In this discussion, we will build a basic hybrid CNN-LSTM model for classification on the EEG dataset

This notebook was inspired to Tonmoy with some attempts to tune by us

(i) Importing the necessary packages

In [ ]:
```
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential,Model
from keras.layers import Dense, Activation, Flatten,Dropout
from keras.layers import Conv2D,LSTM,BatchNormalization,MaxPooling2D,Reshape
from keras.utils import to_categorical
import matplotlib.pyplot as plt
```

```
2023-03-18 20:47:02.872775: I tensorflow/core/platform/cpu_feature_guard.c
c:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN) to use the following CPU instructions in performance-criti
cal operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

(ii) Preprocessing the dataset and preparing the training, validation, and test datasets

In [ ]:
```
def data_prep(X,y,sub_sample,average,noise):

    total_X = None
    total_y = None

    # Trimming the data (sample,22,1000) -> (sample,22,500)
    X = X[:,:,0:500]
    print('Shape of X after trimming:',X.shape)

    # Maxpooling the data (sample,22,1000) -> (sample,22,500/sub_sample)
    X_max = np.max(X.reshape(X.shape[0], X.shape[1], -1, sub_sample), axis=3


    total_X = X_max
    total_y = y
    print('Shape of X after maxpooling:',total_X.shape)

    # Averaging + noise
    X_average = np.mean(X.reshape(X.shape[0], X.shape[1], -1, average),axis=
    X_average = X_average + np.random.normal(0.0, 0.5, X_average.shape)

    total_X = np.vstack((total_X, X_average))
    total_y = np.hstack((total_y, y))
    print('Shape of X after averaging+noise and concatenating:',total_X.shap

    # Subsampling

    for i in range(sub_sample):

        X_subsample = X[:, :, i::sub_sample] + \
                            (np.random.normal(0.0, 0.5, X[:, :,i::sub_sample

        total_X = np.vstack((total_X, X_subsample))
        total_y = np.hstack((total_y, y))


    print('Shape of X after subsampling and concatenating:',total_X.shape)
    return total_X,total_y
```

In [ ]:
```
## Loading the dataset


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



## Random splitting and reshaping the data
# First generating the training and validation indices using random splittin

ind_valid = np.random.choice(2115, 375, replace=False)
ind_train = np.array(list(set(range(2115)).difference(set(ind_valid))))

# Creating the training and validation sets using the generated indices
(X_train, X_valid) = X_train_valid[ind_train], X_train_valid[ind_valid]
(y_train, y_valid) = y_train_valid[ind_train], y_train_valid[ind_valid]


## Preprocessing the dataset
x_train,y_train = data_prep(X_train,y_train,2,2,True)
x_valid,y_valid = data_prep(X_valid,y_valid,2,2,True)
X_test_prep,y_test_prep = data_prep(X_test,y_test,2,2,True)


print('Shape of training set:',x_train.shape)
print('Shape of validation set:',x_valid.shape)
print('Shape of training labels:',y_train.shape)
print('Shape of validation labels:',y_valid.shape)
print('Shape of testing set:',X_test_prep.shape)
print('Shape of testing labels:',y_test_prep.shape)


# Converting the labels to categorical variables for multiclass classificati
y_train = to_categorical(y_train, 4)
y_valid = to_categorical(y_valid, 4)
y_test = to_categorical(y_test_prep, 4)
print('Shape of training labels after categorical conversion:',y_train.shape
print('Shape of validation labels after categorical conversion:',y_valid.sha
print('Shape of test labels after categorical conversion:',y_test.shape)

# Adding width of the segment to be 1
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[
```

```
x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_train.shape[
x_test = X_test_prep.reshape(X_test_prep.shape[0], X_test_prep.shape[1], X_t
print('Shape of training set after adding width info:',x_train.shape)
print('Shape of validation set after adding width info:',x_valid.shape)
print('Shape of test set after adding width info:',x_test.shape)


# Reshaping the training and validation dataset
x_train = np.swapaxes(x_train, 1,3)
x_train = np.swapaxes(x_train, 1,2)
x_valid = np.swapaxes(x_valid, 1,3)
x_valid = np.swapaxes(x_valid, 1,2)
x_test = np.swapaxes(x_test, 1,3)
x_test = np.swapaxes(x_test, 1,2)
print('Shape of training set after dimension reshaping:',x_train.shape)
print('Shape of validation set after dimension reshaping:',x_valid.shape)
print('Shape of test set after dimension reshaping:',x_test.shape)

keras.backend.clear_session()
```

```
Shape of X after trimming: (1740, 22, 500)
Shape of X after maxpooling: (1740, 22, 250)
Shape of X after averaging+noise and concatenating: (3480, 22, 250)
Shape of X after subsampling and concatenating: (6960, 22, 250)
Shape of X after trimming: (375, 22, 500)
Shape of X after maxpooling: (375, 22, 250)
Shape of X after averaging+noise and concatenating: (750, 22, 250)
Shape of X after subsampling and concatenating: (1500, 22, 250)
Shape of X after trimming: (443, 22, 500)
Shape of X after maxpooling: (443, 22, 250)
Shape of X after averaging+noise and concatenating: (886, 22, 250)
Shape of X after subsampling and concatenating: (1772, 22, 250)
Shape of training set: (6960, 22, 250)
Shape of validation set: (1500, 22, 250)
Shape of training labels: (6960,)
Shape of validation labels: (1500,)
Shape of testing set: (1772, 22, 250)
Shape of testing labels: (1772,)
Shape of training labels after categorical conversion: (6960, 4)
Shape of validation labels after categorical conversion: (1500, 4)
Shape of test labels after categorical conversion: (1772, 4)
Shape of training set after adding width info: (6960, 22, 250, 1)
Shape of validation set after adding width info: (1500, 22, 250, 1)
Shape of test set after adding width info: (1772, 22, 250, 1)
Shape of training set after dimension reshaping: (6960, 250, 1, 22)
Shape of validation set after dimension reshaping: (1500, 250, 1, 22)
Shape of test set after dimension reshaping: (1772, 250, 1, 22)
```

(iii) (CNN-LSTM) Defining the architecture of the hybrid CNN-LSTM model

```
In [ ]: # Building the CNN model using sequential class
def build_model():
# models = []

        # Conv. block 1
    In1 = keras.Input(shape =(250,1,22) )
    c1 = Conv2D(filters=30, kernel_size=(11,1), padding='same', activation='
    p1 =  MaxPooling2D(pool_size=(4,1), padding='same')(c1) # Read the keras
    b1 = BatchNormalization()(p1)
    d1 = Dropout(0.5)(b1)

    # Conv. block 2
    c2 = Conv2D(filters=60, kernel_size=(9,1), padding='same', activation='s
    p2 =  MaxPooling2D(pool_size=(4,1), padding='same')(c2) # Read the keras
    b2 = BatchNormalization()(p2)
    d2 = Dropout(0.6)(b2)

    # Conv. block 3
    c3 = Conv2D(filters=120, kernel_size=(5,1), padding='same', activation='
    p3 =  MaxPooling2D(pool_size=(4,1), padding='same')(c3) # Read the keras
    b3 = BatchNormalization()(p3)
    d3 = Dropout(0.6)(b3)

    # Conv. block 4
    c4 = Conv2D(filters=240, kernel_size=(3,1), padding='same', activation='
    p4 =  MaxPooling2D(pool_size=(4,1), padding='same')(c4) # Read the keras
    b4 = BatchNormalization()(p4)
    d4 = Dropout(0.6)(b4)

    # FC+LSTM layers
    lstm  = Flatten()(d4) # Adding a flattening operation to the output of C
    lstm = Dense((120))(lstm) # FC layer with 100 units
    lstm = Reshape((120,1))(lstm) # Reshape my output of FC layer so that it
    lstm = LSTM(10, dropout=0.6, recurrent_dropout=0.1, input_shape=(120,1),

    # Output layer with Softmax activation
    fc = Dense(4, activation='softmax') (lstm) # Output FC layer with softma

    # Define the final model
    final_model = Model(inputs=In1, outputs=[fc])

    # # Printing the model summary
    final_model.compile(loss='categorical_crossentropy',
            optimizer='adam',
            metrics=['accuracy'])
    final_model.summary()
    return final_model
```

(iv) (CNN-LSTM) Defining the hyperparameters of the hybrid CNN-LSTM model

In [ ]:
```
# Model parameters
learning_rate = 1e-3
epochs = 100
hybrid_cnn_lstm_optimizer = keras.optimizers.Adam(lr=learning_rate)
```

```
/Users/lilyzhou/opt/anaconda3/envs/finalProject147/lib/python3.9/site-packa
ges/keras/optimizers/optimizer_v2/adam.py:114: UserWarning: The `lr` argume
nt is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

(v)(CNN-LSTM) Compiling, training and validating the model

In [ ]:
```
# Compiling the model
keras.backend.clear_session()

# Training and validating the model

batch_sizes = [32,64,128]
import matplotlib.pyplot as plt

hybrid_cnn_lstm_model = build_model()
# Compiling the model
hybrid_cnn_lstm_model.compile(loss='categorical_crossentropy',
               optimizer=hybrid_cnn_lstm_optimizer,
               metrics=['accuracy'])

# Training and validating the model

callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=20) #
hybrid_cnn_lstm_model_results = hybrid_cnn_lstm_model.fit(x_train,
                y_train,
                batch_size=32,
                epochs=epochs,
                validation_data=(x_valid, y_valid),
                verbose=True,
                callbacks = [callback])
```

```
2023-03-18 20:47:10.147668: I tensorflow/core/platform/cpu_feature_guard.c
c:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN) to use the following CPU instructions in performance-criti
cal operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 250, 1, 22)]      0         
                                                                 
 conv2d (Conv2D)             (None, 250, 1, 30)        7290      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 63, 1, 30)        0         
 )                                                               
                                                                 
 batch_normalization (BatchN  (None, 63, 1, 30)        120       
 ormalization)                                                   
                                                                 
 dropout (Dropout)           (None, 63, 1, 30)         0         
                                                                 
 conv2d_1 (Conv2D)           (None, 63, 1, 60)         16260     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 16, 1, 60)        0         
 2D)                                                             
                                                                 
 batch_normalization_1 (Batc  (None, 16, 1, 60)        240       
 hNormalization)                                                 
                                                                 
 dropout_1 (Dropout)         (None, 16, 1, 60)         0         
                                                                 
 conv2d_2 (Conv2D)           (None, 16, 1, 120)        36120     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 4, 1, 120)        0         
 2D)                                                             
                                                                 
 batch_normalization_2 (Batc  (None, 4, 1, 120)        480       
 hNormalization)                                                 
                                                                 
 dropout_2 (Dropout)         (None, 4, 1, 120)         0         
                                                                 
 conv2d_3 (Conv2D)           (None, 4, 1, 240)         86640     
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 1, 1, 240)        0         
 2D)                                                             
                                                                 
 batch_normalization_3 (Batc  (None, 1, 1, 240)        960       
 hNormalization)                                                 
                                                                 
 dropout_3 (Dropout)         (None, 1, 1, 240)         0         
                                                                 
 flatten (Flatten)           (None, 240)               0         
                                                                 
 dense (Dense)               (None, 120)               28920     
                                                                 
 reshape (Reshape)           (None, 120, 1)            0         
                                                                 
 lstm (LSTM)                 (None, 10)                480       
                                                                 
 dense_1 (Dense)             (None, 4)                 44        
```

```
=================================================================
Total params: 177,554
Trainable params: 176,654
Non-trainable params: 900
_________________________________________________________________
Epoch 1/100
218/218 [==============================] - 30s 126ms/step - loss: 1.3888 -
accuracy: 0.2530 - val_loss: 1.3825 - val_accuracy: 0.2587
Epoch 2/100
218/218 [==============================] - 26s 122ms/step - loss: 1.3784 -
accuracy: 0.2855 - val_loss: 1.3781 - val_accuracy: 0.2933
Epoch 3/100
218/218 [==============================] - 26s 121ms/step - loss: 1.3601 -
accuracy: 0.3174 - val_loss: 1.3142 - val_accuracy: 0.3740
Epoch 4/100
218/218 [==============================] - 27s 122ms/step - loss: 1.3147 -
accuracy: 0.3671 - val_loss: 1.3202 - val_accuracy: 0.3513
Epoch 5/100
218/218 [==============================] - 27s 122ms/step - loss: 1.2898 -
accuracy: 0.3856 - val_loss: 1.2761 - val_accuracy: 0.3987
Epoch 6/100
218/218 [==============================] - 28s 128ms/step - loss: 1.2602 -
accuracy: 0.4088 - val_loss: 1.2645 - val_accuracy: 0.4027
Epoch 7/100
218/218 [==============================] - 28s 127ms/step - loss: 1.2473 -
accuracy: 0.4164 - val_loss: 1.2164 - val_accuracy: 0.4293
Epoch 8/100
218/218 [==============================] - 28s 129ms/step - loss: 1.2289 -
accuracy: 0.4332 - val_loss: 1.1946 - val_accuracy: 0.4140
Epoch 9/100
218/218 [==============================] - 27s 126ms/step - loss: 1.2133 -
accuracy: 0.4376 - val_loss: 1.1867 - val_accuracy: 0.4393
Epoch 10/100
218/218 [==============================] - 28s 129ms/step - loss: 1.1955 -
accuracy: 0.4527 - val_loss: 1.1772 - val_accuracy: 0.4187
Epoch 11/100
218/218 [==============================] - 25s 115ms/step - loss: 1.1801 -
accuracy: 0.4611 - val_loss: 1.1564 - val_accuracy: 0.4753
Epoch 12/100
218/218 [==============================] - 24s 112ms/step - loss: 1.1599 -
accuracy: 0.4744 - val_loss: 1.1605 - val_accuracy: 0.4460
Epoch 13/100
218/218 [==============================] - 24s 111ms/step - loss: 1.1481 -
accuracy: 0.4839 - val_loss: 1.1220 - val_accuracy: 0.4713
Epoch 14/100
218/218 [==============================] - 28s 128ms/step - loss: 1.1344 -
accuracy: 0.4907 - val_loss: 1.1273 - val_accuracy: 0.4793
Epoch 15/100
218/218 [==============================] - 26s 117ms/step - loss: 1.1247 -
accuracy: 0.4894 - val_loss: 1.1509 - val_accuracy: 0.4580
Epoch 16/100
218/218 [==============================] - 27s 123ms/step - loss: 1.1074 -
accuracy: 0.4981 - val_loss: 1.1036 - val_accuracy: 0.4813
Epoch 17/100
218/218 [==============================] - 24s 112ms/step - loss: 1.0924 -
accuracy: 0.5076 - val_loss: 1.1046 - val_accuracy: 0.4793
```

```
Epoch 18/100
218/218 [==============================] - 26s 118ms/step - loss: 1.0796 -
accuracy: 0.5194 - val_loss: 1.1236 - val_accuracy: 0.4827
Epoch 19/100
218/218 [==============================] - 26s 118ms/step - loss: 1.0717 -
accuracy: 0.5162 - val_loss: 1.0904 - val_accuracy: 0.4873
Epoch 20/100
218/218 [==============================] - 26s 118ms/step - loss: 1.0579 -
accuracy: 0.5283 - val_loss: 1.1677 - val_accuracy: 0.4513
Epoch 21/100
218/218 [==============================] - 26s 119ms/step - loss: 1.0630 -
accuracy: 0.5341 - val_loss: 1.0984 - val_accuracy: 0.4867
Epoch 22/100
218/218 [==============================] - 24s 112ms/step - loss: 1.0345 -
accuracy: 0.5434 - val_loss: 1.1001 - val_accuracy: 0.4653
Epoch 23/100
218/218 [==============================] - 25s 113ms/step - loss: 1.0343 -
accuracy: 0.5447 - val_loss: 1.0842 - val_accuracy: 0.5007
Epoch 24/100
218/218 [==============================] - 25s 116ms/step - loss: 1.0207 -
accuracy: 0.5468 - val_loss: 1.1349 - val_accuracy: 0.4713
Epoch 25/100
218/218 [==============================] - 25s 116ms/step - loss: 1.0206 -
accuracy: 0.5563 - val_loss: 1.1191 - val_accuracy: 0.4733
Epoch 26/100
218/218 [==============================] - 27s 124ms/step - loss: 1.0016 -
accuracy: 0.5619 - val_loss: 1.0863 - val_accuracy: 0.4967
Epoch 27/100
218/218 [==============================] - 25s 117ms/step - loss: 1.0061 -
accuracy: 0.5585 - val_loss: 1.0849 - val_accuracy: 0.4847
Epoch 28/100
218/218 [==============================] - 25s 117ms/step - loss: 0.9898 -
accuracy: 0.5644 - val_loss: 1.0872 - val_accuracy: 0.4927
Epoch 29/100
218/218 [==============================] - 25s 116ms/step - loss: 0.9812 -
accuracy: 0.5615 - val_loss: 1.1203 - val_accuracy: 0.4833
Epoch 30/100
218/218 [==============================] - 26s 119ms/step - loss: 0.9754 -
accuracy: 0.5721 - val_loss: 1.0724 - val_accuracy: 0.5260
Epoch 31/100
218/218 [==============================] - 26s 119ms/step - loss: 0.9756 -
accuracy: 0.5720 - val_loss: 1.1420 - val_accuracy: 0.4860
Epoch 32/100
218/218 [==============================] - 26s 118ms/step - loss: 0.9658 -
accuracy: 0.5843 - val_loss: 1.0819 - val_accuracy: 0.4993
Epoch 33/100
218/218 [==============================] - 29s 135ms/step - loss: 0.9469 -
accuracy: 0.5874 - val_loss: 1.0912 - val_accuracy: 0.5160
Epoch 34/100
218/218 [==============================] - 39s 178ms/step - loss: 0.9495 -
accuracy: 0.5861 - val_loss: 1.0730 - val_accuracy: 0.5173
Epoch 35/100
218/218 [==============================] - 28s 126ms/step - loss: 0.9444 -
accuracy: 0.5948 - val_loss: 1.1131 - val_accuracy: 0.5193
Epoch 36/100
218/218 [==============================] - 27s 122ms/step - loss: 0.9452 -
```

```
accuracy: 0.5922 - val_loss: 1.0920 - val_accuracy: 0.5173
Epoch 37/100
218/218 [==============================] - 25s 114ms/step - loss: 0.9203 -
accuracy: 0.6010 - val_loss: 1.0875 - val_accuracy: 0.5147
Epoch 38/100
218/218 [==============================] - 28s 130ms/step - loss: 0.9207 -
accuracy: 0.6080 - val_loss: 1.1145 - val_accuracy: 0.5273
Epoch 39/100
218/218 [==============================] - 33s 151ms/step - loss: 0.9098 -
accuracy: 0.6092 - val_loss: 1.1204 - val_accuracy: 0.5133
Epoch 40/100
218/218 [==============================] - 37s 172ms/step - loss: 0.9177 -
accuracy: 0.6043 - val_loss: 1.1075 - val_accuracy: 0.5213
Epoch 41/100
218/218 [==============================] - 39s 178ms/step - loss: 0.9066 -
accuracy: 0.6154 - val_loss: 1.0882 - val_accuracy: 0.5380
Epoch 42/100
218/218 [==============================] - 39s 180ms/step - loss: 0.8933 -
accuracy: 0.6157 - val_loss: 1.0898 - val_accuracy: 0.5267
Epoch 43/100
218/218 [==============================] - 43s 195ms/step - loss: 0.8898 -
accuracy: 0.6261 - val_loss: 1.1282 - val_accuracy: 0.5333
Epoch 44/100
218/218 [==============================] - 36s 167ms/step - loss: 0.8853 -
accuracy: 0.6177 - val_loss: 1.1274 - val_accuracy: 0.5453
Epoch 45/100
218/218 [==============================] - 26s 121ms/step - loss: 0.8816 -
accuracy: 0.6224 - val_loss: 1.1006 - val_accuracy: 0.5440
Epoch 46/100
218/218 [==============================] - 28s 128ms/step - loss: 0.8881 -
accuracy: 0.6332 - val_loss: 1.0269 - val_accuracy: 0.6000
Epoch 47/100
218/218 [==============================] - 34s 157ms/step - loss: 0.8632 -
accuracy: 0.6425 - val_loss: 1.1025 - val_accuracy: 0.5507
Epoch 48/100
218/218 [==============================] - 46s 213ms/step - loss: 0.8664 -
accuracy: 0.6392 - val_loss: 1.0858 - val_accuracy: 0.5747
Epoch 49/100
218/218 [==============================] - 44s 200ms/step - loss: 0.8622 -
accuracy: 0.6404 - val_loss: 1.0234 - val_accuracy: 0.5833
Epoch 50/100
218/218 [==============================] - 42s 194ms/step - loss: 0.8523 -
accuracy: 0.6526 - val_loss: 1.0947 - val_accuracy: 0.5593
Epoch 51/100
218/218 [==============================] - 40s 183ms/step - loss: 0.8420 -
accuracy: 0.6557 - val_loss: 1.0424 - val_accuracy: 0.5860
Epoch 52/100
218/218 [==============================] - 43s 196ms/step - loss: 0.8377 -
accuracy: 0.6582 - val_loss: 1.0468 - val_accuracy: 0.5927
Epoch 53/100
218/218 [==============================] - 43s 196ms/step - loss: 0.8355 -
accuracy: 0.6658 - val_loss: 1.1237 - val_accuracy: 0.5513
Epoch 54/100
218/218 [==============================] - 41s 189ms/step - loss: 0.8258 -
accuracy: 0.6690 - val_loss: 1.0386 - val_accuracy: 0.6073
Epoch 55/100
```

```
218/218 [==============================] - 41s 188ms/step - loss: 0.8243 -
accuracy: 0.6658 - val_loss: 1.0317 - val_accuracy: 0.6053
Epoch 56/100
218/218 [==============================] - 38s 173ms/step - loss: 0.8162 -
accuracy: 0.6700 - val_loss: 1.0584 - val_accuracy: 0.5920
Epoch 57/100
218/218 [==============================] - 38s 172ms/step - loss: 0.8020 -
accuracy: 0.6823 - val_loss: 1.0689 - val_accuracy: 0.6193
Epoch 58/100
218/218 [==============================] - 38s 173ms/step - loss: 0.8004 -
accuracy: 0.6826 - val_loss: 1.0095 - val_accuracy: 0.6267
Epoch 59/100
218/218 [==============================] - 36s 166ms/step - loss: 0.7959 -
accuracy: 0.6931 - val_loss: 1.0293 - val_accuracy: 0.6320
Epoch 60/100
218/218 [==============================] - 45s 204ms/step - loss: 0.8029 -
accuracy: 0.6886 - val_loss: 0.9983 - val_accuracy: 0.6167
Epoch 61/100
218/218 [==============================] - 41s 190ms/step - loss: 0.7871 -
accuracy: 0.6935 - val_loss: 0.9741 - val_accuracy: 0.6433
Epoch 62/100
218/218 [==============================] - 40s 183ms/step - loss: 0.7792 -
accuracy: 0.6934 - val_loss: 1.0127 - val_accuracy: 0.6187
Epoch 63/100
218/218 [==============================] - 41s 190ms/step - loss: 0.7799 -
accuracy: 0.6950 - val_loss: 0.9592 - val_accuracy: 0.6427
Epoch 64/100
218/218 [==============================] - 41s 189ms/step - loss: 0.7686 -
accuracy: 0.6986 - val_loss: 0.9687 - val_accuracy: 0.6373
Epoch 65/100
218/218 [==============================] - 41s 189ms/step - loss: 0.7457 -
accuracy: 0.7108 - val_loss: 0.9956 - val_accuracy: 0.6433
Epoch 66/100
218/218 [==============================] - 42s 194ms/step - loss: 0.7464 -
accuracy: 0.7151 - val_loss: 1.0392 - val_accuracy: 0.6293
Epoch 67/100
218/218 [==============================] - 42s 193ms/step - loss: 0.7421 -
accuracy: 0.7152 - val_loss: 1.0770 - val_accuracy: 0.6227
Epoch 68/100
218/218 [==============================] - 43s 197ms/step - loss: 0.7391 -
accuracy: 0.7131 - val_loss: 1.0092 - val_accuracy: 0.6207
Epoch 69/100
218/218 [==============================] - 40s 185ms/step - loss: 0.7318 -
accuracy: 0.7220 - val_loss: 1.0386 - val_accuracy: 0.6273
Epoch 70/100
218/218 [==============================] - 37s 172ms/step - loss: 0.7245 -
accuracy: 0.7266 - val_loss: 0.9895 - val_accuracy: 0.6320
Epoch 71/100
218/218 [==============================] - 24s 111ms/step - loss: 0.7143 -
accuracy: 0.7325 - val_loss: 0.9589 - val_accuracy: 0.6407
Epoch 72/100
218/218 [==============================] - 25s 117ms/step - loss: 0.7122 -
accuracy: 0.7282 - val_loss: 0.9545 - val_accuracy: 0.6407
Epoch 73/100
218/218 [==============================] - 25s 117ms/step - loss: 0.7008 -
accuracy: 0.7346 - val_loss: 0.9777 - val_accuracy: 0.6527
```

```
Epoch 74/100
218/218 [==============================] - 25s 112ms/step - loss: 0.6979 -
accuracy: 0.7362 - val_loss: 0.9727 - val_accuracy: 0.6480
Epoch 75/100
218/218 [==============================] - 24s 111ms/step - loss: 0.6917 -
accuracy: 0.7362 - val_loss: 0.9871 - val_accuracy: 0.6313
Epoch 76/100
218/218 [==============================] - 22s 99ms/step - loss: 0.6941 - a
ccuracy: 0.7371 - val_loss: 0.9648 - val_accuracy: 0.6360
Epoch 77/100
218/218 [==============================] - 19s 85ms/step - loss: 0.7047 - a
ccuracy: 0.7330 - val_loss: 0.9653 - val_accuracy: 0.6480
Epoch 78/100
218/218 [==============================] - 18s 84ms/step - loss: 0.6903 - a
ccuracy: 0.7386 - val_loss: 1.0132 - val_accuracy: 0.6167
Epoch 79/100
218/218 [==============================] - 18s 84ms/step - loss: 0.6685 - a
ccuracy: 0.7486 - val_loss: 1.0329 - val_accuracy: 0.6287
Epoch 80/100
218/218 [==============================] - 18s 85ms/step - loss: 0.6888 - a
ccuracy: 0.7404 - val_loss: 0.9615 - val_accuracy: 0.6353
Epoch 81/100
218/218 [==============================] - 19s 85ms/step - loss: 0.6706 - a
ccuracy: 0.7447 - val_loss: 0.9342 - val_accuracy: 0.6507
Epoch 82/100
218/218 [==============================] - 18s 84ms/step - loss: 0.6798 - a
ccuracy: 0.7468 - val_loss: 0.9883 - val_accuracy: 0.6340
Epoch 83/100
218/218 [==============================] - 18s 83ms/step - loss: 0.6730 - a
ccuracy: 0.7444 - val_loss: 0.9681 - val_accuracy: 0.6520
Epoch 84/100
218/218 [==============================] - 19s 85ms/step - loss: 0.6567 - a
ccuracy: 0.7540 - val_loss: 0.9574 - val_accuracy: 0.6513
Epoch 85/100
218/218 [==============================] - 18s 84ms/step - loss: 0.6687 - a
ccuracy: 0.7473 - val_loss: 0.9385 - val_accuracy: 0.6567
Epoch 86/100
218/218 [==============================] - 18s 84ms/step - loss: 0.6514 - a
ccuracy: 0.7547 - val_loss: 0.9621 - val_accuracy: 0.6527
Epoch 87/100
218/218 [==============================] - 16s 71ms/step - loss: 0.6498 - a
ccuracy: 0.7524 - val_loss: 0.9330 - val_accuracy: 0.6667
Epoch 88/100
218/218 [==============================] - 14s 66ms/step - loss: 0.6520 - a
ccuracy: 0.7547 - val_loss: 0.9660 - val_accuracy: 0.6340
Epoch 89/100
218/218 [==============================] - 15s 67ms/step - loss: 0.6533 - a
ccuracy: 0.7517 - val_loss: 0.9687 - val_accuracy: 0.6473
Epoch 90/100
218/218 [==============================] - 15s 66ms/step - loss: 0.6489 - a
ccuracy: 0.7547 - val_loss: 0.9672 - val_accuracy: 0.6547
Epoch 91/100
218/218 [==============================] - 15s 67ms/step - loss: 0.6363 - a
ccuracy: 0.7588 - val_loss: 0.9459 - val_accuracy: 0.6413
Epoch 92/100
218/218 [==============================] - 15s 67ms/step - loss: 0.6472 - a
```

```
ccuracy: 0.7585 - val_loss: 0.8979 - val_accuracy: 0.6673
Epoch 93/100
218/218 [==============================] - 15s 69ms/step - loss: 0.6253 - a
ccuracy: 0.7611 - val_loss: 0.8922 - val_accuracy: 0.6793
Epoch 94/100
218/218 [==============================] - 14s 66ms/step - loss: 0.6239 - a
ccuracy: 0.7651 - val_loss: 0.9626 - val_accuracy: 0.6453
Epoch 95/100
218/218 [==============================] - 13s 62ms/step - loss: 0.6283 - a
ccuracy: 0.7675 - val_loss: 0.9193 - val_accuracy: 0.6667
Epoch 96/100
218/218 [==============================] - 13s 61ms/step - loss: 0.6360 - a
ccuracy: 0.7573 - val_loss: 0.9163 - val_accuracy: 0.6633
Epoch 97/100
218/218 [==============================] - 13s 61ms/step - loss: 0.6319 - a
ccuracy: 0.7651 - val_loss: 0.9545 - val_accuracy: 0.6380
Epoch 98/100
218/218 [==============================] - 13s 61ms/step - loss: 0.6256 - a
ccuracy: 0.7635 - val_loss: 0.8699 - val_accuracy: 0.6860
Epoch 99/100
218/218 [==============================] - 13s 62ms/step - loss: 0.6233 - a
ccuracy: 0.7670 - val_loss: 0.8874 - val_accuracy: 0.6880
Epoch 100/100
218/218 [==============================] - 14s 64ms/step - loss: 0.6180 - a
ccuracy: 0.7684 - val_loss: 0.8565 - val_accuracy: 0.6860
```

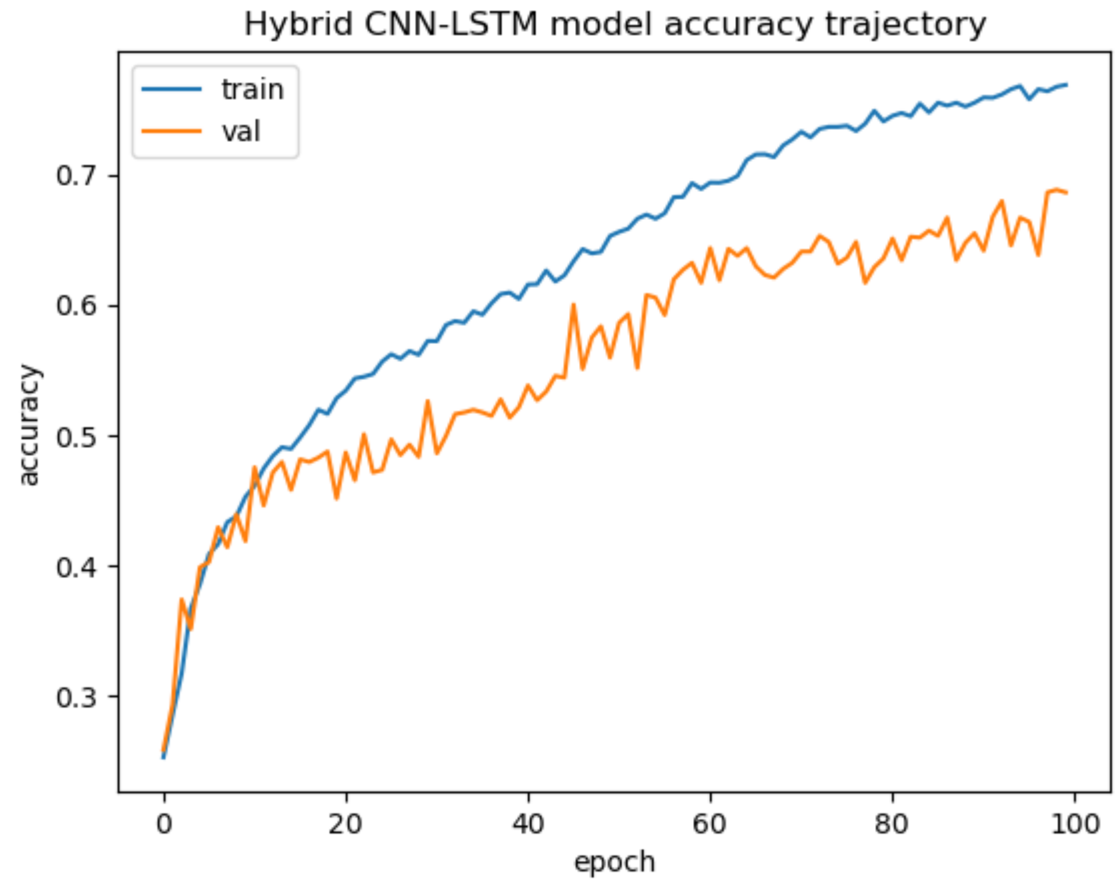
(vi)(CNN-LSTM) Visualizing the accuracy and loss trajectory

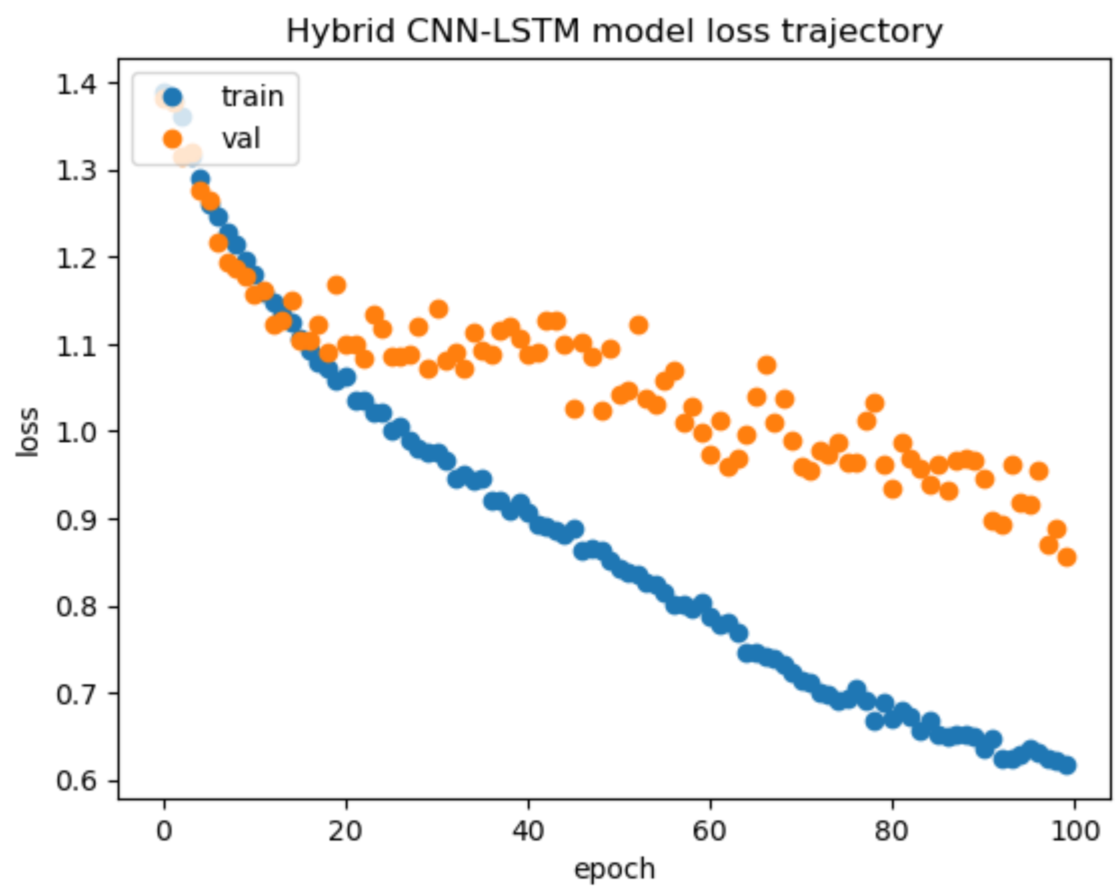
In [ ]:

```python
import matplotlib.pyplot as plt

# Plotting accuracy trajectory
plt.plot(hybrid_cnn_lstm_model_results.history['accuracy'])
plt.plot(hybrid_cnn_lstm_model_results.history['val_accuracy'])
plt.title('Hybrid CNN-LSTM model accuracy trajectory')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Plotting loss trajectory
plt.plot(hybrid_cnn_lstm_model_results.history['loss'],'o')
plt.plot(hybrid_cnn_lstm_model_results.history['val_loss'],'o')
plt.title('Hybrid CNN-LSTM model loss trajectory')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```


Hybrid CNN-LSTM model accuracy trajectory
train
val
accuracy
epoch
0.3
0.4
0.5
0.6
0.7
0
20
40
60
80
100


Hybrid CNN-LSTM model loss trajectory
train
val
loss
epoch
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.4
0
20
40
60
80
100

(vii) (CNN-LSTM) Testing the performance of the hybrid CNN-LSTM model on the held out test set

In [ ]:
```
## Testing the hybrid CNN-LSTM model

hybrid_cnn_lstm_score = hybrid_cnn_lstm_model.evaluate(x_test, y_test, verbo
print('Test accuracy of the hybrid CNN-LSTM model:',hybrid_cnn_lstm_score[1]
```

```
Test accuracy of the hybrid CNN-LSTM model: 0.6834085583686829
```

In [ ]: