_val_preds/total_val_preds
                val_acc.append(v_acc)
                print('Validation loss:',validation_loss)
                print('Validation accuracy:',v_acc)




    return model, train_loss, train_acc, val_loss, val_acc
```

Training,validating and testing the model with 1000 samples per trial

```
## Preparing the training and validation data

num_samples_frame = 1000
stride = 50
X_train_win,y_train_win = make_win_data_pipeline(X_train_valid,y_train_valid,num_samples_fram

print ('Windowed Training/Valid data shape: {}'.format(X_train_win.shape))
print ('Windowed Training/Valid label shape: {}'.format(y_train_win.shape))

# Converting the numpy data to torch tensors

X_train_valid_tensor = torch.from_numpy(X_train_win).float().to(device)
y_train_valid_tensor = torch.from_numpy(y_train_win).float().long().to(device)

print ('Training/Valid tensor shape: {}'.format(X_train_valid_tensor.shape))
print ('Training/Valid target tensor shape: {}'.format(y_train_valid_tensor.shape))

init_dataset = TensorDataset(X_train_valid_tensor, y_train_valid_tensor)

# Spliting the dataset into training and validation
```

```
# Spliting the dataset into training and validation

lengths = [int(len(init_dataset)*0.8), int(len(init_dataset)*0.2)]
subset_train, subset_val = random_split(init_dataset, lengths)

train_data = EEGDataset(subset_train, transform=None)
val_data = EEGDataset(subset_val, transform=None)

# Constructing the training and validation dataloaders

dataloaders = {
    'train': torch.utils.data.DataLoader(train_data, batch_size=32, shuffle=True, num_workers
    'val': torch.utils.data.DataLoader(val_data, batch_size=8, shuffle=False, num_workers=0)
}
```

```
Windowed Training/Valid data shape: (2115, 22, 1000)
Windowed Training/Valid label shape: (2115,)
Training/Valid tensor shape: torch.Size([2115, 22, 1000])
Training/Valid target tensor shape: torch.Size([2115])
```

```
# Defining the parameters for model training


weight_decay = 0.15  # weight decay to alleviate overfiting
shallow_model = ShallowConv(in_channels=1, num_conv_filters=40,num_samples_frame=1000,num_eeg
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(shallow_model.parameters(), lr = 1e-4, weight_decay=weight_decay

# Training and validating the model
num_epoch=200
shallow_model,t_l,t_a,v_l,v_a=train_val(shallow_model, optimizer, criterion, num_epochs=num_e
```

```
----------
Training loss: 13.62040539085865
Training accuracy: 0.9905437352245863
Validation loss: 46.24142950773239
Validation accuracy: 0.640661938534279
Epoch 191/199
----------
Training loss: 11.871786415576935
Training accuracy: 0.99822695035461
Validation loss: 46.138053804636
Validation accuracy: 0.640661938534279
Epoch 192/199
----------
Training loss: 11.304404214024544
Training accuracy: 0.9994089834515366

Validation loss: 45.59165704250336
Validation accuracy: 0.6524822695035462
```

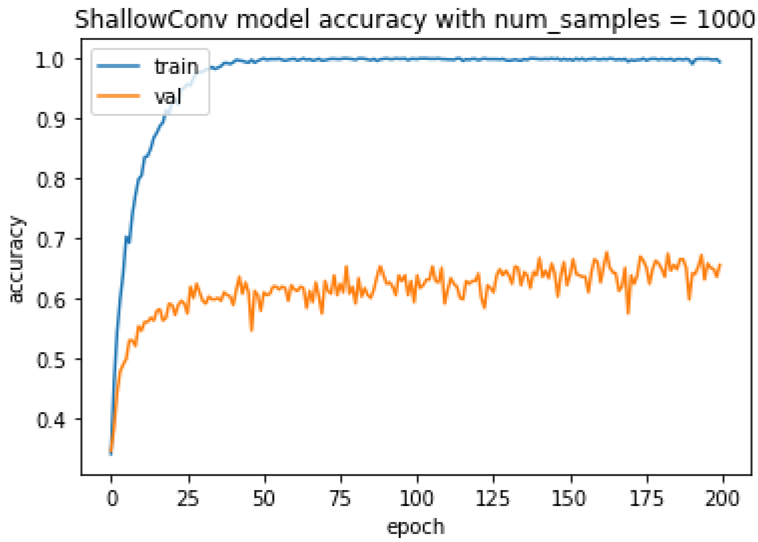
```
Epoch 193/199
----------
Training loss: 11.66141477227211
Training accuracy: 0.9994089834515366
Validation loss: 45.31048172712326
Validation accuracy: 0.6713947990543735
Epoch 194/199
----------
Training loss: 11.064806789159775
Training accuracy: 0.9988179669030733
Validation loss: 46.9757679104805
Validation accuracy: 0.6312056737588653
Epoch 195/199
----------
Training loss: 11.57563641667366
Training accuracy: 0.9994089834515366
Validation loss: 45.78339001536369
Validation accuracy: 0.6572104018912529
Epoch 196/199
----------
Training loss: 11.33141578733921
Training accuracy: 0.99822695035461
Validation loss: 46.35113310813904
Validation accuracy: 0.6501182033096927
Epoch 197/199
----------
Training loss: 11.987960189580917
Training accuracy: 0.9976359338061466
Validation loss: 45.7522796690464
Validation accuracy: 0.6477541371158393
Epoch 198/199
----------
Training loss: 11.420435383915901
Training accuracy: 0.9988179669030733
Validation loss: 45.86351674795151
Validation accuracy: 0.6359338061465721
Epoch 199/199
----------
Training loss: 11.868240520358086
Training accuracy: 0.9940898345153665
Validation loss: 45.230103462934494
Validation accuracy: 0.6548463356973995
```

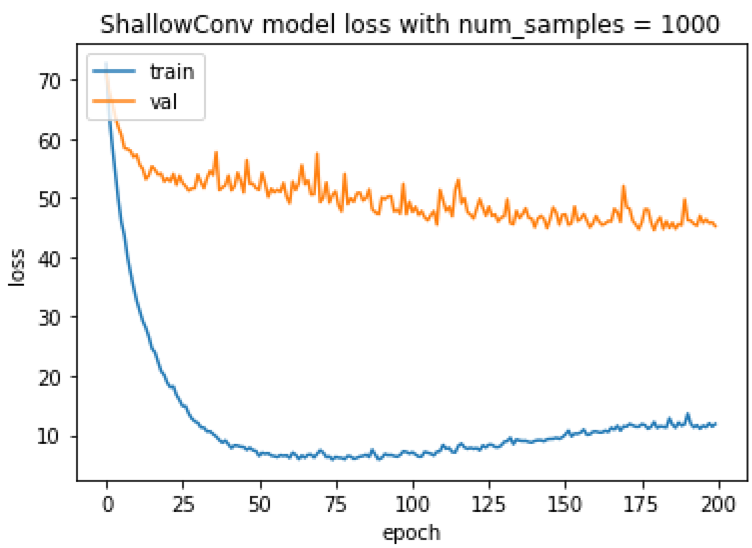
```
# Plotting the training and validation history
```

```
import matplotlib.pyplot as plt

print(v_a)
plt.plot(range(num_epoch),t_a)
plt.plot(range(num_epoch),v_a)
plt.title('ShallowConv model accuracy with num_samples = 1000')
plt.ylabel('accuracy')
plt.xlabel('epoch')
```

```
plt.legend(['train','val'], loc='upper left')
plt.show()



plt.plot(range(num_epoch),t_l)
plt.plot(range(num_epoch),v_l)
plt.title('ShallowConv model loss with num_samples = 1000')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','val'], loc='upper left')
plt.show()
```

```
[0.34515366430260047, 0.3829787234042553, 0.44208037825059104, 0.47754137115839246, 0.48
```






```
# Testing the model

num_samples_frame = 1000
stride = 50
```

```
X_test_win,y_test_win = make_win_data_pipeline(X_test,y_test,num_samples_frame,stride)

# Preparing the test dataset
X_test_tensor = torch.from_numpy(X_test_win).float().to(device)
y_test_tensor = torch.from_numpy(y_test_win).float().long().to(device)

test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

test_data = EEGDataset(test_dataset, transform=None)



def test_model(model,test_data,criterion):


    # Creating the test dataloader
    test_dataloader = torch.utils.data.DataLoader(test_data, batch_size=8, shuffle=True, num_

    # Making the predictions on the dataset

    total_test_preds = 0
    correct_test_preds = 0
    test_loss = 0

    model.eval()
    with torch.no_grad():

        for test_inputs, test_labels in test_dataloader:

            # Transfer test data and labels to device
            test_inputs = test_inputs.to(device)
            test_labels = test_labels.to(device)

            # Perform forward pass
            test_outputs = model(test_inputs)

            # Compute loss
            test_loss = criterion(test_outputs,test_labels)

            # Compute test statistics

            test_loss += test_loss.item()
            _, test_predicted = test_outputs.max(1)
            total_test_preds += test_labels.size(0)
            correct_test_preds += test_predicted.eq(test_labels).sum().item()

        test_acc = correct_test_preds/total_test_preds
        print('Test loss', test_loss)
        print('Test accuracy',test_acc*100)


    return test_acc
```

```
    return test_acc

test_a = test_model(shallow_model,test_data,criterion)
```

```
Test loss tensor(1.1953, device='cuda:0')
Test accuracy 65.23702031602708
```

Training,validating and testing the model with 500 samples per trial

```
## Preparing the training and validation data

num_samples_frame = 500
stride = 100
X_train_win,y_train_win = make_win_data_pipeline(X_train_valid,y_train_valid,num_samples_fram

print ('Windowed Training/Valid data shape: {}'.format(X_train_win.shape))
print ('Windowed Training/Valid label shape: {}'.format(y_train_win.shape))

# Converting the numpy data to torch tensors

X_train_valid_tensor = torch.from_numpy(X_train_win).float().to(device)
y_train_valid_tensor = torch.from_numpy(y_train_win).float().long().to(device)

print ('Training/Valid tensor shape: {}'.format(X_train_valid_tensor.shape))
print ('Training/Valid target tensor shape: {}'.format(y_train_valid_tensor.shape))

init_dataset = TensorDataset(X_train_valid_tensor, y_train_valid_tensor)

# Spliting the dataset into training and validation

lengths = [int(len(init_dataset)*0.8), int(len(init_dataset)*0.2)]
subset_train, subset_val = random_split(init_dataset, lengths)

train_data = EEGDataset(subset_train, transform=None)
val_data = EEGDataset(subset_val, transform=None)

# Constructing the training and validation dataloaders

dataloaders = {
    'train': torch.utils.data.DataLoader(train_data, batch_size=32, shuffle=True, num_workers
    'val': torch.utils.data.DataLoader(val_data, batch_size=8, shuffle=False, num_workers=0)
}
```

```
Windowed Training/Valid data shape: (12690, 22, 500)
Windowed Training/Valid label shape: (12690,)
Training/Valid tensor shape: torch.Size([12690, 22, 500])
Training/Valid target tensor shape: torch.Size([12690])
```

```
# Defining the parameters for model training


weight_decay = 0.15  # weight decay to alleviate overfiting
shallow_model = ShallowConv(in_channels=1, num_conv_filters=40,num_samples_frame=500,num_eeg_
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(shallow_model.parameters(), lr = 1e-5, weight_decay=weight_decay

# Training and validating the model
epoch=600
shallow_model,t_l,t_a,v_l,v_a=train_val(shallow_model, optimizer, criterion, num_epochs=epoch
```

```
----------
Training loss: 117.18125839531422
Training accuracy: 0.969070133963751
Validation loss: 167.66588066518307
Validation accuracy: 0.8782505910165485
Epoch 591/599
----------
Training loss: 116.57842734456062
Training accuracy: 0.9710401891252955
Validation loss: 158.88496893644333
Validation accuracy: 0.9026792750197006
Epoch 592/599
----------
Training loss: 116.43788632750511
Training accuracy: 0.9713356973995272
Validation loss: 160.04617117345333
Validation accuracy: 0.9003152088258471
Epoch 593/599
----------
Training loss: 114.91826002299786
Training accuracy: 0.9716312056737588
Validation loss: 162.16221196949482
Validation accuracy: 0.8888888888888888
Epoch 594/599
----------
Training loss: 115.09499441087246
Training accuracy: 0.973404255319149
Validation loss: 160.72544640302658
Validation accuracy: 0.8987391646966115
Epoch 595/599
----------
Training loss: 114.1037253588438
Training accuracy: 0.9741922773837668
Validation loss: 157.73168356716633
Validation accuracy: 0.9109535066981875
Epoch 596/599

----------
Training loss: 115.34859521687031
Training accuracy: 0.9706461780929866
Validation loss: 159.916859716177
Validation accuracy: 0.9034672970843184
Epoch 597/599
----------
Training loss: 116.22640573978424
```

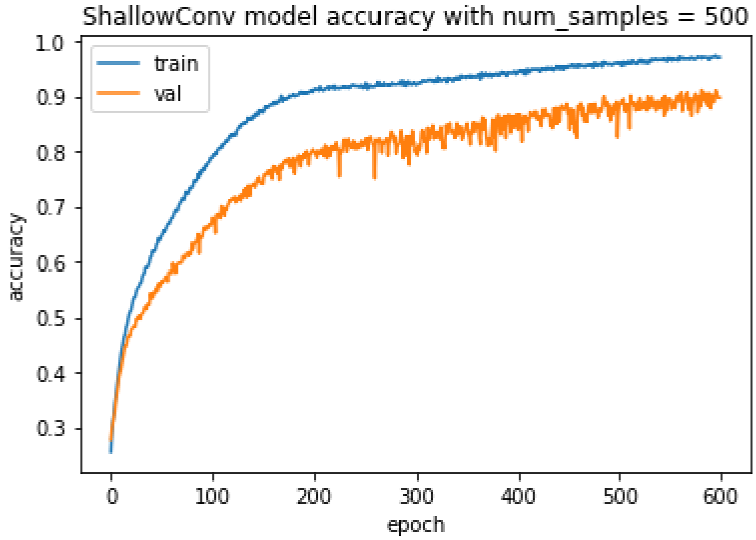
```
Training accuracy: 0.969956658786446
Validation loss: 159.92002944648266
Validation accuracy: 0.8975571315996848
Epoch 598/599
----------
Training loss: 115.20905143022537
Training accuracy: 0.9705476753349094
Validation loss: 159.32293625175953
Validation accuracy: 0.8975571315996848
Epoch 599/599
----------
Training loss: 115.96913474798203
Training accuracy: 0.9711386918833728
Validation loss: 159.1123557537794
Validation accuracy: 0.8975571315996848
```

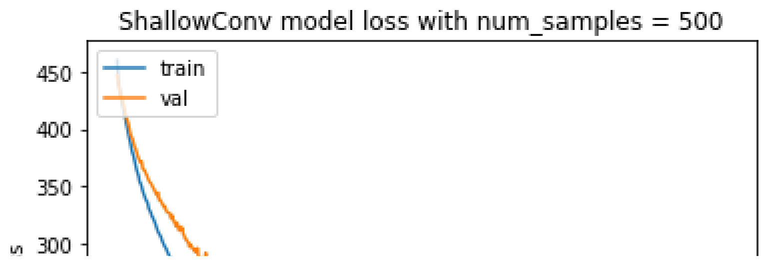
```
# Plotting the training and validation history
```

```
import matplotlib.pyplot as plt

print(v_a)
plt.plot(range(epoch),t_a)
plt.plot(range(epoch),v_a)
plt.title('ShallowConv model accuracy with num_samples = 500')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','val'], loc='upper left')
plt.show()
```

```
plt.plot(range(epoch),t_l)
plt.plot(range(epoch),v_l)
plt.title('ShallowConv model loss with num_samples = 500')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','val'], loc='upper left')
plt.show()
```

```
[0.2777777777777778, 0.2970843183609141, 0.3073286052009456, 0.3195429472025217, 0.3297{
```






```
# Testing the model

num_samples_frame = 500
stride = 100
X_test_win,y_test_win = make_win_data_pipeline(X_test,y_test,num_samples_frame,stride)

# Preparing the test dataset
X_test_tensor = torch.from_numpy(X_test_win).float().to(device)
y_test_tensor = torch.from_numpy(y_test_win).float().long().to(device)

test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

test_data = EEGDataset(test_dataset, transform=None)

test_a = test_model(shallow_model,test_data,criterion)
```

```
Test loss tensor(1.7838, device='cuda:0')
Test accuracy 48.23175319789315
```

Training,validating and testing the model with 300 samples per trial

```
## Preparing the training and validation data

num_samples_frame = 300
stride = 60
X_train_win,y_train_win = make_win_data_pipeline(X_train_valid,y_train_valid,num_samples_fram
```

```
print ('Windowed Training/Valid data shape: {}'.format(X_train_win.shape))
print ('Windowed Training/Valid label shape: {}'.format(y_train_win.shape))

# Converting the numpy data to torch tensors

X_train_valid_tensor = torch.from_numpy(X_train_win).float().to(device)
y_train_valid_tensor = torch.from_numpy(y_train_win).float().long().to(device)

print ('Training/Valid tensor shape: {}'.format(X_train_valid_tensor.shape))
print ('Training/Valid target tensor shape: {}'.format(y_train_valid_tensor.shape))

init_dataset = TensorDataset(X_train_valid_tensor, y_train_valid_tensor)

# Spliting the dataset into training and validation

lengths = [int(len(init_dataset)*0.8), int(len(init_dataset)*0.2)]
subset_train, subset_val = random_split(init_dataset, lengths)

train_data = EEGDataset(subset_train, transform=None)
val_data = EEGDataset(subset_val, transform=None)

# Constructing the training and validation dataloaders

dataloaders = {
    'train': torch.utils.data.DataLoader(train_data, batch_size=32, shuffle=True, num_workers
    'val': torch.utils.data.DataLoader(val_data, batch_size=8, shuffle=False, num_workers=0)
}
```

```
Windowed Training/Valid data shape: (25380, 22, 300)
Windowed Training/Valid label shape: (25380,)
Training/Valid tensor shape: torch.Size([25380, 22, 300])
Training/Valid target tensor shape: torch.Size([25380])
```

```
# Defining the parameters for model training


weight_decay = 0.15  # weight decay to alleviate overfiting
shallow_model = ShallowConv(in_channels=1, num_conv_filters=40,num_samples_frame=300,num_eeg_
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(shallow_model.parameters(), lr = 1e-5, weight_decay=weight_decay
epoch=500
# Training and validating the model

shallow_model,t_l,t_a,v_l,v_a=train_val(shallow_model, optimizer, criterion, num_epochs=epoch
```

```
----------
Training loss: 335.36366787552834
Training accuracy: 0.9218873128447597
Validation loss: 432.33226826786995
Validation accuracy: 0.814026792750197
```

```
Validation accuracy: 0.814026792750197
Epoch 491/499
----------
Training loss: 335.4767629802227
Training accuracy: 0.9238081166272656
Validation loss: 431.3445480763912
Validation accuracy: 0.8130417651694247
Epoch 492/499
----------
Training loss: 336.6296007633209
Training accuracy: 0.9213455476753349
Validation loss: 418.34600818157196
Validation accuracy: 0.8297872340425532
Epoch 493/499
----------
Training loss: 334.1713341474533
Training accuracy: 0.924645390070922
Validation loss: 448.4650750756264
Validation accuracy: 0.7799448384554768
Epoch 494/499
----------
Training loss: 334.402989000082

Training accuracy: 0.9253349093774625
Validation loss: 416.81055849790573
Validation accuracy: 0.8368794326241135
Epoch 495/499
----------
Training loss: 334.5504116117954
Training accuracy: 0.9237096138691884
Validation loss: 411.161266207695
Validation accuracy: 0.8378644602048857
Epoch 496/499
----------
Training loss: 333.81252458691597
Training accuracy: 0.9253841607565012
Validation loss: 422.41213831305504
Validation accuracy: 0.8264381402679275
Epoch 497/499
----------
Training loss: 333.2014458477497
Training accuracy: 0.9234141055949566
Validation loss: 422.80629459023476
Validation accuracy: 0.8157998423955871
Epoch 498/499
----------
Training loss: 334.38831117749214
Training accuracy: 0.9238573680063041
Validation loss: 417.11436754465103
Validation accuracy: 0.8402285263987391
Epoch 499/499
----------
Training loss: 334.5765996873379
Training accuracy: 0.925531914893617
Validation loss: 447.5720668435097
Validation accuracy: 0.7693065405831363
```

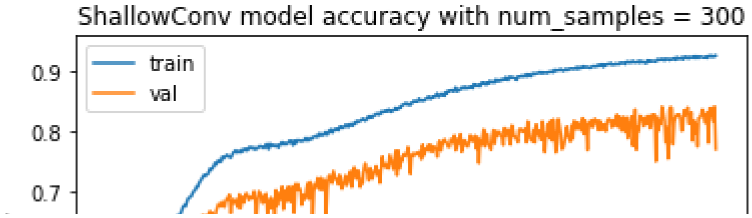
```
# Plotting the training and validation history
```

```
# Plotting the training and validation history
```

```
import matplotlib.pyplot as plt

print(v_a)
plt.plot(range(epoch),t_a)
plt.plot(range(epoch),v_a)
plt.title('ShallowConv model accuracy with num_samples = 300')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','val'], loc='upper left')
plt.show()
```

```
plt.plot(range(epoch),t_l)
plt.plot(range(epoch),v_l)
plt.title('ShallowConv model loss with num_samples = 300')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','val'], loc='upper left')
plt.show()
```

```
[0.27698975571316, 0.2929472025216706, 0.30437352245862886, 0.3195429472025217, 0.32959
```




```
# Testing the model

num_samples_frame = 300
stride = 60
X_test_win,y_test_win = make_win_data_pipeline(X_test,y_test,num_samples_frame,stride)

# Preparing the test dataset
X_test_tensor = torch.from_numpy(X_test_win).float().to(device)
y_test_tensor = torch.from_numpy(y_test_win).float().long().to(device)

test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

test_data = EEGDataset(test_dataset, transform=None)

test_a = test_model(shallow_model,test_data,criterion)
```

```
Test loss tensor(2.7331, device='cuda:0')
Test accuracy 40.105342362678705
```