Real-Time Classification of f-NIRS data to Develop a Brain Computer Interface using Deep Learning

A Thesis

Submitted By

Avinash Kumar Trivedi

For the award of the degree of

MASTER OF TECHNOLOGY, CLINICAL ENGINEERING

Jointly offered by

Is evaluated and approved by

Dr. C. Kesavadas
(Guide)

Dr. Sujesh Sreedharan
(Co-Guide)

THESIS CERTIFICATE

This is to certify that the thesis entitled “Real-time classification of f-NIRS data to develop a brain computer interface using deep learning” submitted by **Avinash Kumar Trivedi** to SCTIMST Trivandrum, for the award of the degree of Master of Technology in Clinical Engineering jointly offered by IIT Madras, CMC Vellore, and SCTIMST Trivandrum is a bonafide record of research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

ACKNOWLEDGMENT

I would like to express my gratitude to my primary guide, Dr. C. Kesavadas, who accepted me to work on the f-NIRS project along with my co-guide, Dr. Sujesh Sreedharan, who mentored me throughout this project. With Dr. Sujesh's expertise and encouragement, I was able to reach a conclusion for the thesis. I wish to acknowledge the help provided by the technical and project staff in the Division of Artificial Internal Organs of BMT Wing, SCTIMST. I would also like to show my deep appreciation to Saranya who helped me in acquiring the data for the thesis. I am also grateful for all the love, support, and encouragement that I received from my family and my friends without whom I would never have come so far in my life.

Avinash Kumar Trivedi
BT20M021

Contents

List of Figures

1. Abstract

The thesis aims to classify motor tasks performed by a participant based on the signals obtained from the brain with the help of a Continuous Wave functional Near Infrared Spectroscopy (CW-f-NIRS) device by using deep learning techniques. The goal of the exercise is to contribute towards development of a real time Brain-Computer Interface (BCI) using f-NIRS technology.

For achieving this goal, f-NIRS data were obtained from one participant performing a 10 second task (left, right, both-hands) along with 25 seconds of rest in between two consecutive tasks. The data was obtained for 10 sessions. The data was processed and fed through a deep learning architecture to predict the task performed. For this purpose, three deep learning models were developed: CNN (Convolutional Neural Network), CNN-LSTM (Long Short-Term Memory), and CNN-GRU (Gate Recurrent Unit). Since our purpose was to label the accuracy of each task individually, we used a 4-class classification. To summarize the results, a confusion matrix was created to obtain precision, recall, and f1-score for each class. Since the performance of any deep learning model was not always superior for a given testing session. I created a deep ensemble model, which uses all three above models for prediction and outputs a result better than or equal to the best of 3 predictions. Another set of f-NIRS data for 10 sessions was then obtained for a new participant to check for repeatability of the experiment.

2. Introduction

2.1 f-NIRS

Functional near-infrared spectroscopy (f-NIRS) detects the changes in hemoglobin level inside the brain with the principle of optical absorption. Within the near-infrared spectrum, light can penetrate biological tissues and be absorbed by chromophores, such as oxyhemoglobin and deoxyhemoglobin. Functional magnetic resonance imaging(fMRI) also works on the same principle of detecting the changes in hemoglobin level. While limited by its inferior spatial resolution and penetration depth, f-NIRS has a much higher temporal resolution than fMRI.
Some more advantageous aspects of this technology are its portability, cost-effectiveness, and potential for long-term monitoring.

2.2 BCI

A brain-computer interface (BCI) is a communication system that allows the use of brain activity to control computers or other external devices. It can, by bypassing the peripheral nervous system, provide a means of communication for people suffering from severe motor disabilities or in a persistent vegetative state. The most frequently used portable BCI devices are based on electroencephalography (EEG). Although EEG provides excellent temporal resolution, making it ideal for real-time applications, the technology suffers from poor spatial resolution and an inherent sensitivity to motion artifacts. F-NIRS can be a better alternative as it provides a better compromise between spatial and temporal resolutions. A general pipeline of a f-NIRS based BCI system is shown below:

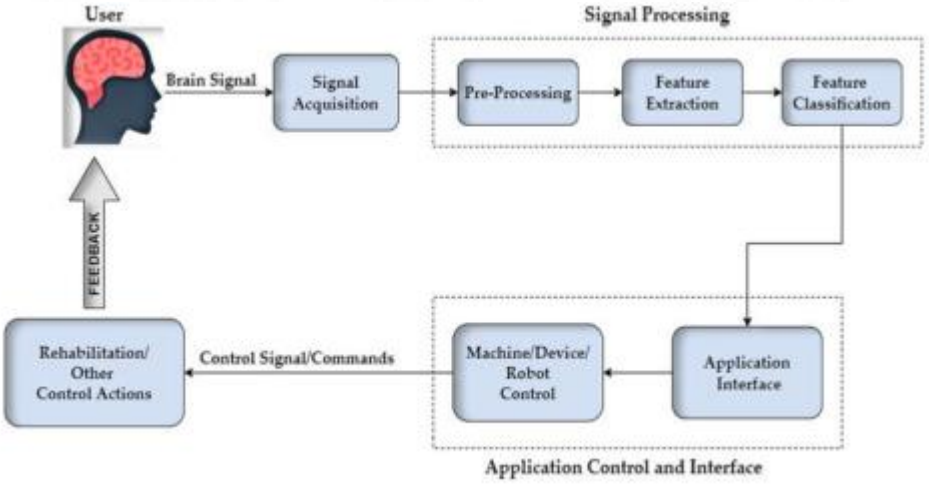


Figure 1. Basic design and operation of the brain-computer interface (BCI)-based control

In the first phase of the thesis, I worked on the pre-processing, feature extraction, and feature classification to improve the classification accuracy of performed motor tasks. Then I collected new data from a participant to repeat the experiment and compare the results.

The future scope of the work would then be designing a BCI application for controlling a maze-game in real-time.

2.3 Deep Learning for Time series data

Time series are present in many real-world applications ranging from health care, human activity recognition, cyber-security, finance, marketing, automated disease detection, anomaly detection, etc. f-NIRS data is also in the form of a time series on which activity recognition must

be done. In our case, we are also dealing with a multivariate time series, which are acquired through the f-NIRS device.

There are a variety of deep learning techniques and architectures available for multivariate time series classification problems. Before delving deep into different architectures, let's look at a Perceptron:

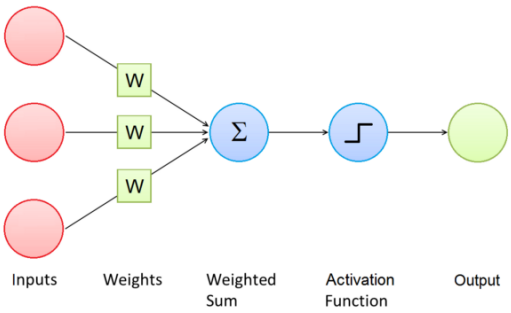


Figure 2: A perceptron

A Perceptron has one or more input values, and every input value is associated to a weight. The goal of a Perceptron is to compute the weighted sum of the input values and then apply an activation function to the result. The most common activation functions are sigmoid, hyperbolic tangent and rectifier. The result of the activation function is referred to as the activation of the Perceptron and represents its output value.

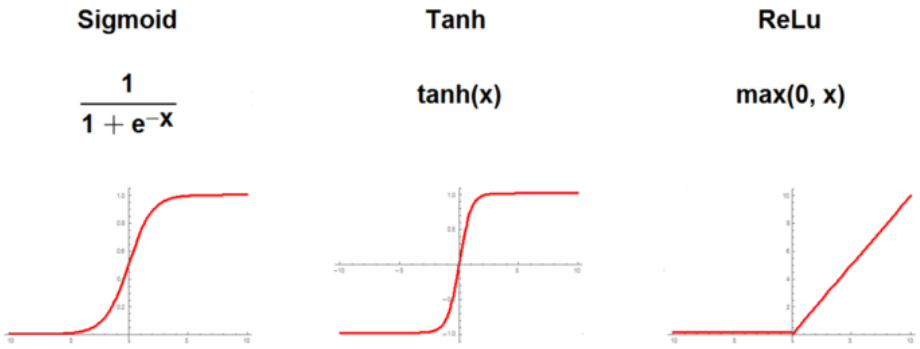


Figure 3: Activation functions

2.3.1 Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron (MLP) is a building block used in many Deep Learning Architectures for Time Series Classification. It is made up of many Perceptrons where each node in the MLP is a Perceptron. It is a class of feedforward neural networks and consists of several layers of nodes: one input layer, one or more hidden layers, and one output layer. Every node is connected to all

the nodes of its layer, of the previous layer and of the next layer. For this reason, we say that the Multi-Layer Perceptron is fully connected.

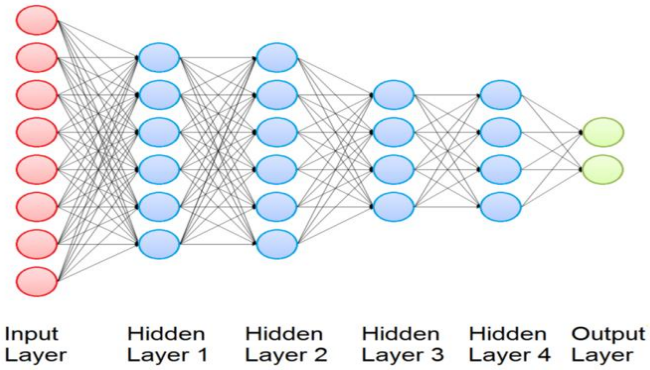


Figure 4: Multi-Layer Perceptron

An MLP can be used to solve classification problems. However, in case of time series data, the size of data is a bottleneck for computational speeds. Furthermore, whole input to an MLP is represented as a one-dimensional vector which compromises the automatic feature extraction of deep learning networks in case of temporal data.

Below figure shows the general deep learning framework for time series classification.

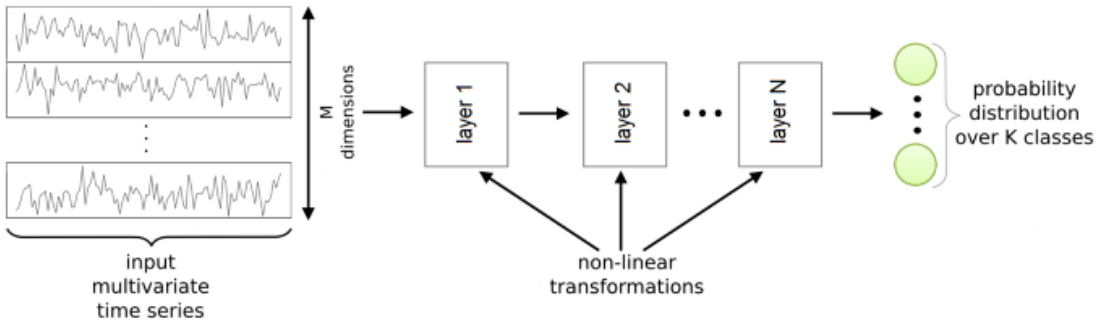


Figure 5: General deep learning framework for TSC

Every layer takes as input the output of the previous layer and applies its non-linear transformation to compute its own output. The behavior of these non-linear transformations is controlled by a set of parameters for each layer. These parameters link the input of the layer to its output and are trainable (like the weights of the Multi-Layer Perceptron). Often, the last layer is a Multi-Layer Perceptron. These nonlinear transformation blocks can be modeled using CNNs or RNNs (Recurrent Neural Networks).

2.3.2 Convolutional Neural Networks (CNNs)

A Convolutional Neural Network is a Deep Learning algorithm that takes as input an image or a time series and can successfully capture the spatial and temporal patterns through the application trainable filters and assigns importance to these patterns using trainable weights. The preprocessing required in a Convolutional Neural Network is much lower as compared to other classification algorithms.

2.3.3 Recurrent Neural Networks (RNNs)

RNN works on the principle of saving the output of a particular layer and feeding this back to the input to predict the output of the layer. RNNs were created because there were a few issues in the feed-forward neural network:

- Cannot handle sequential data
- Considers only the current input
- Cannot memorize previous inputs

The solution to these issues is the RNN. An RNN can handle sequential data, accepting the current input data, and previously receive inputs. RNNs can memorize previous inputs due to their internal memory.

- **LSTM (Long Short-Term Memory)** network models are a type of recurrent neural network that can learn and remember over long sequences of input data. The benefit of using LSTMs for sequence classification is that they can learn from the raw time series data directly, and in turn do not require domain expertise to manually engineer input features. The model can learn an internal representation of the time series data and ideally achieve comparable performance to models fit on a version of the dataset with engineered features.
- **GRU (Gated Recurrent Unit)**, is a type of RNN which aims to solve the vanishing gradient problem which comes with a standard recurrent neural network. GRU can also be considered as a variation on the LSTM because both are designed similarly and, in some cases, produce equally excellent results.

In this thesis, Keras is used for deep learning model training and predictions. Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow.
Using the sequential model we can assign our input layer, hidden layers, and output layer on top of each other.

2.3.4 Parameters of a deep learning model

These are the coefficients of the model, and they are chosen by the model itself. It means that the algorithm, while learning, optimizes these coefficients (according to a given optimization strategy) and returns an array of parameters which minimize the error. To give an example, in a linear regression task, you have your model that will look like y=b + ax, where b and a will be your parameter. The only thing you have to do with those parameters is initialize them which the model handles by itself.

2.3.5 Hyper-parameters of a deep learning model

Hyperparameters are the variables which determine the network structure and the variables which determine how the network is trained. Hyperparameters are set before training (before optimizing the weights and bias).

- **Number of Hidden Layers and units**
  Hidden layers are the layers between input layer and output layer. Many hidden units within a layer with regularization techniques can increase accuracy. Smaller number of units may cause underfitting.
- **Dropout**
  Dropout is a regularization technique to avoid overfitting (increase the validation accuracy) thus increasing the generalizing power. Dropout refers to the percentage of neurons that will be randomly removed from consideration from the next layer of the model.
    - Generally, using a small dropout value of 20%-50% of neurons with 20% provides a good starting point. A probability too low has minimal effect and a value too high results in under-learning by the network.
    - Use a larger network. You are likely to get better performance when dropout is used on a larger network, giving the model more of an opportunity to learn independent representations.
- **Learning rate**
  The learning rate defines how quickly a network updates its parameters. Low learning rate slows down the learning process but converges smoothly. Larger learning rate speeds up the learning but may not converge.
- **Number of epochs**
  Number of epochs is the number of times the whole training data is shown to the network while training. The general rule is to increase the number of epochs until the validation accuracy starts decreasing even when training accuracy is increasing(overfitting).

- **Batch size**
  Batch size is the number of sub samples given to the network after which parameter update happens. It is the number of samples that will be passed through to the network at one time.

2.4 Making sense of results

Once we can train our model and start to predict the hand movements, we need to devise a methodology for classifying the accuracy of our results. Since we are dealing with a multi-class classification problem, we need to differentiate the accuracy for all classes individually. For this purpose, we can use a confusion matrix and calculate precision, recall, and f1-score for each class.

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Figure 6: Confusion matrix for a binary classification

Above is a confusion matrix for a binary class classification problem.

- True Positive (TP) — model correctly predicts the positive class (prediction and actual both are positive).
- True Negative (TN) — model correctly predicts the negative class (prediction and actual both are negative).
- False Positive (FP) — model gives the wrong prediction of the negative class (predicted-positive, actual-negative). FP is also called a TYPE I error.
- False Negative (FN) — model wrongly predicts the positive class (predicted-negative, actual-positive). FN is also called a TYPE II error.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$F1\ score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 * (Precision * Recall)}{(Precision + Recall)}$$

Formula 1: Formulas for Precision, Recall, and F1-Score

Based on it, we can define precision, recall, and f1-score. Precision is out of all the positives predicted, what percentage is truly positive. Recall is out of the total positive, what percentage are predicted positive. F1 score is the harmonic mean of precision and recall. It takes both false positives and false negatives into account. Therefore, it performs well on an imbalanced dataset.

3. Methodology

3.1 Experimental Paradigm

The study consists of two participants, from whom data was obtained. The data from the first participant was already available while I collected the data from the second participant with the help of Saranya to verify repeatability of the experiment. f-NIRS signals were acquired from the primary motor cortex (M1) in the left and right hemispheres. A single experiment contains 105 seconds of data. The data consists of 10 sessions, each session consists of the experiment repeated 5 times.

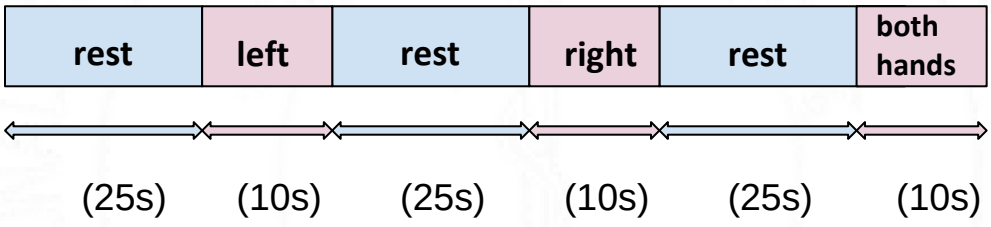


Figure 7. Visualization of a single experiment (105 seconds)

The participant performs these motor execution tasks based on the prompt provided on the screen in front of them.

3.2 System and Acquisition

The signals were acquired by multi-channel continuous-wave imaging system NIRx NIRSport which is developed by NIRx Medical Technologies. The device operates at two infrared frequencies, 760 nm (for oxygenated hemoglobin) and 850 nm (for deoxygenated hemoglobin) with a sampling frequency of 7.8125 Hz. Figure 3 shows the optode placements for signal acquisition. The optode placement corresponds to 10-20 system of EEG electrode placements. Red dots indicate sources and blue dots indicate detectors. There are 20 channels formed for each frequency level resulting in data coming from 40 channels simultaneously.

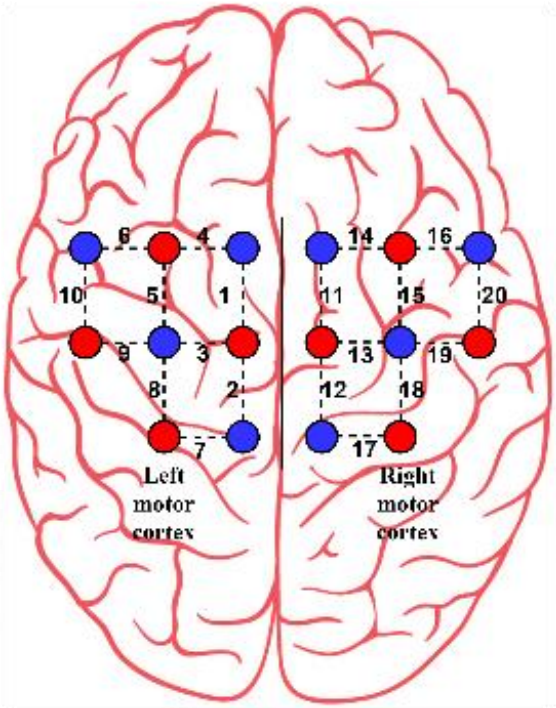


Figure 8. Optode placement for the study

3.3. Data Preprocessing

Data preprocessing has been performed in MATLAB using the NIRS Toolbox package available in open-source platforms. There are several preprocessing steps that were performed on top of each other before the data is sent to a deep learning classifier.

Available raw data from 10 sessions are loaded into the MATLAB program in the beginning. The preprocessing steps can be subdivided into:

- Data Cleaning
  Available data also consists of data points before the beginning and after the end of the session. So, a ***TrimBaseline()*** function was used to remove extra signals.

- Conversion from raw data to hemoglobin concentration
  The raw data was first converted to optical density and then hemodynamic concentration by applying the modified Beer-Lambert's law. The functions used to achieve this in the NIRx Toolbox package are ***OpticalDensity()*** and ***BeerLambertLaw()***.

- Data Filtering

  The converted raw data into hemoglobin concentration was then filtered using two different values for the low cut-off frequency of the band-pass filter. Two frequencies chosen were: (0.01Hz-0.2Hz) and (0.01Hz-1Hz).

  A wavelet filter was also applied as the last step of data pre-processing on the filtered data for correcting errors due to sudden motion. The use of a wavelet filter didn't improve the accuracy. The applied wavelet filter comes out of the NIRS toolbox ***WaveletFilter()*** which uses a sym4 wavelet.

3.4. Deep learning for activity recognition

Let's define our classification problem in terms of dimensionality. We have a multivariate time series, meaning we have data points coming from 40 different channels at the same instant. In a single experiment (105 sec), there would be 7.8125 multiplied by 105 data points which is approximately 820 data points for one experiment. So, as of now we can think of data as a 2-d array (820,40) from one experiment. For an entire session, the total number of data points vary given the sampling frequency.

3.4.1 Data Preparation for model training

From the 10 sessions of data available, firstly the dataset is split into training and testing datasets. In each iteration of model training and testing, one of the sessions is used as the testing data and the rest of the sessions are used as training data. In this manner, a model is trained for 10 iterations and an average accuracy of the model can be determined.

Currently, we will have data of shape *(number of data points, number of channels)*. From this we will create instances of data in the form of overlapping windows to be fed as an input to the neural network one by one. So, one row of data will be shaped *(window length, no. of channels)* and total number of rows will be equal to no. of data points. This one row of data can serve as an input to a CNN or RNN model.

To assign the label or target class to an input, we used the "first 2 seconds approach", in which a particular input window will be labeled according to the task present in the first 2 seconds of the window length. Based on manual tuning and optimization, the values of hyperparameters and other variables are chosen as:

Epochs	30, 40(similar results)
Batch Size	32, 64, 128 (similar results)
Window length	9 seconds (72 points)
Label window	2 seconds (16 points)
Learning rate	0.0025

Table 1: Tuned hyperparameters for models

Three different deep learning architectures and a deep ensemble model based on the first three models were used for prediction. The 2-layer CNN model was chosen after comparing it with 1 layer and 2-layer as well during manual hyperparameter searching. The deep learning models developed are:

A. 2-layer Conv1D

```
input_layer = keras.layers.Input(input_shape)
#Layer 1
conv1 = keras.layers.Conv1D(64, kernel_size=8, activation='relu',padding='same')(input_layer)
conv1 = keras.layers.BatchNormalization()(conv1)
conv1 = keras.layers.MaxPool1D(pool_size =3)(conv1)
conv1 = keras.layers.Dropout(0.5)(conv1)
#Layer 2
conv2 = keras.layers.Conv1D(64, kernel_size=8, activation='relu',padding='same')(conv1)
conv2 = keras.layers.BatchNormalization()(conv2)
conv2 = keras.layers.MaxPool1D(pool_size = 5)(conv2)
conv2 = keras.layers.Dropout(0.5)(conv2)

flatten = keras.layers.Flatten()(conv2)

dense1 = keras.layers.Dense(100, activation='relu')(flatten)
output_layer = keras.layers.Dense(4, activation="softmax")(dense1)
```

Figure 9: 2 CNN layers with a filter size of 64, flattened and connected with a fully connected dense layer.

B. 2-layer Conv1D - LSTM

```
input_layer = keras.layers.Input(input_shape)
#Layer 1
conv1 = keras.layers.Conv1D(64, kernel_size=8, activation='relu',padding='same')(input_layer)
conv1 = keras.layers.BatchNormalization()(conv1)
conv1 = keras.layers.MaxPool1D(pool_size =3)(conv1)
conv1 = keras.layers.Dropout(0.5)(conv1)
#Layer 2
conv2 = keras.layers.Conv1D(64, kernel_size=8, activation='relu',padding='same')(conv1)
conv2 = keras.layers.BatchNormalization()(conv2)
conv2 = keras.layers.MaxPool1D(pool_size = 5)(conv2)
conv2 = keras.layers.Dropout(0.5)(conv2)

lstm = keras.layers.LSTM(64, return_sequences=True)(conv2)
lstm = keras.layers.Flatten()(lstm)
output_layer = keras.layers.Dense(4, activation="softmax")(lstm)
```

Figure 10: 2 CNN layers with a filter size of 64, which is used as a feature extraction and then fed as an input to a LSTM network.

C. 2-layer Conv1D - GRU

```
input_layer = keras.layers.Input(input_shape)
#Layer 1
conv1 = keras.layers.Conv1D(64, kernel_size=8, activation='relu',padding='same')(input_layer)
conv1 = keras.layers.BatchNormalization()(conv1)
conv1 = keras.layers.MaxPool1D(pool_size =3)(conv1)
conv1 = keras.layers.Dropout(0.5)(conv1)
#Layer 2
conv2 = keras.layers.Conv1D(64, kernel_size=8, activation='relu',padding='same')(conv1)
conv2 = keras.layers.BatchNormalization()(conv2)
conv2 = keras.layers.MaxPool1D(pool_size = 5)(conv2)
conv2 = keras.layers.Dropout(0.5)(conv2)

lstm = keras.layers.GRU(64,return_sequences=True)(conv2)
lstm = keras.layers.Flatten()(lstm)
output_layer = keras.layers.Dense(4, activation="softmax")(lstm)
```

Figure 11: 2 CNN layers with a filter size of 64, which is used as a feature extraction and then fed as an input to a GRU network.

Before going to ensemble, let's understand some terminologies we can see in the models.

- **Batch normalization** applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

- **Max Pooling** is a pooling operation that calculates the maximum value for patches of a feature map and uses it to create a down sampled (pooled) feature map. It is usually used after a convolutional layer.
- **Flatten** layer flattens the multi-dimensional input tensors into a single dimension, so you can model your input layer and build your neural network model, then pass that data into every single neuron of the model effectively.

D. Ensemble

Neural network models are a nonlinear method. This means that they can learn complex nonlinear relationships in the data. A downside of this flexibility is that they are sensitive to initial conditions, both in terms of the initial random weights and in terms of the statistical noise in the training dataset.

This stochastic nature of the learning algorithm means that each time a neural network model is trained, it may learn a slightly (or dramatically) different version of the mapping function from inputs to outputs, that in turn will have different performance on the training and holdout datasets.

As such, we can think of a neural network as a method that has a low bias and high variance. Even when trained on large datasets to satisfy the high variance, having any variance in a final model that is intended to be used to make predictions can be frustrating.

A successful approach to reducing the variance of neural network models is to train multiple models instead of a single model and to combine the predictions from these models. This is called ensemble learning and not only reduces the variance of predictions but also can result in predictions that are better than any single model.

Combining the predictions from multiple neural networks adds a bias that in turn counters the variance of a single trained neural network model. The results are predictions that are less sensitive to the specifics of the training data, choice of training scheme, and the serendipity of a single training run. In addition to reducing the variance in the prediction, the ensemble can also result in better predictions than any single best model.

To generate our ensemble model, I chose f1-score to be the criteria for judging predictions from different models. Based on it, a weight is assigned to each model's prediction. Combining all the weights of the models along with their predictions, we get the prediction from the ensemble model.

4. Results

Initially, the first participant data was available, so results were obtained for participant 1 prior to training the model once again for the second participant. Results here are also arranged in the same manner.

4.1 Participant 1

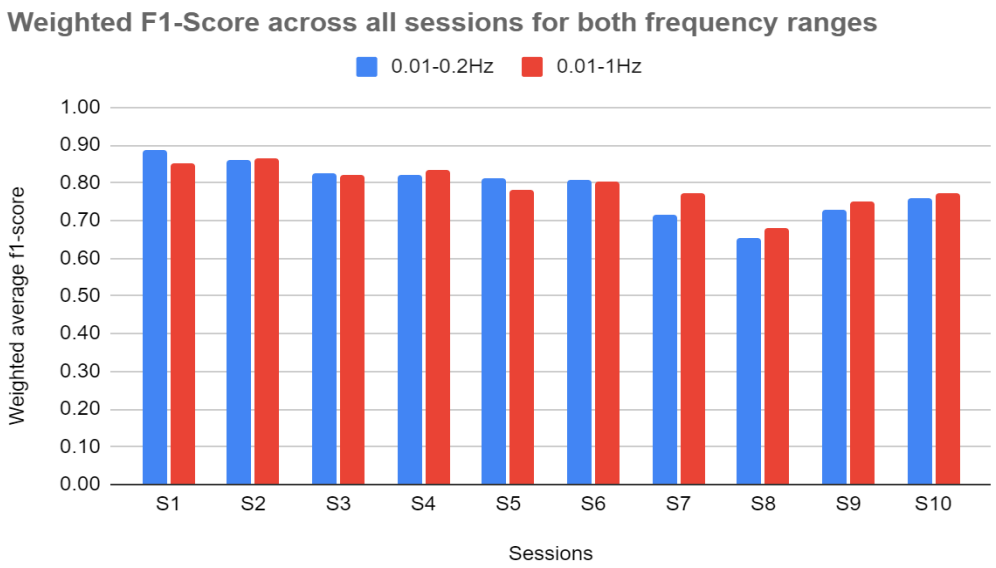
A. Comparison of classification accuracy for data filtered through 0.01-0.2Hz and 0.01-1Hz fourth order Butterworth band-pass filter. Results for the ensemble model are shown which is an average of all 10 testing sessions.

- From the table, we can identify that raw data filtered with 0.01-1Hz frequency band pass filter performs slightly better than the data filtered with 0.01-0.2Hz frequency band pass filter. So, I chose this frequency band for further analysis.

	0.01-0.2Hz			**0.01-1Hz**		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Both Hands	0.61	0.70	0.63	**0.63**	0.67	0.63
Left	0.55	0.66	0.60	**0.57**	**0.67**	0.60
Rest	0.91	0.81	0.86	0.91	**0.83**	**0.87**
Right	0.57	0.72	0.63	**0.58**	0.72	**0.64**
Weighted Average	0.81	0.78	0.79	**0.82**	**0.79**	**0.79**

Table 2: Classification report comparison for data filtered through 0.01-0.2Hz and 0.01-1Hz fourth order Butter-worth filter

- Let's now compare the weighted f1 accuracy for each of the testing sessions for both frequency ranges. The accuracy of the deep learning models varies dramatically with the testing session. This will become even more apparent when we will look at other metrics.



Graph 1: Weighted F1-Score for all sessions for both frequency ranges (Participant 1)

B. Comparison of classification accuracy for data filtered through 0.01-1Hz band-pass filter and 0.01-1Hz with wavelet filter along with same band-pass filter. Results for the ensemble model are shown which is an average of all 10 testing sessions.

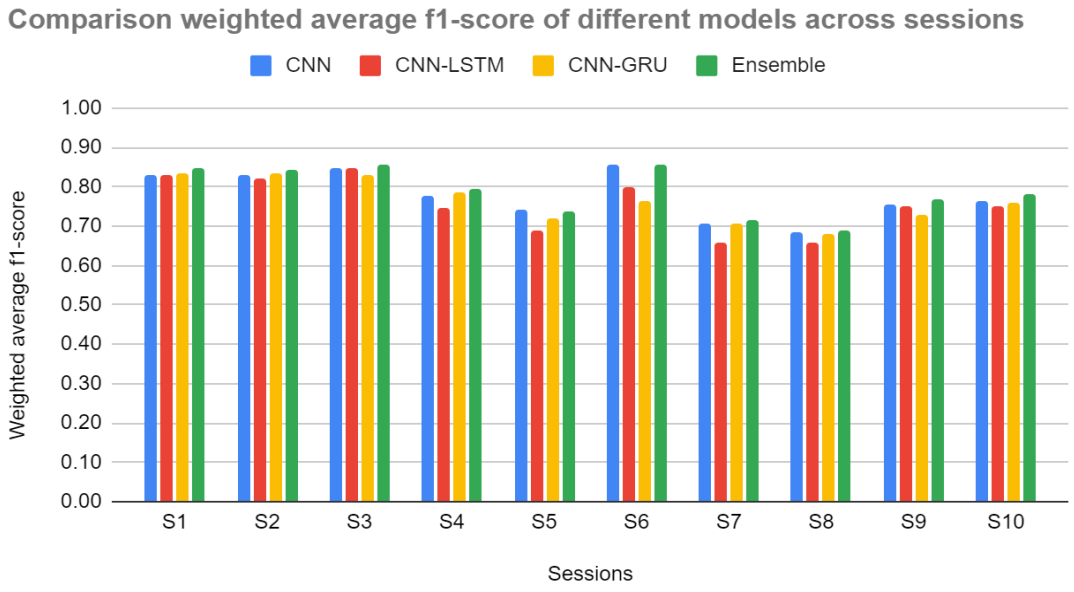
- Now let's have a look at the effect of the use of a wavelet filter on the filtered data for the frequency range 0.1-1Hz. The idea of the wavelet filter came because it is useful in motion correction during data preprocessing, but I didn't observe any significant improvement. The model performed the same for both variations of input data.

	0.01-1Hz			**0.01-1Hz with wavelet filter**		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Both Hands	**0.63**	**0.67**	**0.63**	0.55	0.64	0.58
Left	**0.57**	0.67	0.60	0.56	0.67	0.60
Rest	0.91	0.83	0.87	0.91	0.83	0.87
Right	0.58	**0.72**	**0.64**	0.58	0.70	0.62
Weighted Average	**0.82**	**0.79**	0.79	0.81	0.78	0.79

Table 3: Classification report comparison for data filtered through 0.01-1Hz and 0.01-1Hz with wavelet filter

C. Comparison of weighted f1-score of Conv1D, Conv1D-LSTM, Conv1D-GRU, and Ensemble model for data filtered through 0.01-1Hz for all 10 sessions

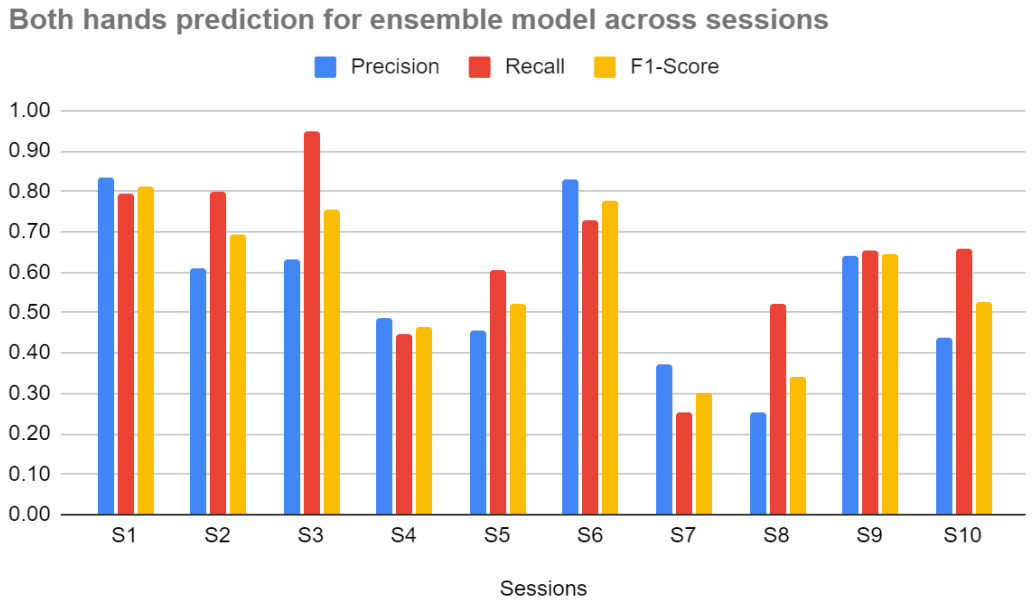
- We can also evaluate the performance of each model; CNN, CNN-LSTM, CNN-GRU, Ensemble. We can clearly observe the benefits of having an ensemble model here. If we compare the independent models, we can see that not a single model can be said to be the best one across all sessions. However, having an ensemble of these models guarantees the best possible outcome for any session. Hence, the ensemble model can be considered more data unbiased.



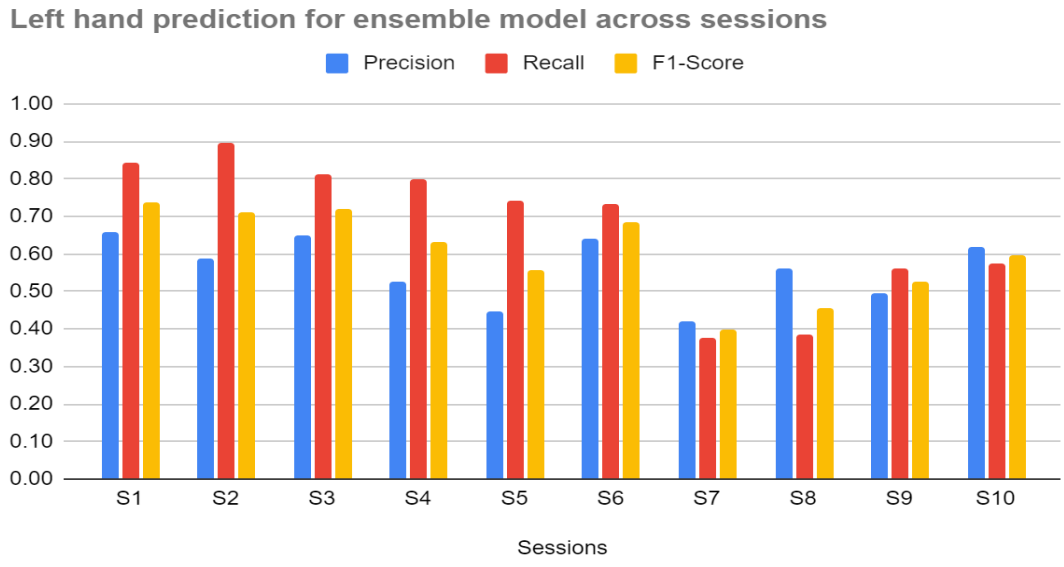
Graph 2: Weighted F1-Score for all sessions for all models (Participant 1)

D. Variation of all task precision, recall, and f1-score of Ensemble model for data filtered through 0.01-1Hz for all 10 sessions

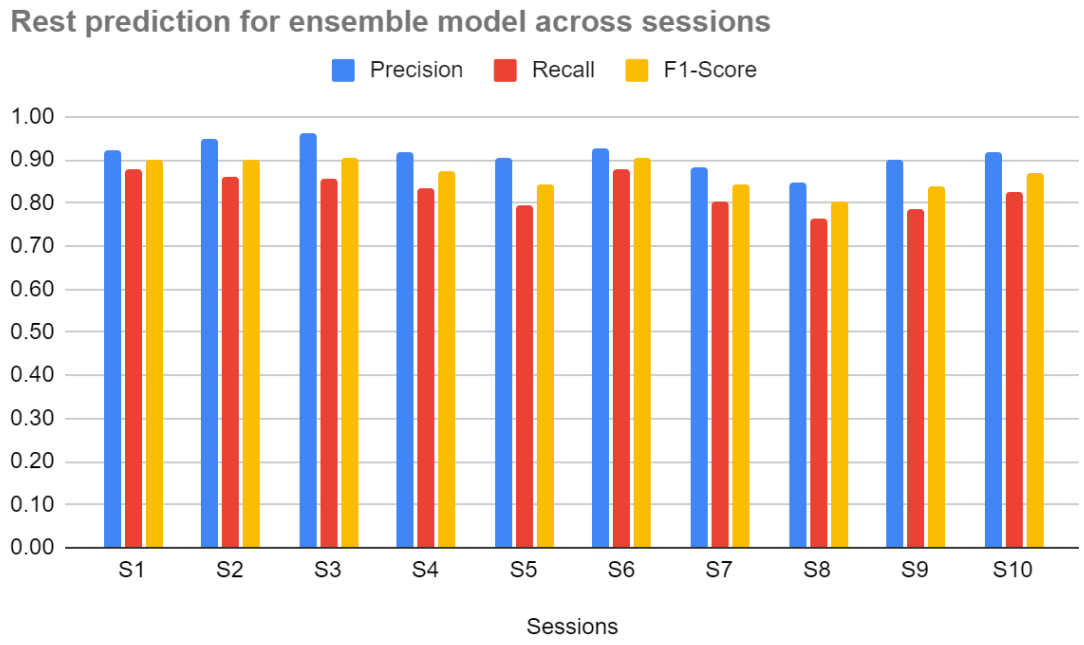
- Now let's look at the precision, recall, and f1-score for the ensemble model for each task (both-hands, left, rest, right) one by one. The graphs presented are for the ensemble model.
- We can see in the graphs that precision, recall, and f1-score for rest data is higher than the other active tasks. This is due to the data imbalance in the dataset. The ratio of rest data to any task data is 15:2.



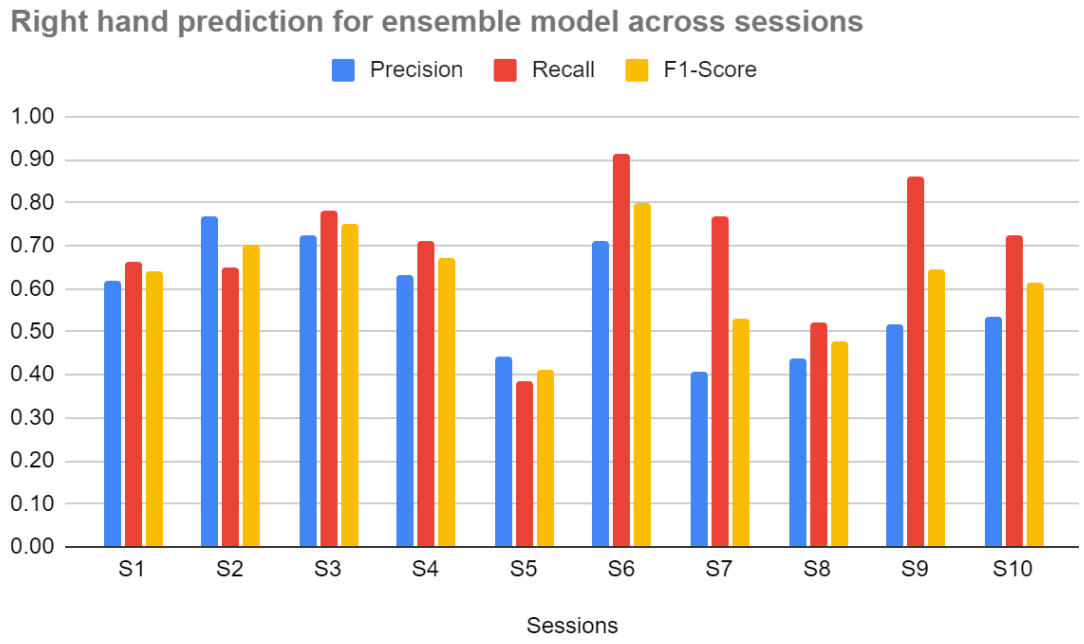
Graph 3: Precision, Recall, and F1-Score for all sessions for frequency band 0.01-1Hz for both hands (Participant 1)



Graph 4: Precision, Recall, and F1-Score for all sessions for frequency band 0.01-1Hz for left hand (Participant 1)



Graph 5: Precision, Recall, and F1-Score for all sessions for frequency band 0.01-1Hz for rest (Participant 1)

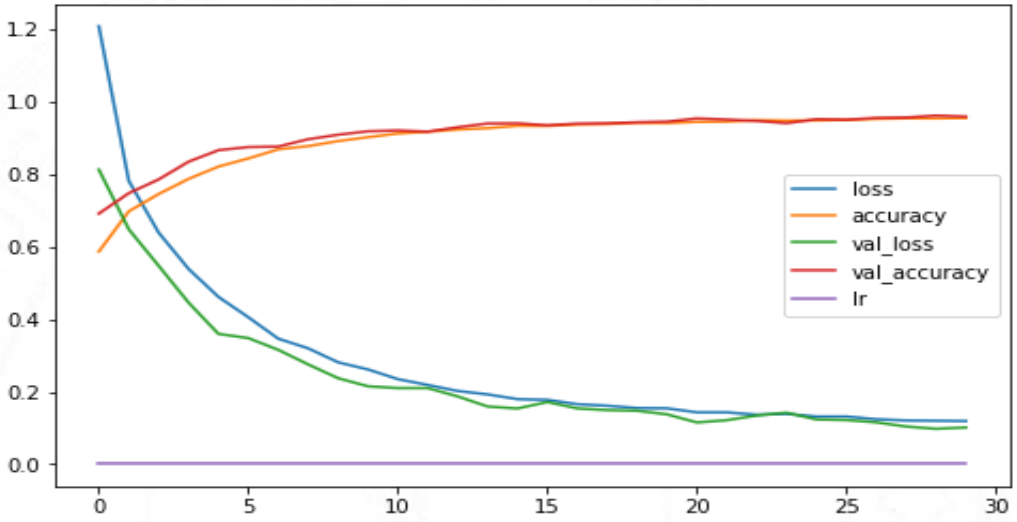


Graph 6: Precision, Recall, and F1-Score for all sessions for frequency band 0.01-1Hz for right hand (Participant 1)

E. Training and Validation losses and accuracy curve

- We can see that the training and validation curves closely fit each other which suggests the model is converging and not overfitting or underfitting the data
- The losses decrease with epochs and the accuracy of the model increases. After 30 epochs, generally we get:

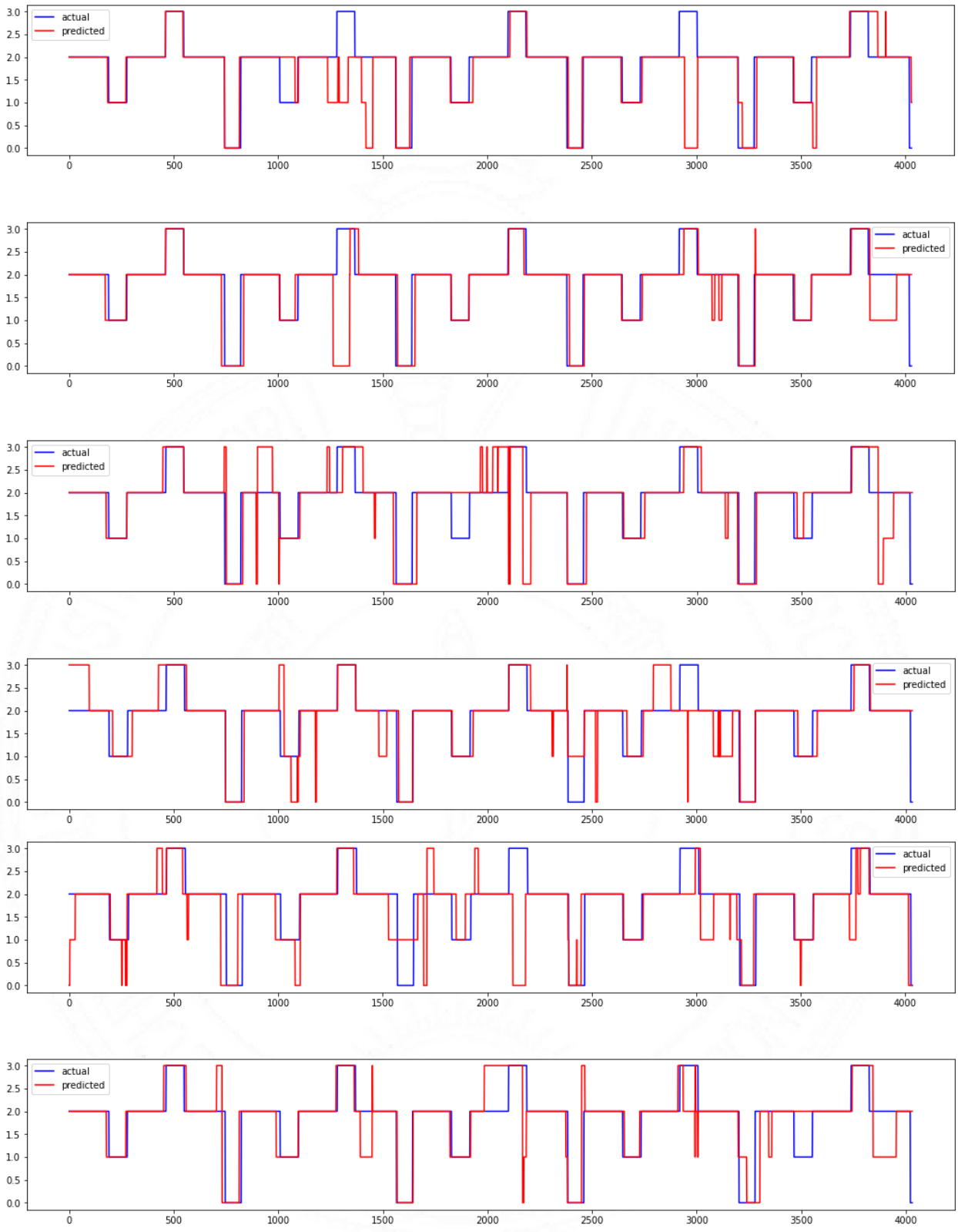
Training accuracy	0.95 – 0.97
Validation accuracy	0.94 – 0.96
Training loss	0.12 – 0.09
Validation loss	0.11 – 0.09

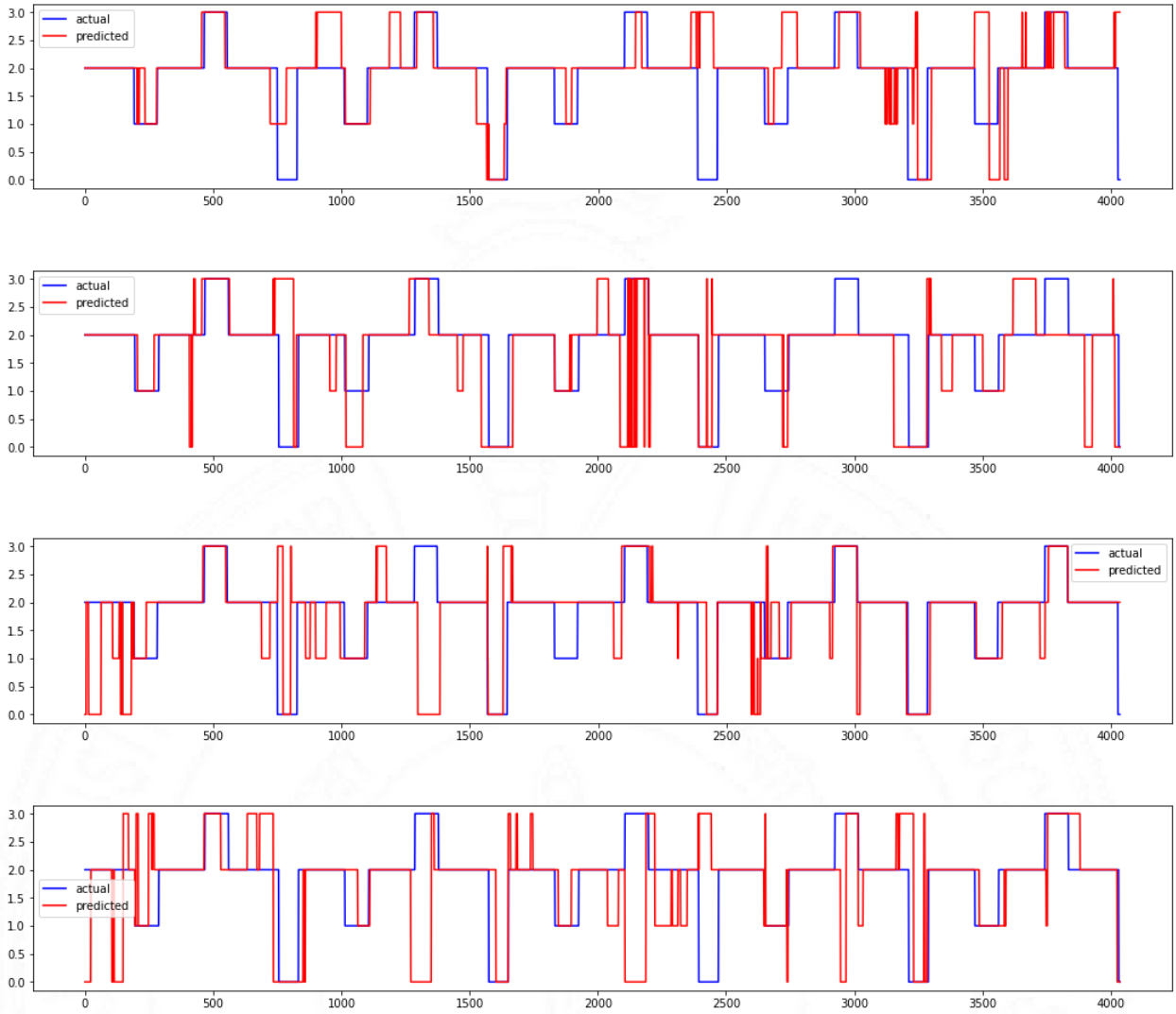


Graph 7: Training loss and accuracy, Validation loss and accuracy, and learning rate (0.025) for Session 1 CNN model

F. Truth vs Prediction line for all testing sessions for ensemble model

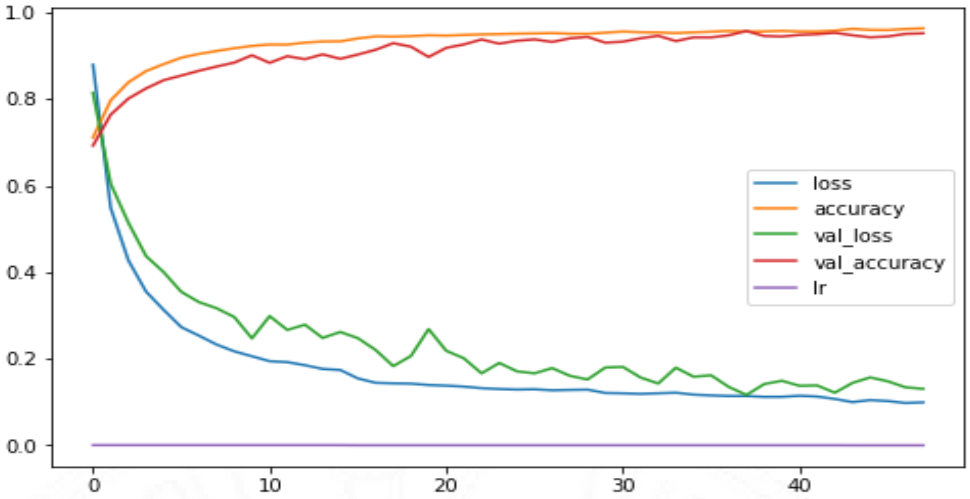
- Perhaps the best way to visualize the results from the experiment is by comparing the actual truth to predicted values by the model. For this I have generated a task label vs samples graph and plotted both the actual labels and predicted labels on top of each other.
- The blue line refers to actual labels and the red line is the predicted labels. The vertical axis values can be 0,1,2, or 3 depending on the task.
  '0' refers to 'both-hands'. '1' refers to 'left hand'. '2' refers to 'rest'. '3' refers to 'right hand'. The graphs are arranged session wise.


actual
predicted
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000
2500
3000
3500
4000



Graph 8: Truth vs Prediction Curve for all sessions (Participant 1)

G. Classification report when 5th experiment in each session used for testing data and rest 4 experiments from all sessions are used for training and validation
    - Number of epochs had to be increased to 50 as more data for training is available, convergence takes more iterations.
    - Validation accuracy of 0.95 and validation loss of 0.1307 was observed for CNN model.



Graph 9: Training loss and accuracy, Validation loss and accuracy, and learning rate (0.025) for CNN model for 5th experiment cycle chosen as testing

	precision	recall	f1-score
both-hands	0.43	0.45	0.44
left	0.46	0.53	0.50
rest	0.88	0.77	0.82
right	0.38	0.68	0.49
accuracy			0.71
macro avg	0.54	0.61	0.56
weighted avg	0.76	0.71	0.73

Table 4: Classification report for ensemble model for 5th experiment cycle chosen as testing

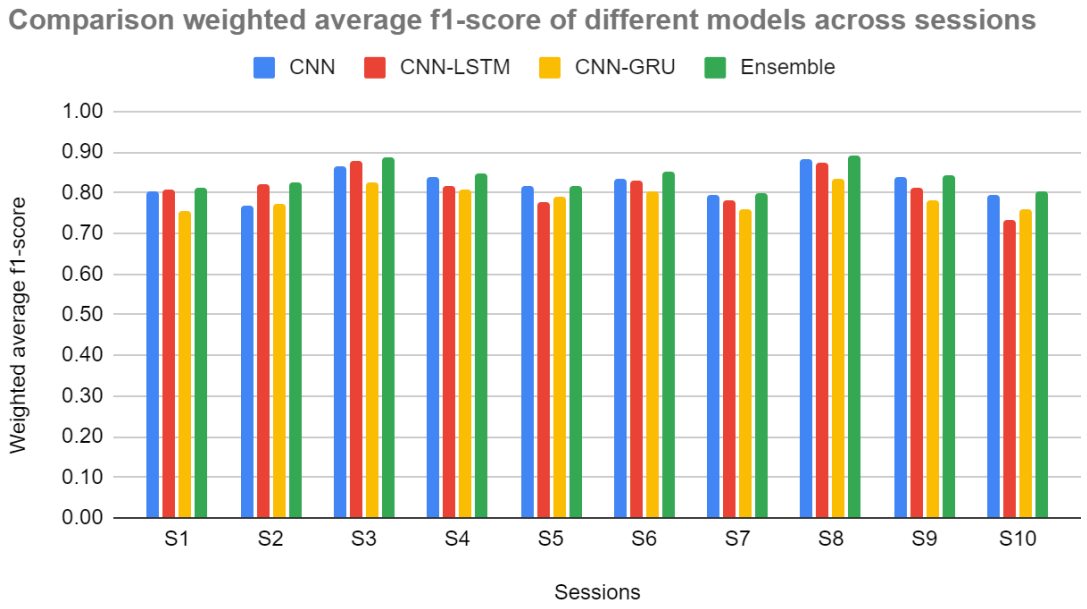
4.2 Participant 2

A. Comparison of classification accuracy for data for Participant 1 and 2. Results for the ensemble model are shown which is an average of all 10 testing sessions.

	Participant 1			Participant 2		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Both Hands	**0.63**	**0.67**	**0.63**	0.56	0.57	0.54
Left	0.57	0.67	0.60	**0.65**	**0.71**	**0.67**
Rest	0.91	0.83	0.87	**0.95**	**0.89**	**0.92**
Right	0.58	0.72	0.64	**0.62**	**0.78**	**0.69**
Weighted Average	0.82	0.79	0.79	**0.85**	**0.83**	**0.84**

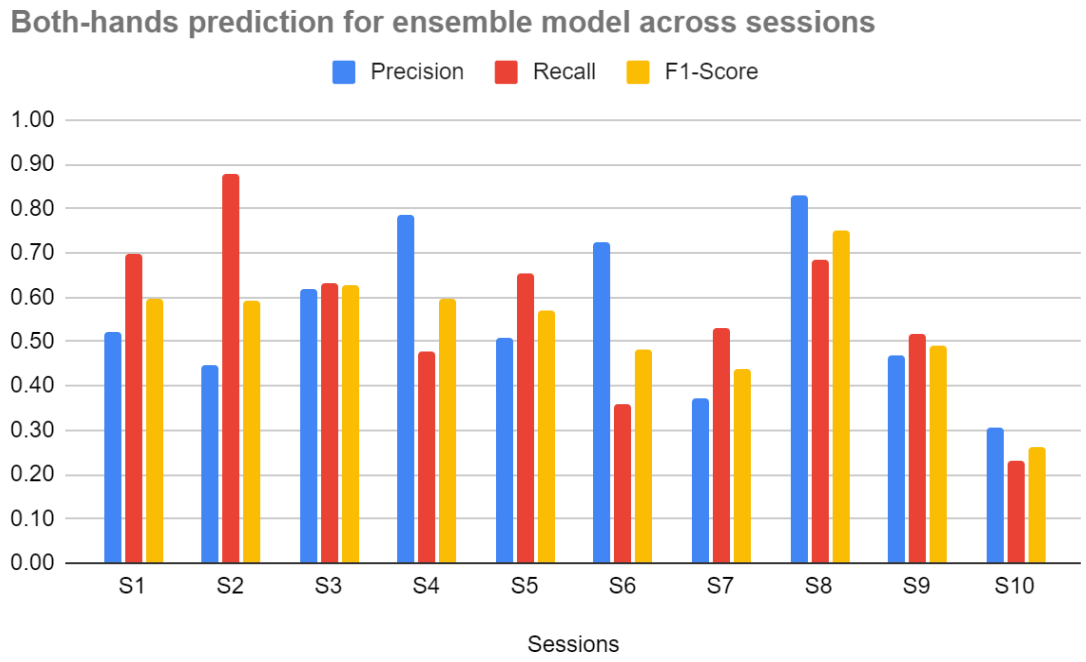
Table 5: Classification report comparison for data from both participants

B. Comparison of weighted f1-score of Conv1D, Conv1D-LSTM, Conv1D-GRU, and Ensemble model for data filtered through 0.01-1Hz for all 10 sessions

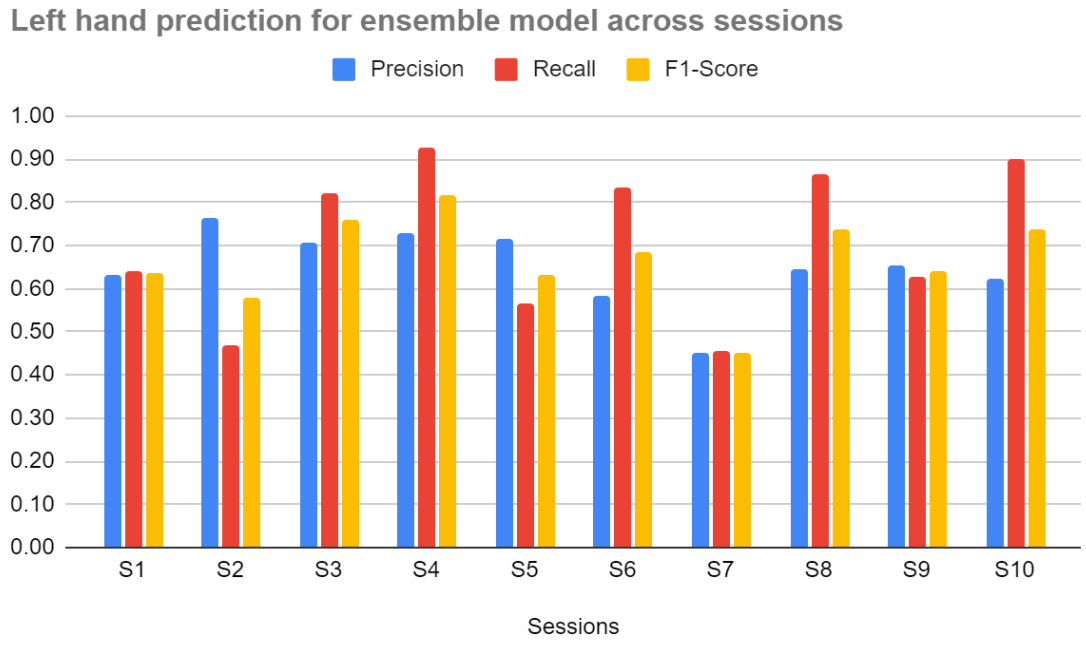


Graph 10: Weighted F1-Score for all sessions for all models (Participant 2)

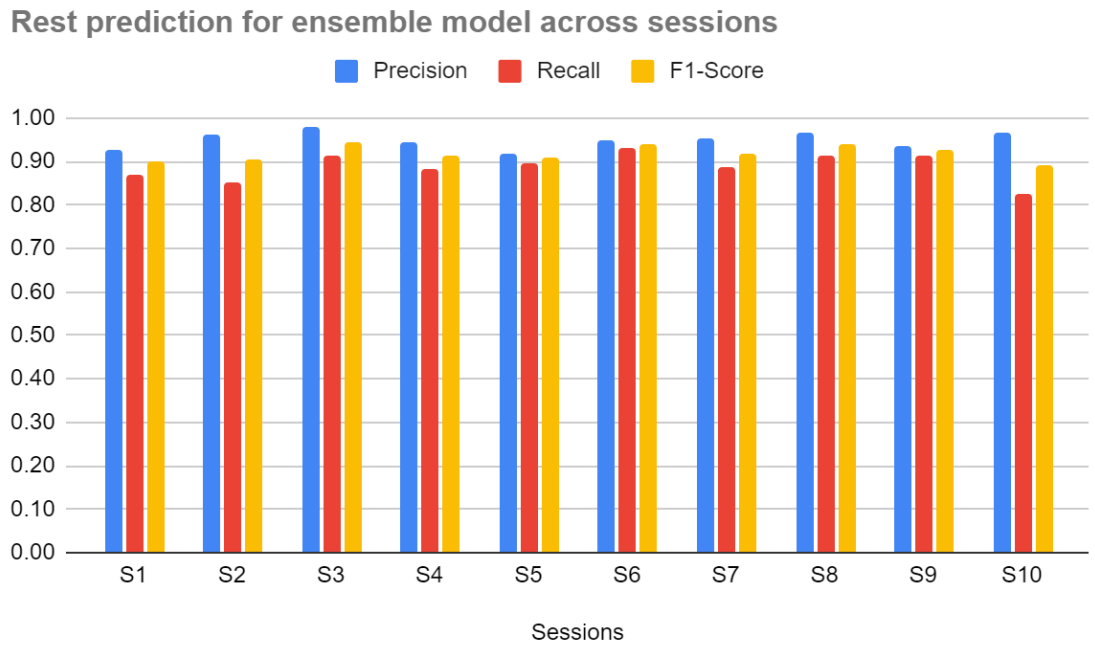
C. Variation of both-hands, left hand, rest, and right-hand task precision, recall, and f1-score of Ensemble model for data filtered through 0.01-1Hz for all 10 sessions



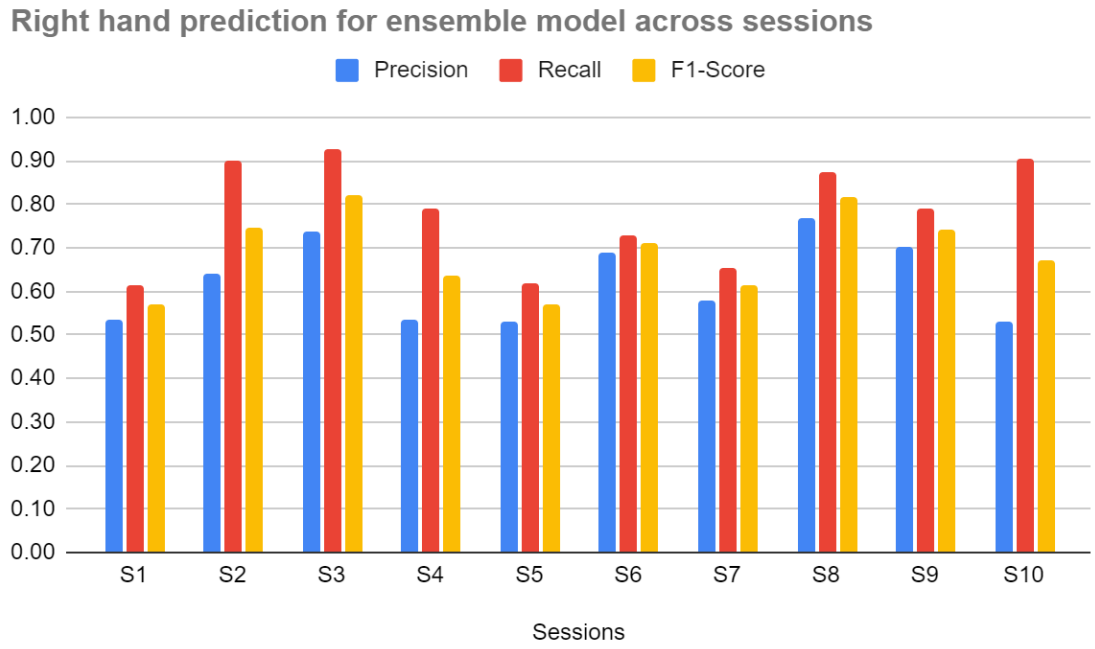
Graph 11: Precision, Recall, and F1-Score for all sessions for frequency band 0.01-1Hz for both hands (Participant 2)



Graph 12: Precision, Recall, and F1-Score for all sessions for frequency band 0.01-1Hz for left hand (Participant 2)



Graph 13: Precision, Recall, and F1-Score for all sessions for frequency band 0.01-1Hz for rest (Participant 2)

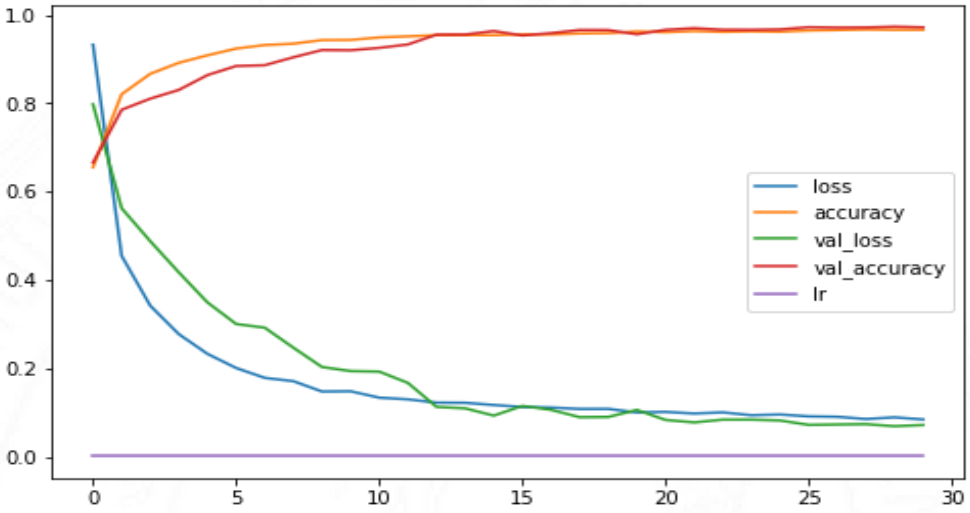


Graph 14: Precision, Recall, and F1-Score for all sessions for frequency band 0.01-1Hz for right hand (Participant 2)

D. Training and Validation losses and accuracy curve

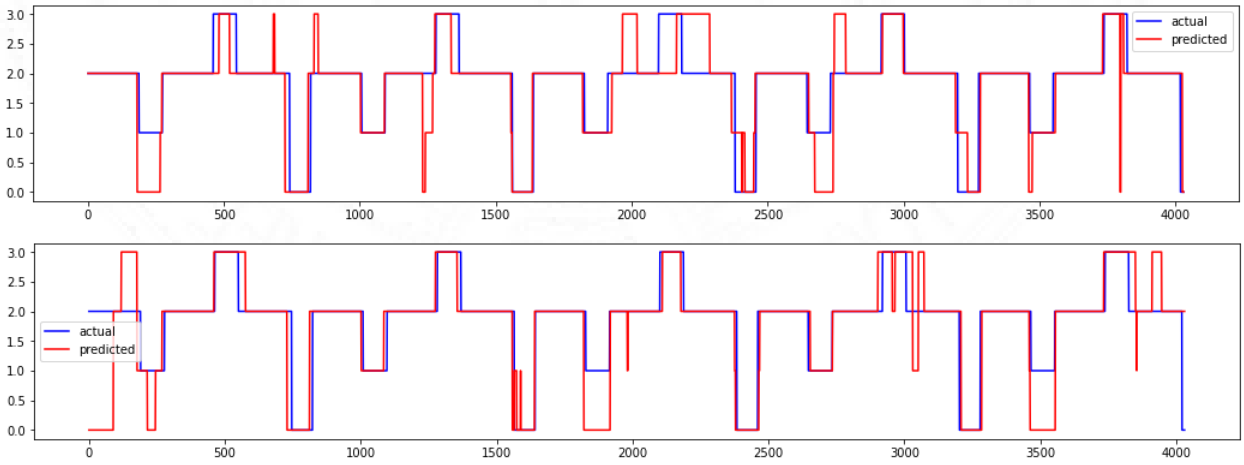
- The losses decrease with epochs and the accuracy of the model increases. After 30 epochs, generally we get:

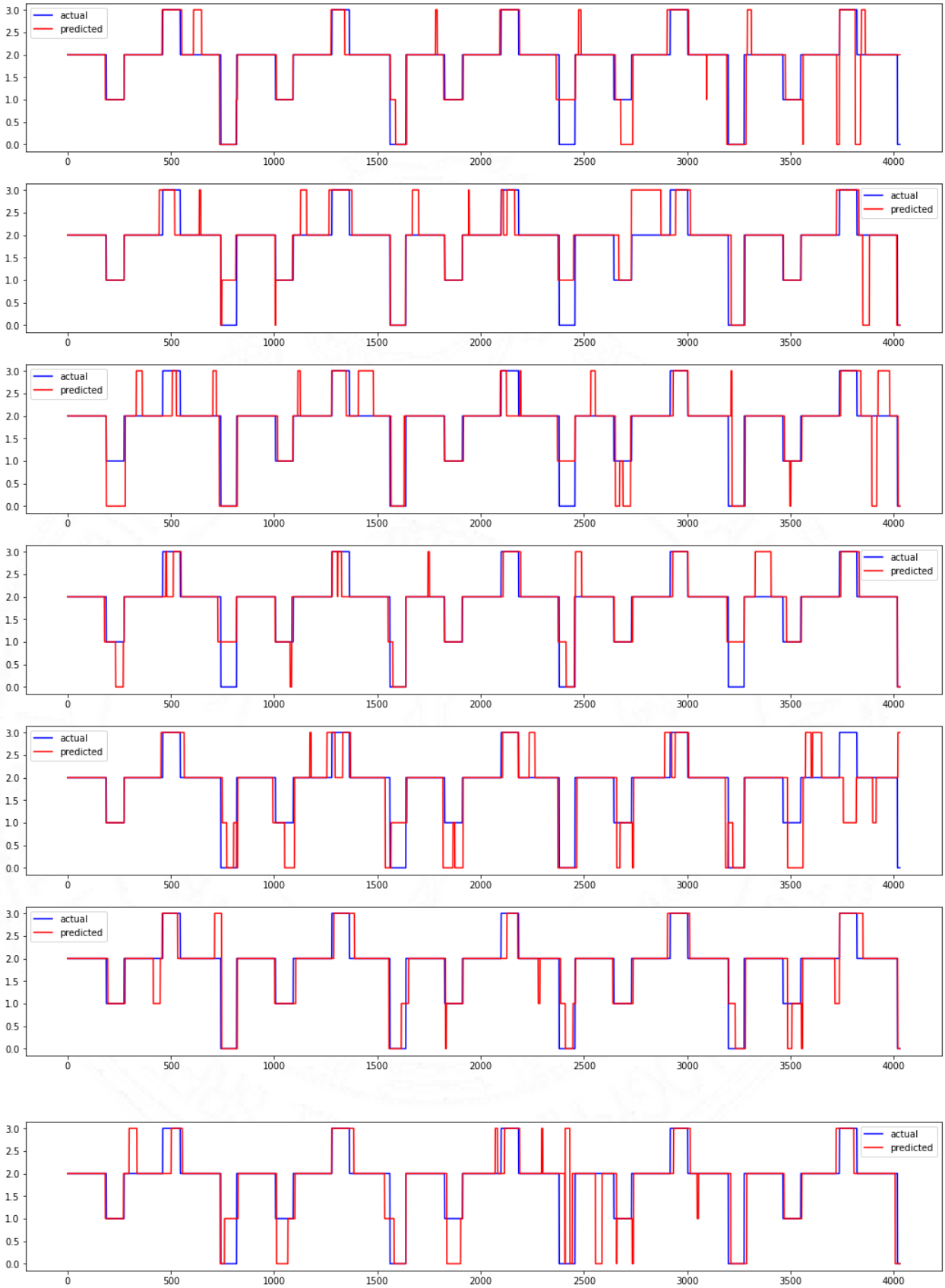
Training accuracy	0.95 – 0.97
Validation accuracy	0.94 – 0.96
Training loss	0.12 – 0.09
Validation loss	0.11 – 0.09

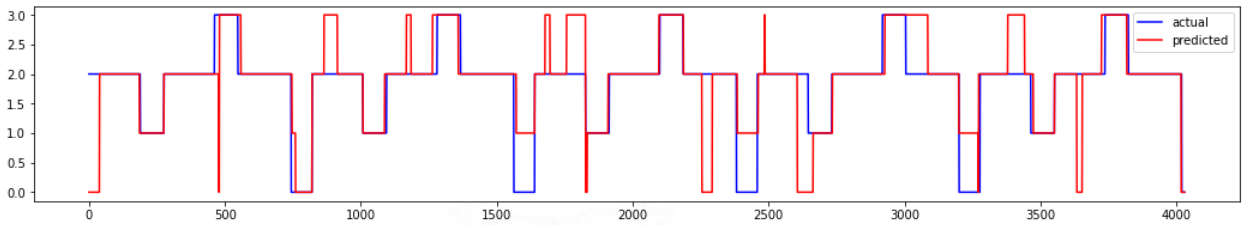


Graph 15: Training loss and accuracy, Validation loss and accuracy, and learning rate (0.025) for Session 1 CNN model

E. Truth vs Prediction line for all testing sessions for ensemble model (Session 1 to 10)




actual
predicted
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000
2500
3000
3500
4000



Graph 16: Truth vs Prediction Curve for all sessions (Participant 2)

In summary:

- The models can converge with a validation accuracy of 96% and testing macro average f1-score of 79% and 84% for participant 1 and 2 respectively. So, there is a drop in accuracy between validation set and new testing session.
- The ensemble model outperforms any individual model and hence is a valuable resource as it considers the stochastic nature of individual deep learning models.
- The truth vs prediction curves clearly indicate that some tasks durations are being predicted with almost perfect accuracy, in some cases model predicts a different task than the actual task. So, it can be said that for certain time of data stream, the model can be extremely accurate.
- The average f1-score for tasks were in the range of 60-70% for both participants.

5. Conclusion

During the work, it became evident that deep learning models can help in f-NIRS classification problems. While the experimental setup was designed for a classification of motor execution task, we can modify the setup for motor imagery tasks as well and use the same model for comparison. I mention motor imagery, which is classification based on thinking itself rather than actions, because there are huge applications of this technology in neurorehabilitation.

I used CNNs and RNNs as the basis of our deep learning models which are hugely sought-after time series data modelling and prediction. In our case, CNNs outperformed RNNs in classifying the data.

For participant 1, raw data was already available from which data preprocessing and classification were done. For participant 2, I also worked on obtaining the raw data as well where we used a dark room setup for getting data from f-NIRS setup.

Future Work

The aim of the project is to develop a real time f-NIRS BCI system. Towards this goal, I hope this thesis helps in overall completion of the project. There are some works that I believe should follow on the thesis:

- If we look back to our figure 1, which shows a basic flowchart level design of a real time BCI system, we must start working on developing an application interface through which we can access the data from the f-NIRS device and predict using our deep learning model in real time. From my initial research, I have found out that the manufacturer has a software that can be purchased and used along with their hardware for live data steaming for BCI applications. Another way is to create the application yourself and use an LSL (Lab Streaming Layer) for live data transfer.

- As of now, since we are getting an overall macro average f1-score in the range of 60-70% for tasks, we need to look for more possibilities to increase accuracy. To this extent, there are some recommendations for the next person working on the project:

    - The data is imbalanced, with any task to rest ratio being 2:15. We may have to acquire more session data than 10 to increase the overall accuracy.
    - We can also investigate GANs (General Adversarial Networks) which help in synthetic data generation. It may help in balancing the data. I have used basic data balancing techniques like under sampling, over sampling, but they don't help.
    - There are some other cutting edge deep learning models as well which can be tried out. Two of them are TCNs (Temporal Convolutional Networks) and Transformers.
    - We can also get into deep adversarial learning techniques, where two versions of same model compete for some reward function.

- One significant reason for lower accuracy can also be related to f-NIRS device. It is very difficult to have good channels in the data stream during data acquisition and the number of channels is also limited to 20 for each wavelength. In fact, data were taken from the participants with no hair on/ shaved head as the device didn't work on people with hair.

6. References

6.1 Research Papers

- Ma T, Chen W, Li X, Xia Y, Zhu X, He S. fNIRS Signal Classification Based on Deep Learning in Rock-Paper-Scissors Imagery Task. Applied Sciences. 2021; 11(11):4922. doi:10.3390/app11114922
- Yücel MA, Lühmann AV, Scholkmann F, et al. Best practices for fNIRS publications [published correction appears in Neurophotonics. 2021 Jan;8(1):019802]. Neurophotonics. 2021;8(1):012101. doi:10.1117/1.NPh.8.1.012101
- Pinti P, Scholkmann F, Hamilton A, Burgess P, Tachtsidis I. Current Status and Issues Regarding Pre-processing of fNIRS Neuroimaging Data: An Investigation of Diverse Signal Filtering Methods Within a General Linear Model Framework. Front Hum Neurosci. 2019;12:505. Published 2019 Jan 11. doi:10.3389/fnhum.2018.00505
- Klein F, Kranczioch C. Signal Processing in fNIRS: A Case for the Removal of Systemic Activity for Single Trial Data. Front Hum Neurosci. 2019;13:331. Published 2019 Sep 24. doi:10.3389/fnhum.2019.00331
- N. M. Sommer, B. Kakillioglu, T. Grant, S. Velipasalar and L. Hirshfield, "Classification of fNIRS Finger Tapping Data With Multi-Labeling and Deep Learning," in *IEEE Sensors Journal*, vol. 21, no. 21, pp. 24558-24569, 1 Nov.1, 2021, doi: 10.1109/JSEN.2021.3115405.
- Hamid H, Naseer N, Nazeer H, Khan MJ, Khan RA, Shahbaz Khan U. Analyzing Classification Performance of fNIRS-BCI for Gait Rehabilitation Using Deep Neural Networks. *Sensors*. 2022; 22(5):1932. https://doi.org/10.3390/s22051932

6.2 Reference material from internet

- https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-activity-recognition-time-series-classification/
- https://omdena.com/blog/time-series-classification-model-tutorial/
- https://www.kaggle.com/code/ruslankl/eeg-data-analysis/notebook
- https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/
- https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9
- https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec
- https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/