Non-Invasive Motor Imagery Classification

Gokulan Nithianandam

A dissertation submitted to the University of Bath in partial fulfilment of the requirements for the award of MSc in
Robotics and Autonomous Systems

Department of Electronic & Electrical Engineering
University of Bath

September 2022

Declaration

This dissertation is submitted to The University of Bath in accordance with the requirements of the degree of Master of Science in Robotics and Autonomous Systems, in the Department of Electrical and Electronic Engineering. No portion of the work in this document has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed 

Date 14/09/2022

Abstract

This study compares four deep learning models and two data augmentation methods for the classification of Motor Imagery tasks. ShallowConvNet, EEGNet, and DeepConvNet were chosen as deep learning models. In this work, the Data Augmentation and Transfer Learning method is used to address the issue of data scarcity. To augment the existing data, noise addition and a sliding window are used. For the noise addition method, a mean of zero and different standard deviation values are trailed to determine whether the distinctive or similar samples generated from the original data improve classification accuracy. To find the best sliding window method for augmenting Motor Imagery EEG data, augmented data with overlapping windows and non-overlapping windows are tested separately. Transfer learning is implemented using ResNet50, which has been pre-trained on ImageNet. The results of this study demonstrated that augmented data with noise addition significantly improved classification accuracy across all deep learning models.

Table of Contents

Acknowledgements

The authors would like to thank Dr Dingguo Zang, University of Bath for the support extended to this work. Also, thanks to Xin Gao for allowing me to use his unpublished code as reference for epoch the EEG data.

List of abbreviations

BCI	Brain Computer Interface
CNN	Convolutional Neural Network
CCV	Cross Covariance
CSP	Common Spatial Pattern
ECoG	Electrocorticogram
EEG	Electroencephalogram
FBCSP	Filter Bank Common Spatial Pattern
fMRI	functional Magnetic Resonance Imaging
fNIRS	functional Near-Infrared Spectroscopy
FEIS	Fourteen-channel EEG for Imagined Speech
GAN	Generative Adversarial Networks
ICA	Independent Component Analysis
ICE	Electroencephalography
MPC	Mean phase coherence
MSC	Magnitude-Squared Coherence
ResNet	Residual neural network
VGGNet	Visual Geometry Group neural network

List of figures

List of tables

1-Introduction

A practitioner inspecting a physiological signal discovers a pattern and makes a decision. However, as the signal quantity increases, this process becomes more difficult and time-consuming. Furthermore, the practitioner's mood and tiredness influence the quality of signal analysis (Aggarwal, S. and Chugh, N., 2019). To compensate for the limitations of manual analysis, a computer-based signal processing system known as Brain Computer Interface (BCI) is used to interpret physiological signals by connecting a person's brain to a computer system. (Aggarwal, S. and Chugh, N., 2019). BCI systems collect electrical signals related to brain activity using electroencephalography (EEG), functional magnetic resonance imaging (fMRI), electrocorticography (ECoG), functional near-infrared spectroscopy (fNIRS), and electroencephalography (ICE) (Panachakel and Ramakrishnan, 2021). Only EEG-based BCI systems are considered in this project. Mu rhythm, event related P300, slow cortical potential, and steady state visual evoked potential are the types of brain activities used to distinguish BCI systems-based EEG, and the mu rhythm is specific to motor imagery task. (Aggarwal, S. and Chugh, N., 2019).

Motor imagery is a cognitive process that involves imagining body parts moving without actually moving them. Decoding imagined motor movements during the process of motor imagery entails identifying motor movements from the acquired brain signal. This procedure allows people who are in a locked state to interact with the outside world by controlling external devices such as prostheses, wheelchairs, and other devices (Aggarwal, S. and Chugh, N., 2019). Deep learning models have demonstrated the ability to automatically encode a hierarchy of characteristics and adapt to the internal structure of the data when compared to the machine learning approach to classifying motor imagery tasks (Ullah et al., 2018; Nithianandam, G.,2022). Furthermore, Deep Learning extracted features have been shown to be more discriminative and robust than hand-extracted features (Ullah et al., 2018 Nithianandam, G.,2022). The classification accuracy is satisfactory due to the discriminative nature of the extracted features. In this project, we investigate Deep Learning architectures to classify motor imagery tasks for the reasons stated above. Because of the scarcity of EEG data, the use of deep learning models in the field of motor imagery has been slower. The primary causes of EEG data scarcity are the high cost of EEG acquisition devices, the restriction of publicly available datasets due to participant privacy concerns, and the short duration of participant recording time.

The data augmentation technique, which augments new data from existing data, is widely used in the field of BCI to address the issue of data scarcity and has been shown to improve the classification accuracy of deep learning models (Lashgari et al, 2020 Nithianandam, G.,2022). Transfer learning is another emerging technique for dealing with data scarcity. Because of prior knowledge acquired or learned in the related task, transfer learning is capable of learning features with little data. Furthermore, transfer learning addresses the issue of individual variation, which means that the characteristics of EEG data vary greatly depending on an individual's age and mentality. As a result, the EEG signal for the same event may differ from person to person. According to the facts stated above, in this project, we use data augmentation techniques and transfer learning techniques to address the problem of data scarcity in the field of BCI and improve classification accuracy of motor imagery tasks. The structure of the report is as follows: The background is covered in Section 2. Section 3 is dedicated to project execution. Section 4 is devoted to the publication of results. Section 5 is devoted to discussing the results. Section 6 is conclusion. Section 7 is devoted to future works discussion.

1.1 Aim and Objectives

The project's major goal is finding the optimal combination of approaches to decode motor imagery task with less data and achieving a higher classification accuracy by performing a comparison study between combination of Deep Learning models and data augmentation methods. The objectives can be broken down into

- Selection of pre-existing EEG clinical dataset on imagined speech.
- Researching different EEG data pre-processing techniques.
- Investigating various data augmentation methods to find the most effective approach for improving classification accuracy.
- Investigating various pretrained Convolutional Neural Network (CNN) for transfer learning implementation.
- Examining various deep learning models and determining the best deep learning model.
- Pre-process the EEG data and epoch it. Organize the data into the appropriate classes.
- Train and validate deep learning models, including pre-trained CNN, on the training and validating data with and without data augmentation.
- Test the deep learning models, including the pre-trained CNN, on the test data with and without data augmentation.
- Determine the best deep learning model and data augmentation method for motor imagery task classification using the obtained classification accuracy.

2-Background

2.1-Artefacts in EEG

EEG signal displays electrical activity variation as recorded by electrodes on the scalp. The EEG signals are influenced by a number of sources that are unrelated to brain activity; these sources, known as artefacts, alter the signal's originality. The artefacts in the EEG signal can be classified into two categories such as physiological and technical (Frølich et al., 2015). Sources contributing to the two categories of artefacts are explained in the table 1. It is challenging to remove physiological artefacts from the EEG signals because it is difficult to ignore the subject's eyes blinking throughout the EEG recording, for instance. Contrarily, all technological artefacts—aside from those brought on by the electric grid be removed with the right experimental setup. The artefacts occurring at the power grid's working frequency, which is a known frequency, make it easier to filter out interference from the power grid. The two most frequent operating frequencies for electrical grids are 50Hz and 60Hz. Because the notch filter arrangement offers a smaller stop band and won't obliterate the information in the nearby frequency band, it is used to remove the power grid frequency (Frølich et al., 2015).

Table 1 EEG signal artefact's categories and sources

Artefact's category	Sources
Physiological	Eye movement Muscle contraction Heartbeat
Technical	Loose electrodes Power grid

Al-Saegh, Dawwd and Abdul-Jabbar. (2021) reviewed 40 research papers based on EEG based motor imagery. About 5 out of the 40 publications eliminated artefacts from the EEG signal, and those 5 papers did it using Independent Component Analysis (ICA). Choo et al. (2020) stated the uniqueness of the EEG signal is harmed when artefacts in the signals are removed, and classification accuracy suffers as a result.

Schirrmeister et al. (2017) employed a third-order Butterworth filter to filter data above 4Hz to eliminate eye movement artefacts. In addition, Schirrmeister et al. (2017) stated the deep learning models can learn the transformation on their own with the minimal pre-processing.

Altuwaijri and Muhammad. (2022) did not remove artefacts from the EEG signal because the process of removing artefacts did not have significant effect on the classification accuracy. Additionally, Schirrmeister et al. (2017) had an accuracy rate of 74% versus 81% for Altuwaijri

and Muhammad (2022). Both authors used the same dataset. Since Altuwaijri and Muhammad (2022) had higher classification accuracy despite the fact that the authors did not remove the artefacts, it would appear that the removal of artefacts had no effect on the classification accuracy. Having established the removal of artifacts had no effects in the classification accuracy, we intended not to remove artifacts for this project. In the next section, we discuss about the frequency band used in the Motor Imagery task.

2.2-Frequency bands

The EEG signal's amplitude and frequency are crucial for differentiating between different physiological activities (Al-Saegh, Dawwd and Abdul-Jabbar., 2021). The frequency range in the EEG signal can be divided into various bands as shown in the table 2.

Table 2 Frequency band and range. The table information is adapted from Bitbrain. (2020).

Frequency band name	Frequency range (Hz)
Delta	0.5-4
Alpha	8-12
Beta	12-35
Gamma	32-100

Al-Saegh, Dawwd, and Abdul-Jabbar examined (2021) 40 papers based on EEG Motor imagery. As illustrated in Figure 1, majority of the reviewed paper used the minimum frequency range between 6-8Hz, and the maximum frequency range was between 38-40Hz.

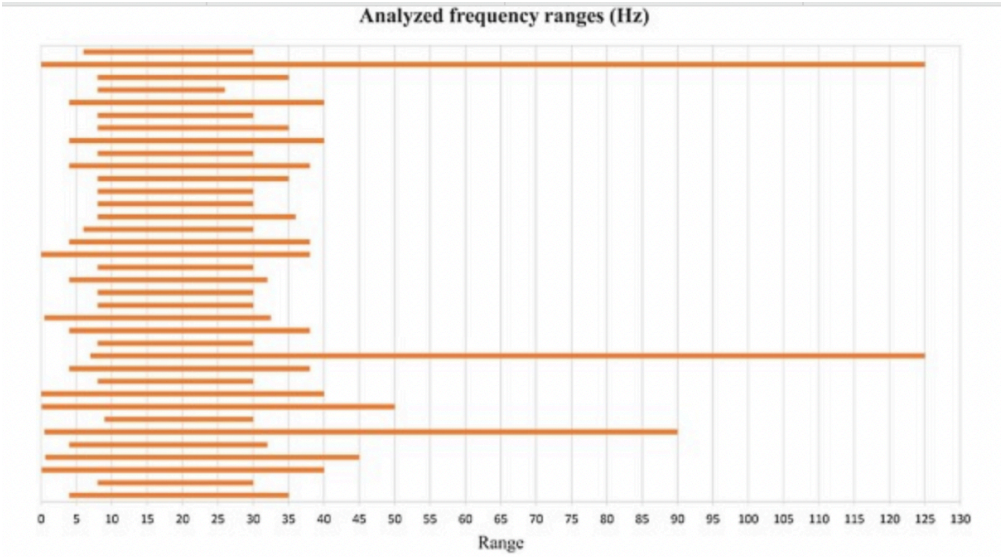

Figure 1 Frequency range used by 40 research papers as reviewed by Al-Saegh, Dawwd and Abdul-Jabbar. (2021)

The power spectrum in the frequency band between 6 and 13 Hz declines when motor imagery takes place. The phenomenon is called event related desynchronization (ERD) (Dai et al., 2019). Furthermore, when motor imagery is occurring, the power spectrum in the 13–30 Hz band increases; this is a phenomenon known as event-related synchronisation (ERS) (Dai et al., 2019). The process of motor imagination is accompanied by ERS and ERD (Dai et al., 2019). Liu et al. (2017) employed a Butterworth band pass filter to filter the frequency range between 8Hz and 30Hz while recording motor imagery at a sampling rate of 256Hz. As represented in figure 2, the brain activity was higher in the frequency range (8-30Hz) during the motor imagery task across all the 14 channels.

According to the aforementioned studies, we consider the frequency range between 4-40Hz is appropriate for this project. In the next section, we discuss about the channel selection.

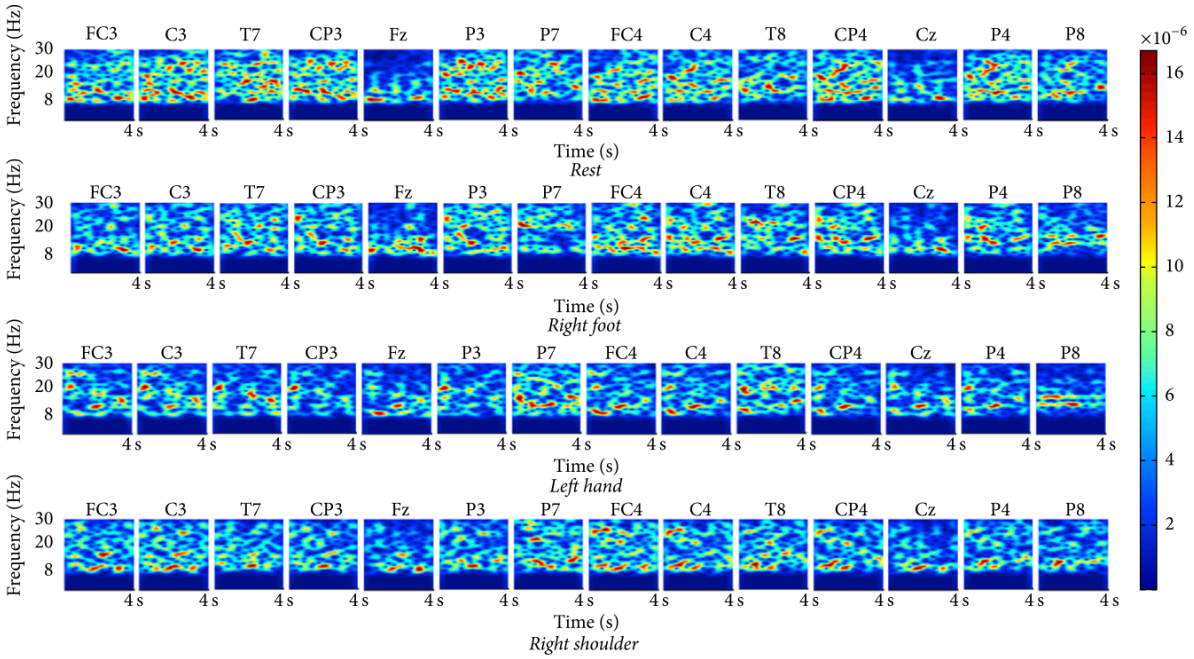

Figure 2 Spectrum map of 14 channel during rest state and imagination of Right foot, Left hand and Right shoulder. Imagen taken from Liu et al. (2017).

2.3- Channel Selection

For the classification of motor imagery tasks, all the channels were used in the research papers (Choo et al.,2020; Schirrmeister et al.,2017 and Altuwaijri and Muhammad.,2022) and achieved mean classification accuracy above 75%. However, the research papers (Choo et al.,2020; Schirrmeister et al.,2017 and Altuwaijri and Muhammad.,2022) did not specify the reason for using all the channels.

Research by Liu et al. (2017) revealed the motor imagination activates multiple cortical regions simultaneously as shown in the figure 3. The black dots on the topographic map (figure 3)

represent the electrodes positions. During the motor imagery, multiple cortical areas are active simultaneously. It appears that elimination of certain channel will result in failure to capture necessary information for the classification of motor imagery. For the above reason, we consider using all the channels in this project. In the next section, we discuss about the deep learning models.

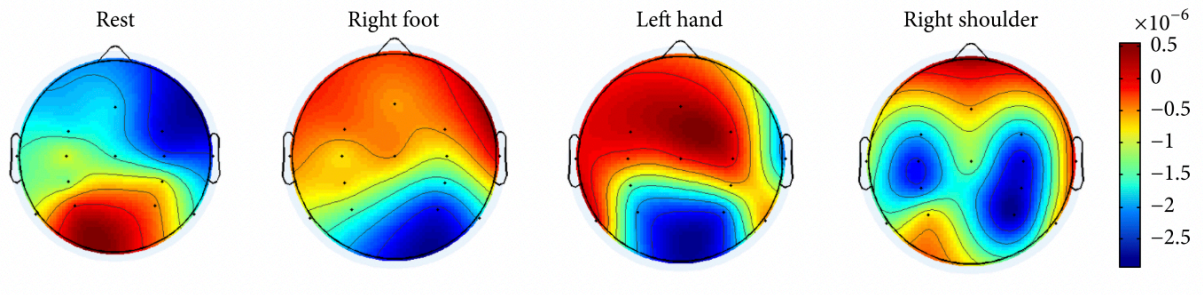

Figure 3 Brain topographic map during rest, imagination of right foot, left hand and right shoulder. Image taken from Liu et al. (2017).

2.4-Deep learning model

EEG signal classification using traditional machine learning methods requires manually extracting feature data and performing specific tasks like the Wavelet transform (WT) and fast Fourier transform (FFT) to produce a satisfactory classification accuracy (Choo et al.,2020). More complex and robust features are required to improve classification accuracy, and the procedure also requires a lot of manual effort, as mentioned by Choo et al. (2020). Unlike typical machine learning methods, deep learning models learned the complex and robust features from the EEG signals on their own, eliminating the requirement for manual feature extraction (Choo et al.,2020; Al-Saegh, Dawwd and Abdul-Jabbar.,2021). Additionally, deep learning algorithms can learn the ideal parameters from a raw EEG data without extensive pre-processing (Al-Saegh, Dawwd and Abdul-Jabbar.,2021). Moreover, in a single pipeline, deep learning models carry out feature extraction, selection, and classification. (Al-Saegh, Dawwd and Abdul-Jabbar.,2021). To prove the aforementioned statement, we use the review study by Nithianandam, G. (2022) where T Nguyen et al. (2017) and Panachakel and Ganesan (2021) used the Arizona State University dataset to characterise EEG signals. Former used a state vector machine and was able to attain a classification accuracy of 49%. Later used Deep Learning architecture and classified EEG signals with an accuracy of 79.7%. We think that Deep Learning architectures can achieve higher classification accuracy than conventional Machine Learning architectures based on the aforementioned results. We examine the ideal Deep Learning architecture since it has been determined that Deep Learning is a viable option.

As reviewed by Al-Saegh, Dawwd and Abdul-Jabbar. (2021), the predominant deep learning architectures are convolutional neural network (CNN), recurrent neural network (RNN) and hybrid CNN (h-CNN). Out of 40 research papers, Al-Saegh, Dawwd, and Abdul-Jabbar (2021) examined them in MI-BCI, 73% of architectures employed CNN, 13% h-CNN, and 14% alternative architectures. In BCI, Lashgari et al. (2020) analysed 60 research papers; 62% of them employed CNN, 16% h-CNN, and 22% alternative architectures. Additionally, CNN has capabilities for accurately representing spatial and temporal features (Choo et al.,2020). CNN based ShallowConvNet and DeepConvNet were created by Schirrmeister et al. (2017) to learn the spatial and temporal features of the raw EEG input. Filter Bank Common Spatial Pattern (FBCSP) served as the foundation for the creation of ShallowConvNet. The detail on FBCSP architecture is available in Ang et al. (2008). One convolution layer is used by the ShallowConvNet to extract temporal information, and another layer is used to extract spatial characteristics. Finally, the collected temporal and spatial information are used for classification using the softmax layer. Two convolution layers make up DeepConvNet, which is followed by additional aggregated layers. While ShallowConvNet has a shallow architecture, DeepConvNet has a deeper architecture. ShallowConvNet performs well, but DeepConvNet performs worse with lesser datasets (Choo et al, 2021). The problem of Overfitting arises with ShallowConvNet due to smaller datasets (Choo et al, 2021). Lawhern et al. (2018) introduced EEGNet, another compact CNN design. EEGNet employs separable and depthwise convolution layers which conducts efficient spatial filtering and filter-bank creation to collect features from the raw EEG data. The main benefit of EEGNet is that there are less training parameters. Choo et al. (2020) used ShallowConvNet, DeepConvNet and EEGNet for the classification of motor imagery task and achieved a satisfactory result and stated that advancement in these three deep learning models allowed for better classification accuracy. As a result, we believe that using these three deep learning architectures will enable us to classify the motor imagery task more accurately.

Even though these deep learning models are superior, they lack consistent performance due to data scarcity, which leads to overfitting or poor classification accuracy depending on the deep learning model. According to Xu et al. (2019), transfer learning is critical in overcoming the data scarcity problem. Xu et al. (2019) proposed three pre-trained CNN models for transfer learning: VGGNet, GoogleNet, and ResNet-50. Panachakel and Ganesan (2021) used ResNet50 and a data augmentation technique called sliding window to achieve EEG signal classification accuracy of 95%. To the best of our knowledge, there are no other better research paper than Panachakel and Ganesan (2021) that uses transfer learning to solve the problem of

data scarcity. As a result, we believe that by implementing transfer learning using a pre-trained CNN model called ResNet50, we can achieve higher classification accuracy with less EEG data. Because the use of transfer learning is not a well-proven technique in the field of EEG classification, we believe that Data Augmentation techniques, which are well-proven techniques to address the problem of data scarcity, should be used as a backup to ensure better classification accuracy for this project. As a result, we intend to use Data Augmentation techniques in conjunction with deep learning models such as ShallowConvNet, DeepConvNet, EEGNet, and ResNet-50. As this hypothesis will demonstrate the positive and negative effects of data augmentation techniques in motor imagery classification. Having established the hypothesis, we address the data augmentation techniques in the next section.

2.5-Data Augmentation

In recent years, the use of data augmentation techniques in EEG signal classification has increased, proving to improve the reliability and stability of classification results (Lashgari et al, 2020). The process of data augmentation involves the creation of new data from existing data. According to Lashgari et al. (2020), the most common data augmentation methods for EEG data are noise addition, GAN, sliding window, sampling, Fourier transform, and recombination of segmentation, as shown in figure 4. According to Figure 4, the most commonly used data augmentation methods are sliding window and General Adversarial Network (GAN). However, as shown in Figure 5, the use of a sliding window and noise addition improved classification accuracy above 30% when compared to GAN and other data augmentation methods. As a result, we believe that sliding window and noise addition methods are better suited methods for data augmentation in this project.

To augment new EEG data, various types of noises such as Gaussian, Poisson, Salt, Pepper, and others can be added to the EEG signal (Wang et al., 2018). Adding Poisson, Salt, and Pepper noise causes feature changes in the EEG signal, resulting in poor classification accuracy (Wang et al., 2018). Adding gaussian noise, on the other hand, preserves the originality of the EEG signal while augmenting the EEG data (Wang et al., 2018). Based on the facts stated above, we consider using Gaussian noise to augment the EEG data using the noise addition method. The project implementation is discussed in the following section.

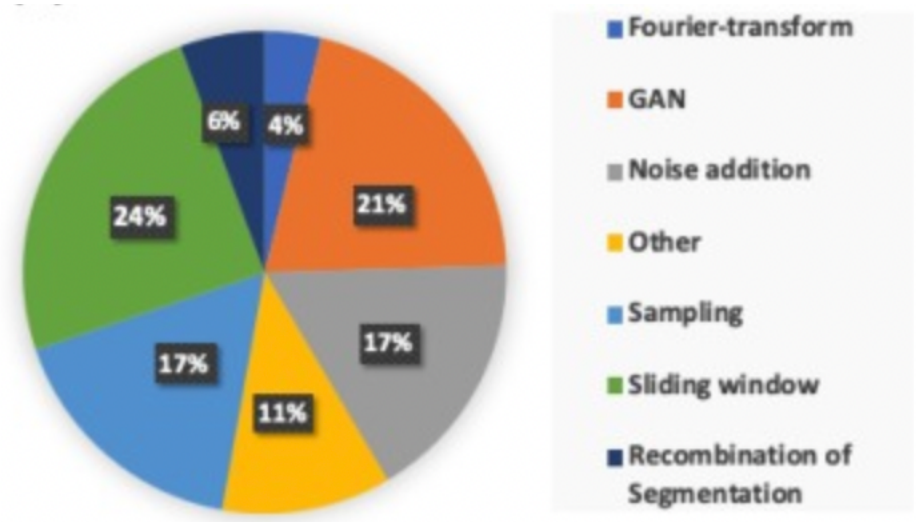

Figure 4 Pie chart representation of various data augmentation methods as reviewed by Lashgari et al. (2020). Image taken from Lashgari et al. (2020)

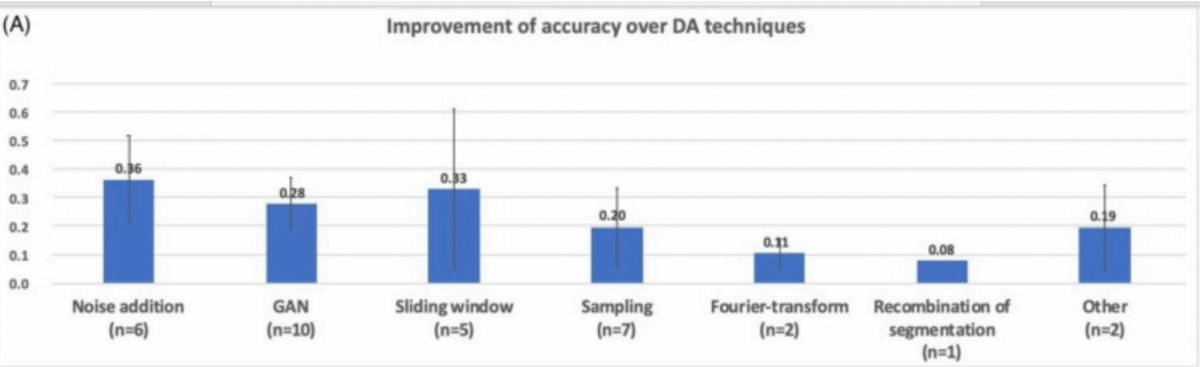

Figure 5 Improvement of EEG classification accuracy over various data augmentation techniques. n represents the number of papers as reviewed by Lashgari et al.(2020). Image taken from Lashgari et al. (2020).

3-Implementation

3.1-Electroencephalogram Datasets

The dataset used for this project is BCI dataset IIa from Graz University (Leeb et al., 2008). EEG signals are collected from nine subjects at a sampling rate of 250Hz using 22 channels from the international 10-20 systems and three EOG monopolar channels. Figure 6 depicts the electrode positions on the scalp. A band pass filter is used to filter the signals between 0.5Hz and 100Hz. A notch filter is used to filter 50Hz to remove the artefacts caused by power line interference. The sensitivity of the amplifier is 100μV. The BCI paradigm for the dataset consists of four motor imagery task such as

- Class1-Imagination of left hand
- Class 2-Imagination of right hand
- Class 3-Imagination of both feet
- Class4-Imagination of tongue

The dataset yields a total of 288 trails per session for each subject by combining 6 runs, with each run accounting for 48 trails because each class accounts for 12 trials. As a result, four classes of twelve trails each yield 48 trails per run.

A 5-minute recording was made at the start of the session to estimate the influence of EOG. The 5-minute recording was subdivided into

- Looking at fixation point for 2 minutes
- Closing the eyes for one minute
- Moving the eyes around for one minute.

A fixation cross appeared on the screen at the start of the trial, accompanied by a beep tone, to alert the subject to the start of the trial. After 2 seconds, a cue in the form of an arrow appears for 1.25 seconds. As shown in the table 3, four arrows are used in the experiment to represent four classes. The cue encouraged the subject to imagine the motor movement corresponding to the arrow displayed on the screen until the fixation cross vanished from view. The duration of fixation cross shown in the screen was 6 seconds. After the fixation cross disappears from the screen, the subject was allowed a short break followed by the next trial. The trial process is illustrated in the figure 7.

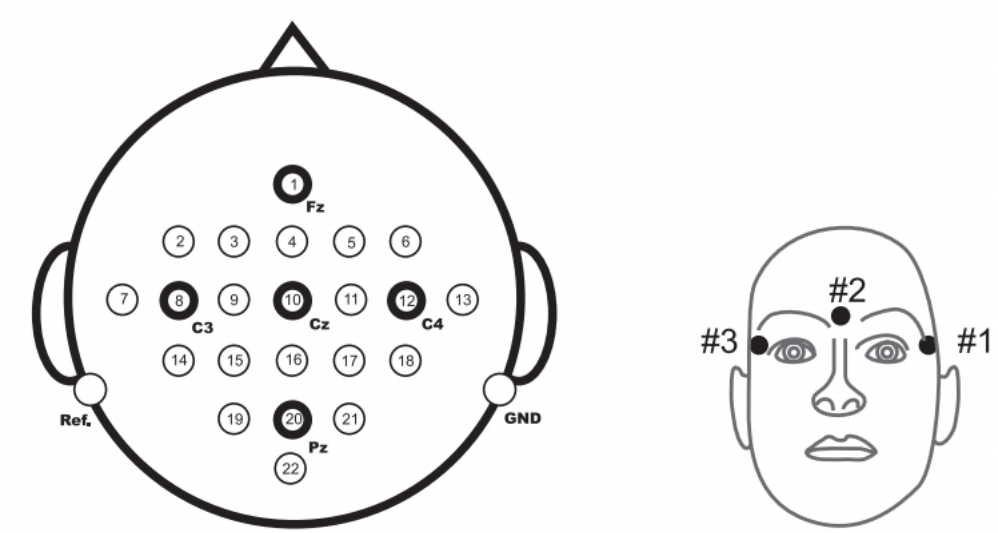

Figure 6 Left image represents electrodes positions and channel names. Right image represents the EOG electrode positions. Image taken from Leeb et al. (2008).

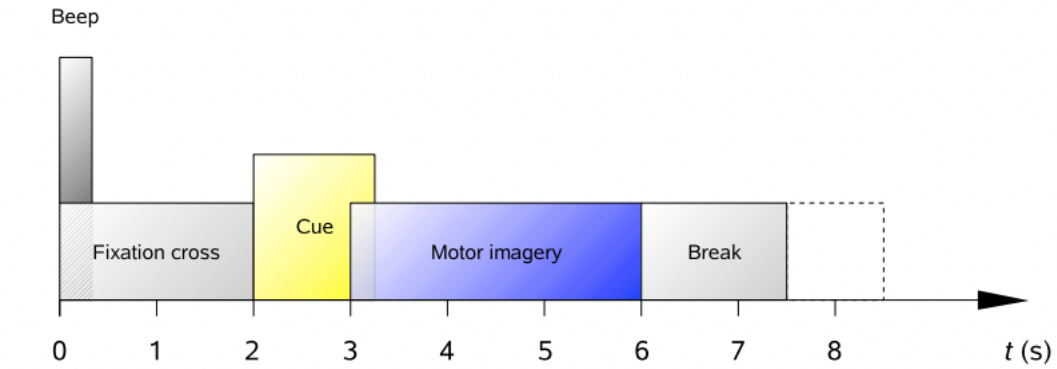

Figure 7 Trial process. Image taken from Leeb et al. (2008).

Table 3 Represents arrow direction and the corresponding motor imagination class.

Arrow direction	Class
Left	Left hand
Right	Right hand
Down	Both feet
Up	Tongue

3.2-Experimental procedure

The Python programming language is used to develop the software that will decode the motor imagery. GoogleColab is used to run the code. The MNE library is used to load the data, filter and epoch. The open-source Tensorflow library will be used to implement the CNNs (ResNet-50, ShallowConvNet, DeepConvNet and EEGNet). Following the extraction of the subject's EEG data for imagined motor prompts, the data is used in both the augmentation and non-augmentation forms, as shown in Figure 8. For the classification task using pretrained CNN ResNet-50, the augmented data (figures 9 and 10) and non-augmented data (figure 11) are used separately in the form of Cross Covariance Matrix (CCV). For the classification task, augmented data (figures 9 and 10) and non-augmented data (figure 11) are used separately, with CNNs such as ShallowConvNet, DeepConvNet, and EEGNet. Finally, using a combination of data augmentation methods and deep learning models, we have 12 different classification results per subject. The approach with the highest mean classification accuracy across all subjects will be considered the best approach for decoding motor imagery tasks. Following the establishment of the project methodology, the data pre-processing is demonstrated in the following section.

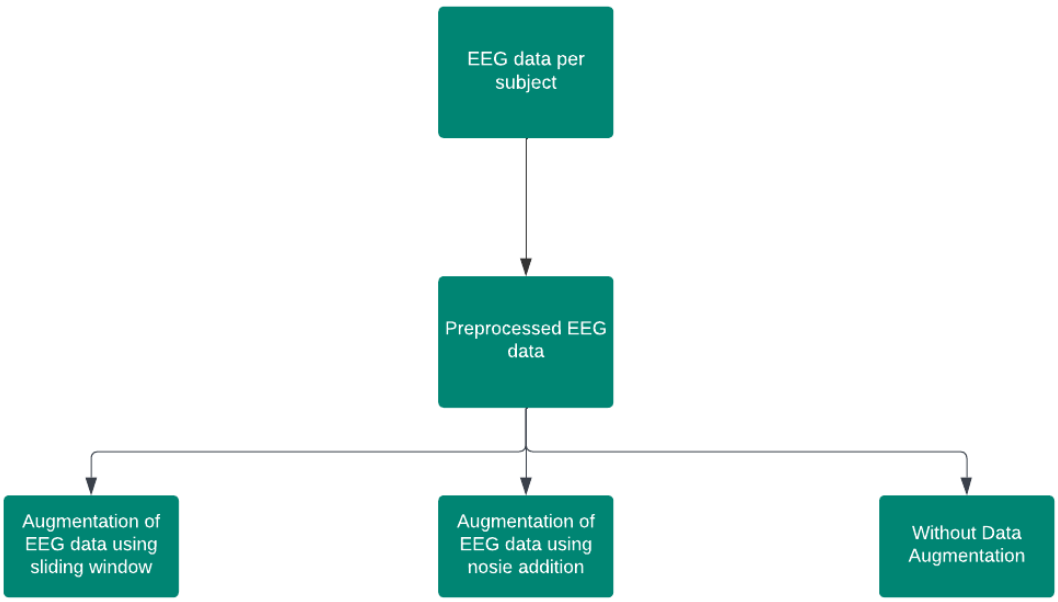

Figure 8 Top level software flowchart for decoding motor imagery.

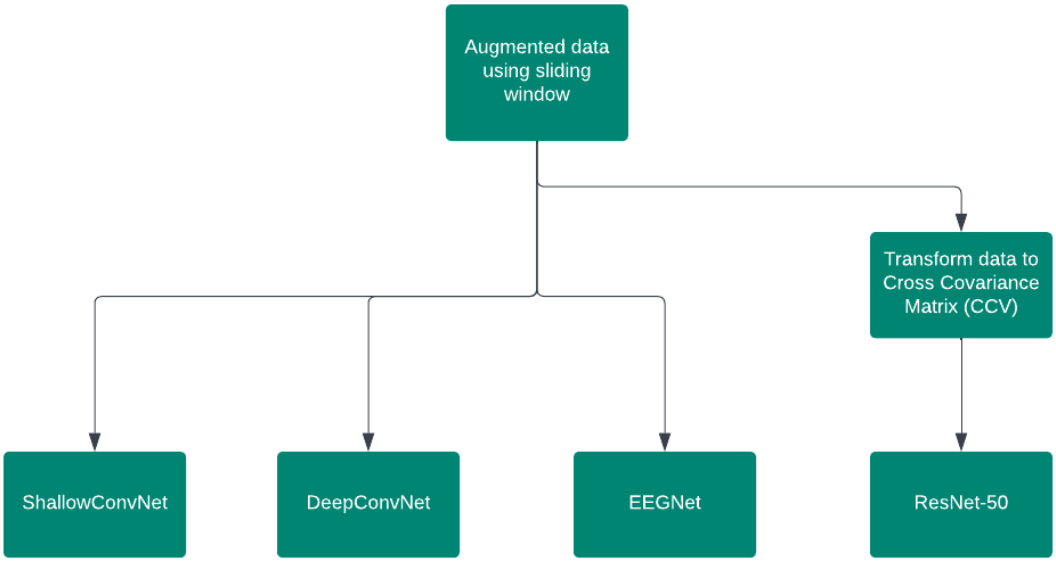

Figure 9 Software flowchart for decoding the motor imagery using sliding window augmented EEG data

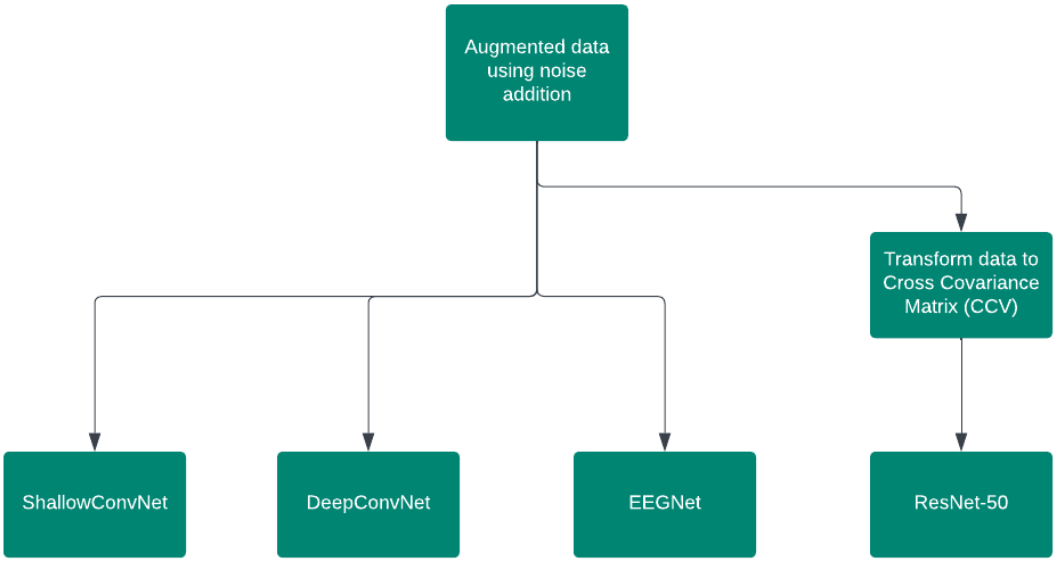

Figure 10 Software flowchart for decoding the motor imagery using noise addition augmented EEG data

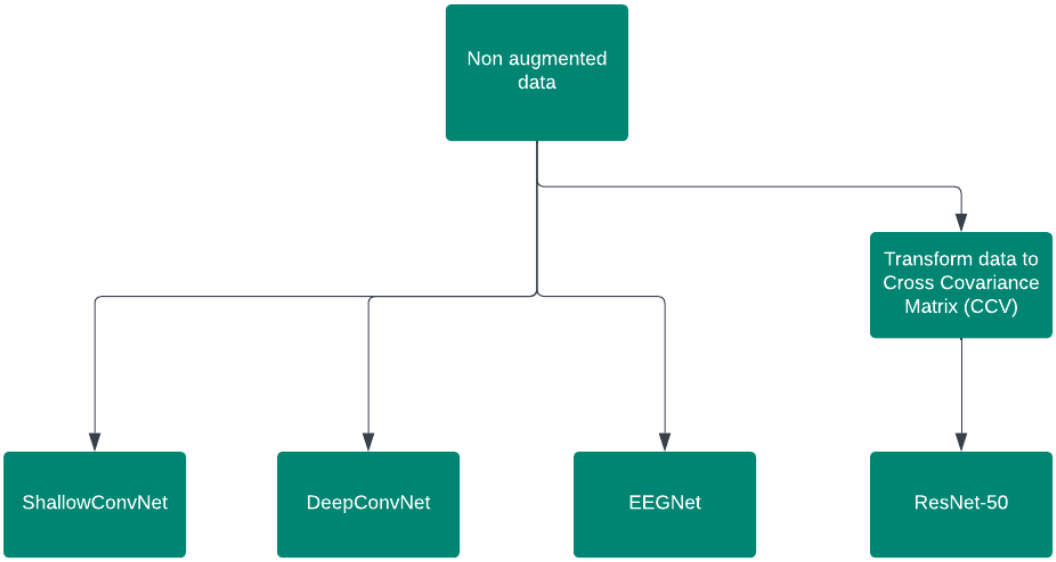

Figure 11 Software flowchart for decoding the motor imagery using noise addition augmented EEG data

3.3- Data pre-processing

The subject's EEG data is saved with the extension'. gdf' and A0xT as the filename. The letter x represents the subject number. All of the subject files are saved to Google Drive, which is mounted with the Google Colab environment. As shown in Figure 12, the relevant subject EEG data file is attached under the variable name 'file.'

```
from google.colab import drive
drive.mount('/content/drive')
file='/content/drive/MyDrive/BCICIV_2a_gdf/A01T.gdf'
```

Figure 12 Code snippet for extracting EEG data from the subject specific file.

The raw EEG data is read from the file using the MNE library function "mne.io.read raw gdf." We examined the EEG data file information to ensure that the data matched the dataset specification. The output of the variable “file” is shown in the figure 13. Figure 13 shows that the information in the EEG data file corresponds to the information provided by the dataset's author. Butterworth filter with the frequency range 4Hz-40Hz is used to filter the original EEG data to capture the highest brain activity during the motor imagination. The filter EEG data is verified by plotting the power spectral density as shown in figure 15 and compared with power spectral density plot (Figure 14) of EEG data prior to filtering. Figure 15 shows that the energy variations are lower after 40Hz and before 4Hz.

```
Info | 7 non-empty values
bads: []
ch_names: EEG-Fz, EEG-0, EEG-1, EEG-2, EEG-3, EEG-4, EEG-5, EEG-C3, EEG-6, ...
chs: 25 EEG
custom_ref_applied: False
highpass: 0.5 Hz
lowpass: 100.0 Hz
meas_date: 2005-01-17 12:00:00 UTC
nchan: 25
projs: []
sfreq: 250.0 Hz
```

Figure 13 EEG data information

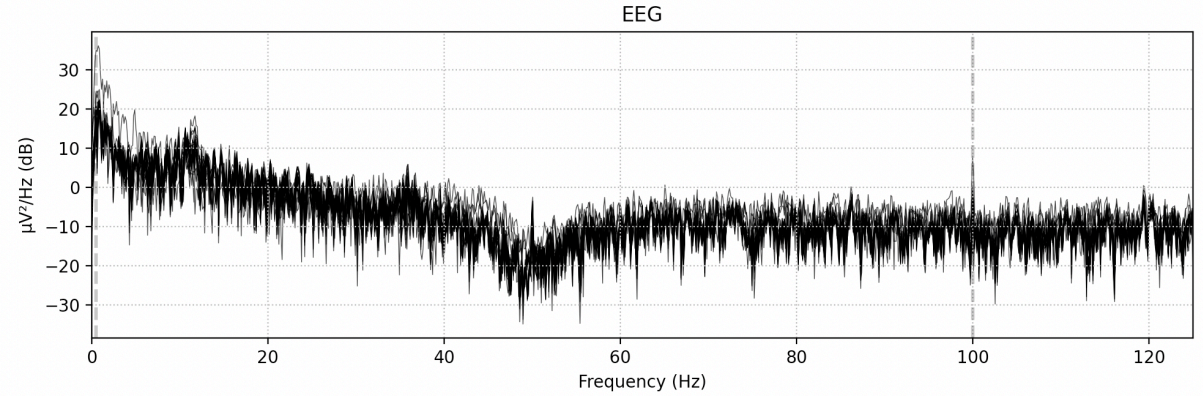

Figure 14 Power spectral density plot of original EEG data

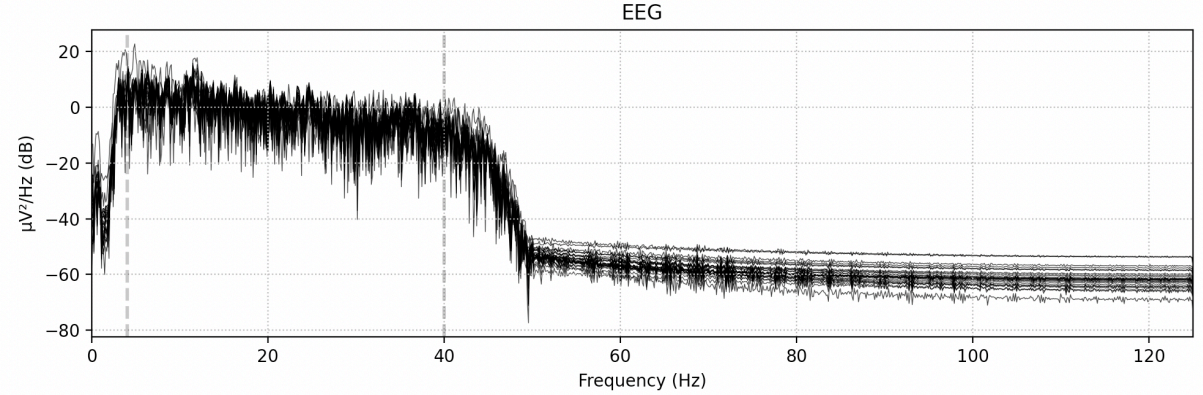

Figure 15 Power spectral density plot of filtered EEG data between 4Hz and 40Hz.

The event id for each cue is required to epoch the EEG data according to the class. The function mne.events from annotations() is used to extract the events and their respective event ids.

The event id returned "'1023': 1,'1072': 2,'276': 3,'277': 4,'32766': 5,'768': 6,'769': 7,'770': 8,'771': 9,'772': 10". Table 4 displays the event id and the corresponding cue.

Table 4 Event number list with the corresponding Event_id and Description. Leeb et al. (2008)

Event number	Event_id	Description
1	1023	Rejected trial
2	1072	Eye movements
3	276	Idling EEG (eyes open)
4	277	Idling EEG (eyes closed)
5	32766	Start of a new run
6	768	Start of a trial
7	769	Cue onset left (class 1)
8	770	Cue onset right (class 2)
9	771	Cue onset foot (class 3)
10	772	Cue onset tongue (class 4)

The following code was used to map the event description to the event number in order to visualise the event on the EEG plot. "*mapping_events = {6:'Trial start',7: 'left', 8: 'right', 9: 'feet',10: 'tongue'}*".

Created an annotation object by the following code

"*annotObject = mne.annotations_from_events(*

events=events, event_desc=mapping_events, sfreq=raw.info['sfreq'],

orig_time=raw.info['meas_date'])".

We set the annotation object to the raw EEG data by the following code "*raw.set_annotations(annotObject)*". We plot raw EEG data using the code "*raw.plot()*". Figure 16 displays the plotted EEG data. From figure 16, the trial begins at 914.6s, followed by the left cue (left hand class) at 916.6s, and the next trial begins at 922.8s. The time interval between the trial was 8.2s. From the visual inspection of the EEG plot, we confirmed that the timing between the events matches as per the figure 17. After extracting the event id and visually inspecting the EEG data, we epoch the EEG data in the following step.

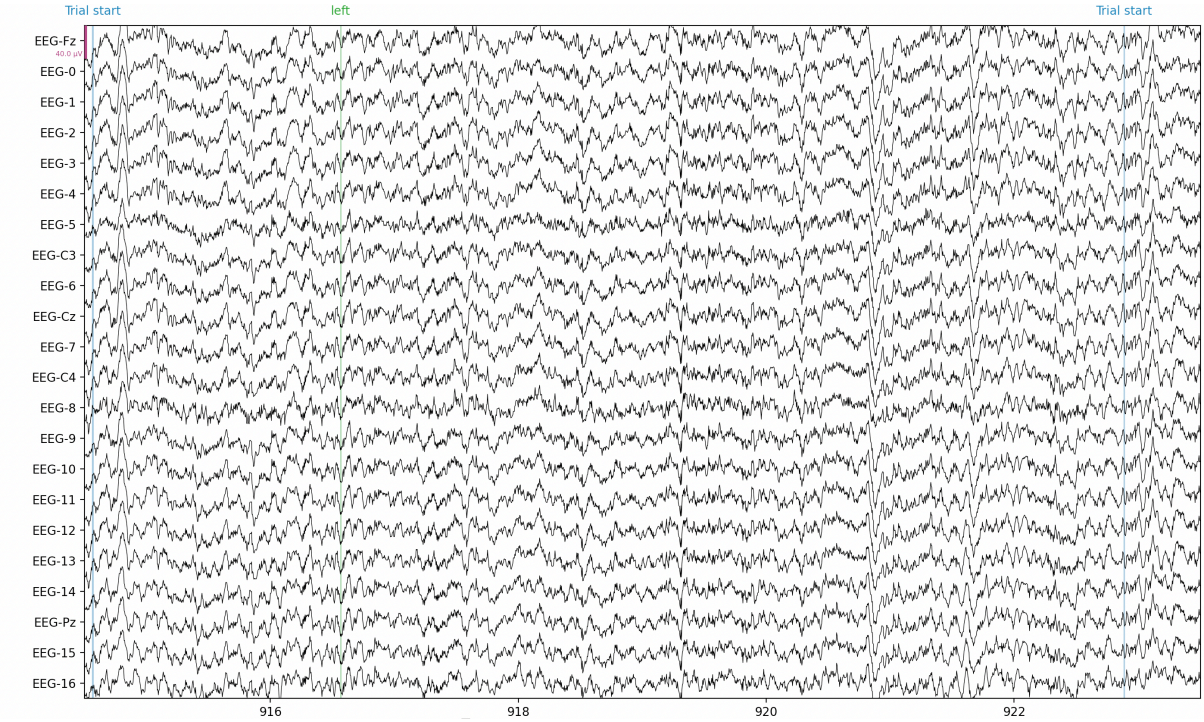

Figure 16 EEG plot from duration 914.5s with a duration of 8.5s and 22 channels

Choo et al. (2020) and Altuwaijri and Muhammad. (2022) extracted 4.5 seconds of EEG data per trial, beginning 0.5 seconds before the cue and ending at the end of motor imagery. The number of samples per trial is 1125, which is calculated by multiplying the sampling rate by the trial length in seconds (250Hz x 4.5s =1125). However, we extracted a trial length of 4s because a whole number makes parameter calculation in the sliding window process easier. As illustrated in Figure 17, This project's extracted trial began 0.5s before the cue and ended at 4.5s. Each trial contains 1000 samples (250Hz x 4s =1000). Having established the trial length, we look at epoch the EEG data as per the class.

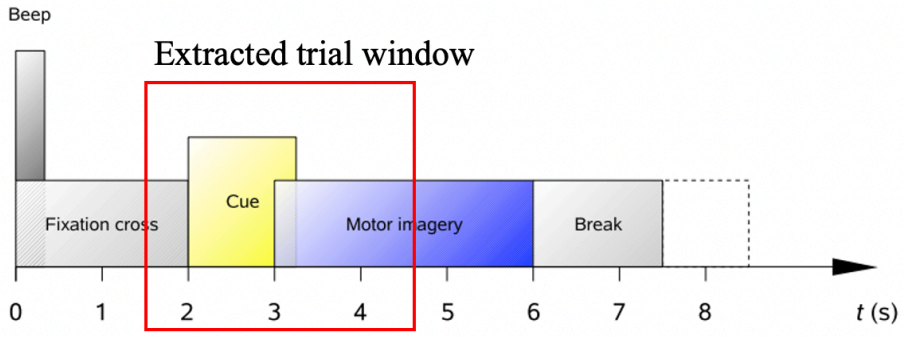

Figure 17 Trial process. Image adapted from Leeb et al. (2008).

Figure 18 depicts the Epoch of EEG data and the organisation of the epoch data to the respective class. Individual empty lists for four classes were created and named with the class names (LeftHand, RightHand, Tongue, BothFeet). The output shape of the raw EEG data is (25, 672528) where the row describes the number of EEG channels, and column represents the EEG samples. The raw data for each event in event list is segmented at the start of 125 timepoints and ended at 1125 timepoints. Timepoints can be translated to event time by the following equation.

$$Time(s) = \frac{Timepoint}{Sampling\ frequency\ (Hz)}$$

125 timepoints translates to 0.5s (125/250Hz=0.5s) and 1125 timepoint translates to 4.5s (1125/250Hz=4.5s). The code snippet for epoch the EEG data as per the class is shown in figure 19.

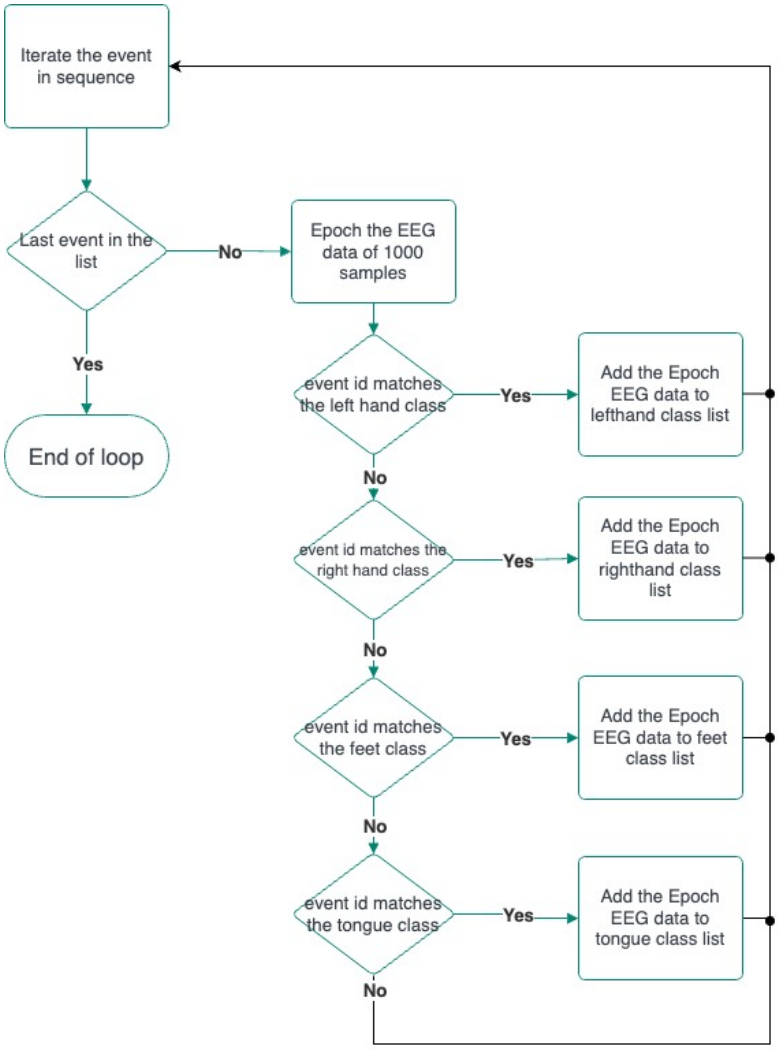

Figure 18 Software flowchart for Epoch EEG data as per the classes.

```
LeftHand = []
RightHand = []
BothFeet = []
Tongue = []
for i in range(events.shape[0]-1):
  Data = rawdata[:,events[i,0]+125:events[i,0]+1125]
  if (events[i,2] == 7):
    LeftHand.append(Data)
  elif (events[i,2] == 8):
    RightHand.append(Data)
  elif (events[i,2] == 9):
     BothFeet.append(Data)
  elif (events[i,2] == 10):
     Tongue.append(Data)
```

Figure 19 Code snippet for epoch EEG data as per the class. Code was adapted from Xin Gao from his unpublished work.

Having epoch, the EEG data in relation to the classes, and the EOG channels must be removed. To remove the EOG channels, we used the numpy.delete() function to remove the last three rows (23,24,25) from each class list. The code snippet for removing the EOG channels is shown in figure 20. Figure 21 depicts the output shape of a four numpy array. Except for the left-hand class, which has 71 trials, the total number of trials across four classes is 287. In addition, the number of channels is 22 and the number of samples per trial is 1000 as per the figure 21.

```
LeftHand= np.delete(LeftHand, [22,23,24], 1)
RightHand = np.delete(RightHand, [22,23,24], 1)
BothFeet = np.delete(BothFeet, [22,23,24], 1)
Tongue= np.delete(Tongue, [22,23,24], 1)
```

Figure 20 Code snippet of removing EOG channels from EEG data of four classes.

```
LeftHand shape = (71, 22, 1000)
RightHand shape = (72, 22, 1000)
BothFeet shape = (72, 22, 1000)
Tongue shape = (72, 22, 1000)
```

Figure 21 Four class list output shape

To divide the data into training, validation, and testing sets, all of the class EEG data is concatenated into a single variable called EntireClassData. Furthermore, each class's labels are concatenated into a single variable called EntireClassLabels. The code shown in Figure 22 executes the aforementioned process.

```
leftLabels = [0] * LeftHand.shape[0]
rightLabels = [1] * RightHand.shape[0]
footLabels = [2] * BothFeet.shape[0]
tongueLabels = [3] * Tongue.shape[0]
EntireClassLabels = leftLabels + rightLabels + footLabels + tongueLabels
EntireClassLabels = np.array(EntireClassLabels)
EntireClassData = np.concatenate((LeftHand, RightHand, BothFeet, Tongue))
```

Figure 22 Code snippet for concatenate the entire class of EEG data and their respective labels.

EEG signal contains features which have various scale and dimensions. It makes the deep learning model to biased prediction due to misclassification error and accuracy rates (Mulani, S., 2022). Therefore, it is mandatory to standardize the EEG data. EEG signals across all the channels are standardized to have a particular mean and standard deviation. Ideally, mean of 0 and standard deviation of 1. Standardized signal can be obtained by the below equation.

$$Standardized\ signal = \frac{Signal - mean\ of\ signal}{Signal\ Standard\ deviation}$$

Where Signal= [Number of channels, Samples]

Mean of signal is obtained using the numpy.mean() function and standard deviation of the signal is obtained using numpy.std() function.

After standardizing the signal, the EEG data is divided into training, validation, and testing using the sklearn library's train test split() function. The training set contains 60% of the EEG data, while the validation and testing sets contain 30% and 10% of the EEG data, respectively. The code for splitting the data is shown in the figure 23. Having established the implementation of EEG data pre-processing, in the section we look at the implementation of data augmentation methods such as noise addition and sliding window.

```
X_train, X_rem, Y_train, Y_rem = train_test_split(EntireClassData, EntireClassLabels, test_size=0.4, random_state=42)
X_validate,X_test,Y_validate,Y_test  = train_test_split(X_rem, Y_rem, test_size=0.1,random_state=42)
```

Figure 23 Code snippet for splitting the data into training, validation and testing set

3.4- Noise addition

We augmented the training data with gaussian noise in order to extract new EEG data. As a result, the training data is multiplied by a factor of two. The following equation expresses the probability density function P of a gaussian random variable z (Wang et al., 2018).

$$P_G(Z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

Where the σ represents the standard deviation, z represents the grey level and μ represents the mean value. Because the original data is standardised to have a mean value of zero, the mean value is set to zero. This means that the amplitude of the original data and the augmented data are the same. The standard deviation used in this project are 0.001,0.01,0.1,0.02,0.2 and 0.5. The choice of the standard deviation values is based on the research papers (Choo et al, (2020), Wang et al. (2018) and Roy. (2022)). The gaussian noise is generated with the numpy.random.normal function (mean, standard deviation, size). The size is (22,1000), where 22 is the number of channels and 1000 is the sample size. As illustrated in Figure 24, the generated gaussian noise (figure 24, a) is combined with the original EEG data (figure 24, b) to produce a new sample (figure 24, c). The preceding procedure is repeated for each sample in the training set, and the resulting new samples are stored in a list with their respective class labels. The code for the above process is shown in the figure 26. Finally, the generated samples are concatenated with the original samples. The training sample size is 172, and the training sample size after augmentation is 344.

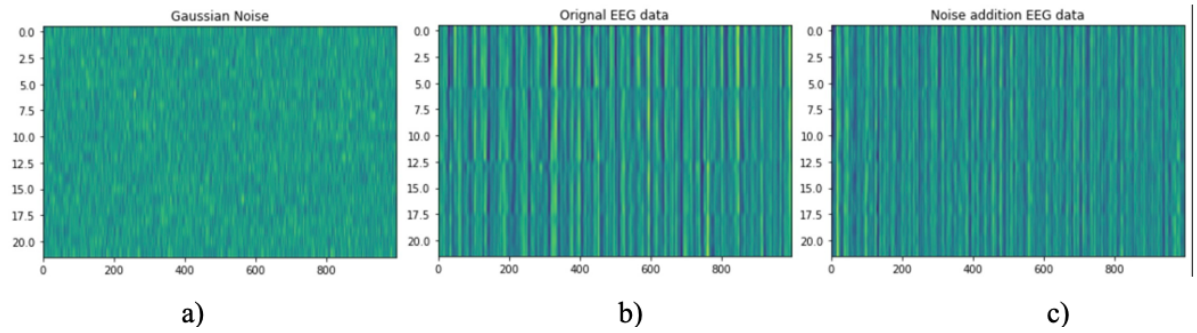

Figure 24 a) Generated gaussian noise b) Original EEG data c) Generated EEG data with noise addition.

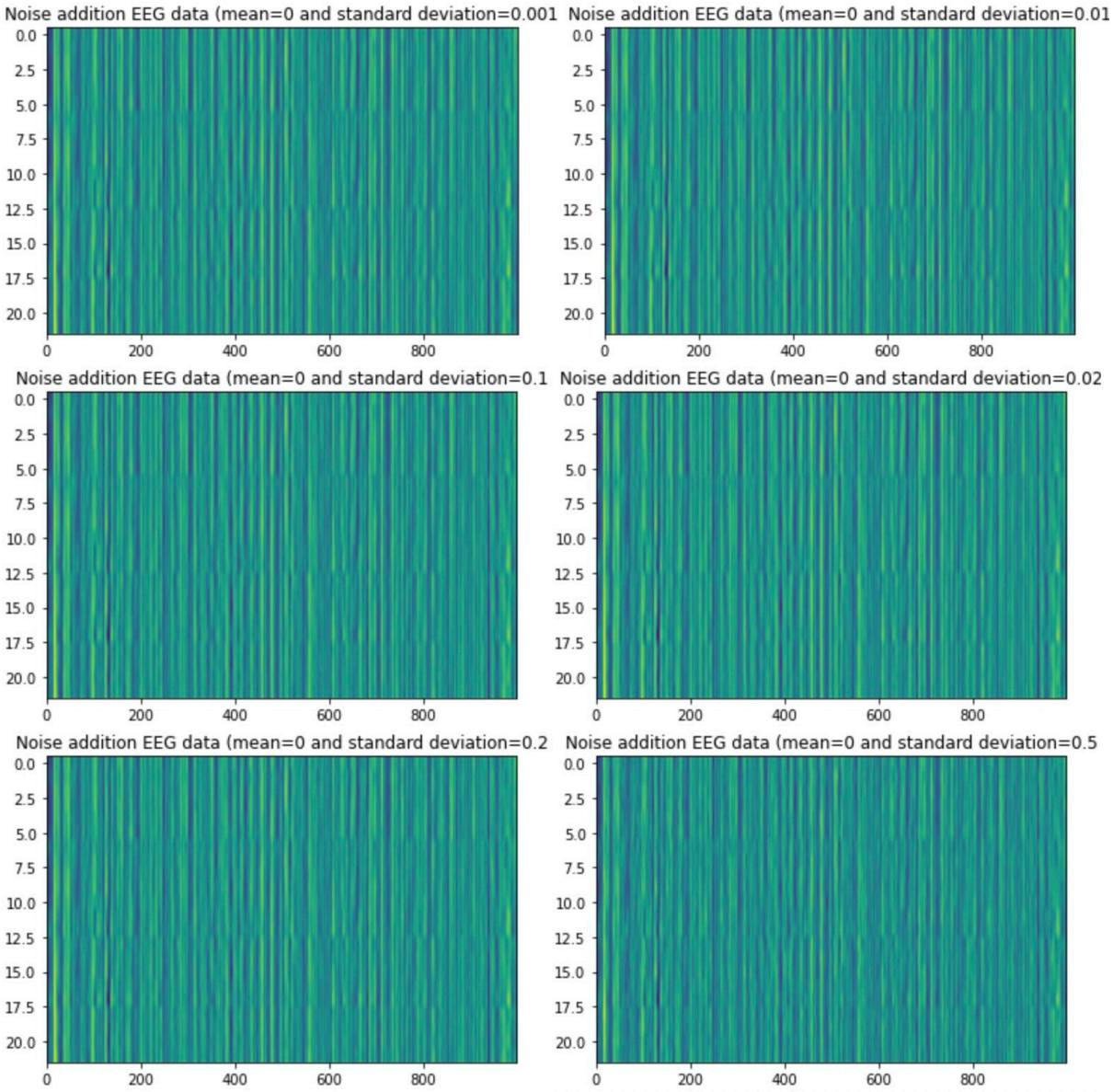

Figure 25 Generated EEG samples with various standard deviation values and a mean value of zero.

```
mu=0.0 # mean value
std =0.5 # standard deviation value
def gaussian_noise(x,mu,std):
    noise = np.random.normal(mu, std, size = x.shape)
    x_noisy = x + noise
    return x_noisy

TrainData_temp=[]
TrainData_temp_label=[]
for i in range(X_train.shape[0]):
  NoisyData=gaussian_noise(X_train[i],mu,std)
  TrainData_temp.append(NoisyData)
  TrainData_temp_label.append(Y_train[i])


TrainData_temp=np.array(TrainData_temp)
TrainData_temp_label=np.array(TrainData_temp_label)
print(TrainData_temp.shape,TrainData_temp_label)


TrainData=np.concatenate((X_train,TrainData_temp))
TrainLabel=np.concatenate((Y_train,TrainData_temp_label
print(TrainLabel.shape, TrainData.shape)
```

Figure 26 Code snippet for augmenting EEG data using noise addition.

3.5- Sliding window

The sliding window length is set to 2s (500 samples) for this project, with stride lengths of 1s (250 samples) and 2s (500 samples). Having overlaps of 50% and 0% is critical in determining the impact of the overlapping window on classification accuracy. The two sliding window methods used in this project are depicted in Table 5. As illustrated in Figure 27, two windows are extracted from each trial with no overlaps. As illustrated in Figure 28, three windows are extracted from each trial, with 50% overlap. Furthermore, because the deep learning model input shape must match during training, validation, and testing, trails in training, validation, and testing sets are augmented using the respective sliding window method. Code snippet for the implementation of sliding window is shown in figure 29. Having established the implementation of data augmentation methods, we look at the implementation of deep learning models in the next section.

Table 5 Parameters used for sliding window method.

	Window length	Stride length	Overlap (%)	Number of samples generated per trial
Method 1	500	250	50	3
Method 2	500	500	0	2

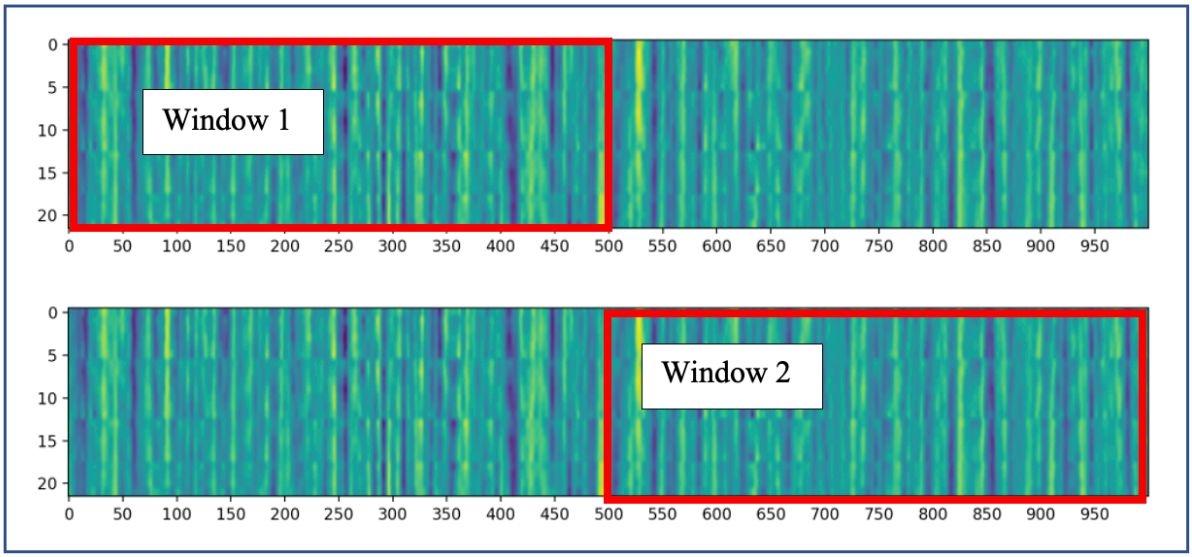

Figure 27 Representation of sliding window for window size 500 samples and step size 500 smaples.

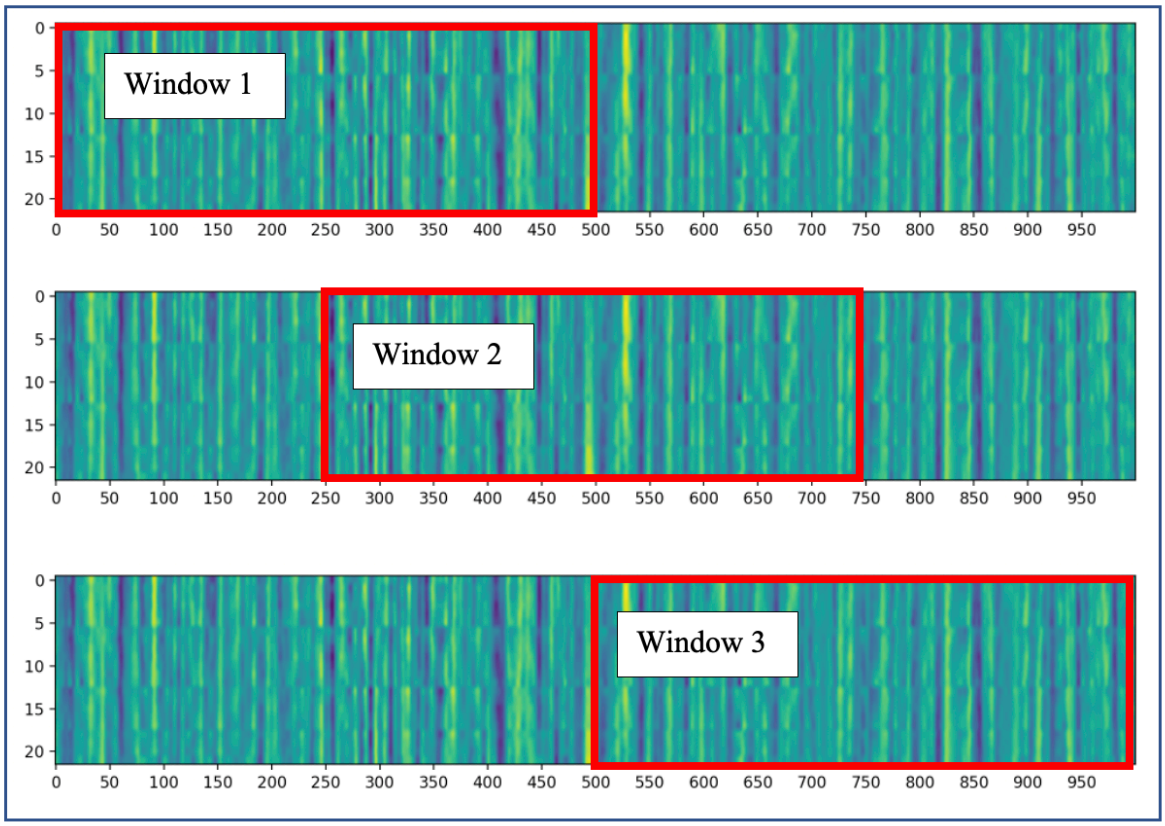

Figure 28 Representation of sliding window for window size 500 samples and step size 250 samples.

```
# sliding window for train data
# Initialise two list for appending the augmented data and the respective labels.
TrainData=list()
TrainLabel=list()


step_size=250
window_size=500
samples   = X_train.shape[2]  # trial sample size
cuts  = list(range(0, samples, step_size)) # cuts=[0,250,500,750]

for i in range(X_train.shape[0]): #iterate the trials in sequence
  for j in cuts:                  #iterate the cuts list in sequence
      if j + window_size <= samples: # cut the windows until the end of trial sample.
        TrainData .append(X_train[i:i+1, :, j:j+window_size]) #apppend the cut EEG data into the Train data list
        TrainLabel .append(Y_train[i]) #append the class label for the particular trial number i.


TrainData  = np.concatenate(TrainData, axis=0) #concatenate the first and second axis from the train data list
                                               #(170,3,22,500) to (510,22,500)
TrainLabel = np.array(TrainLabel)
print('sliding window is ', TrainData.shape, TrainLabel.shape)
```

Figure 29 Code snippet for augmenting EEG data using sliding window method.

3.6-ShallowConvNet

The ShallowConvNet's first layer performs temporal convolution, while the second layer performs spatial filters. Shallow Convnet has a kernel size of 25, which helps it perform larger transformations in the temporal convolution layer. The temporal and spatial convolution layers mimic the bandpass technique by separating the various frequency bands from the EEG signal and applying Common Spatial Pattern (CSP) to extract the spatial pattern that allows the trials to be differentiated using the filtered trial signal power. Furthermore, using squaring nonlinearity, a mean polling layer, and a logarithmic activation function, the trial log variance for each frequency band is computed. The log variance is made up of the feature vectors from the band pass filtered signals, which act as a feature map to predict the trial's class label. Each neuron in the deep learning model has an activation function that determines whether the neuron should be activated or not based on whether the neuron's input is relevant to the deep learning model prediction. Square and logarithmic activation functions are used in the ShallowConvNet architecture. Batch normalisation is used to avoid overfitting, speed up learning, and regularise the deep learning model (Altuwaijri, G.A. and Muhammad, G., 2022). The network architecture is shown in the figure 30. The model architecture and kernel size, pool size and stride length are based on the research paper by Schirrmeister et al. (2017). The code for the implementation of ShallowConvNet is based on vlawhern. (2022). The input shape is (X,Y,Z), where X represents the number of channels as rows, Y as columns, and Z is set to 1 because the model has 2D convolution.

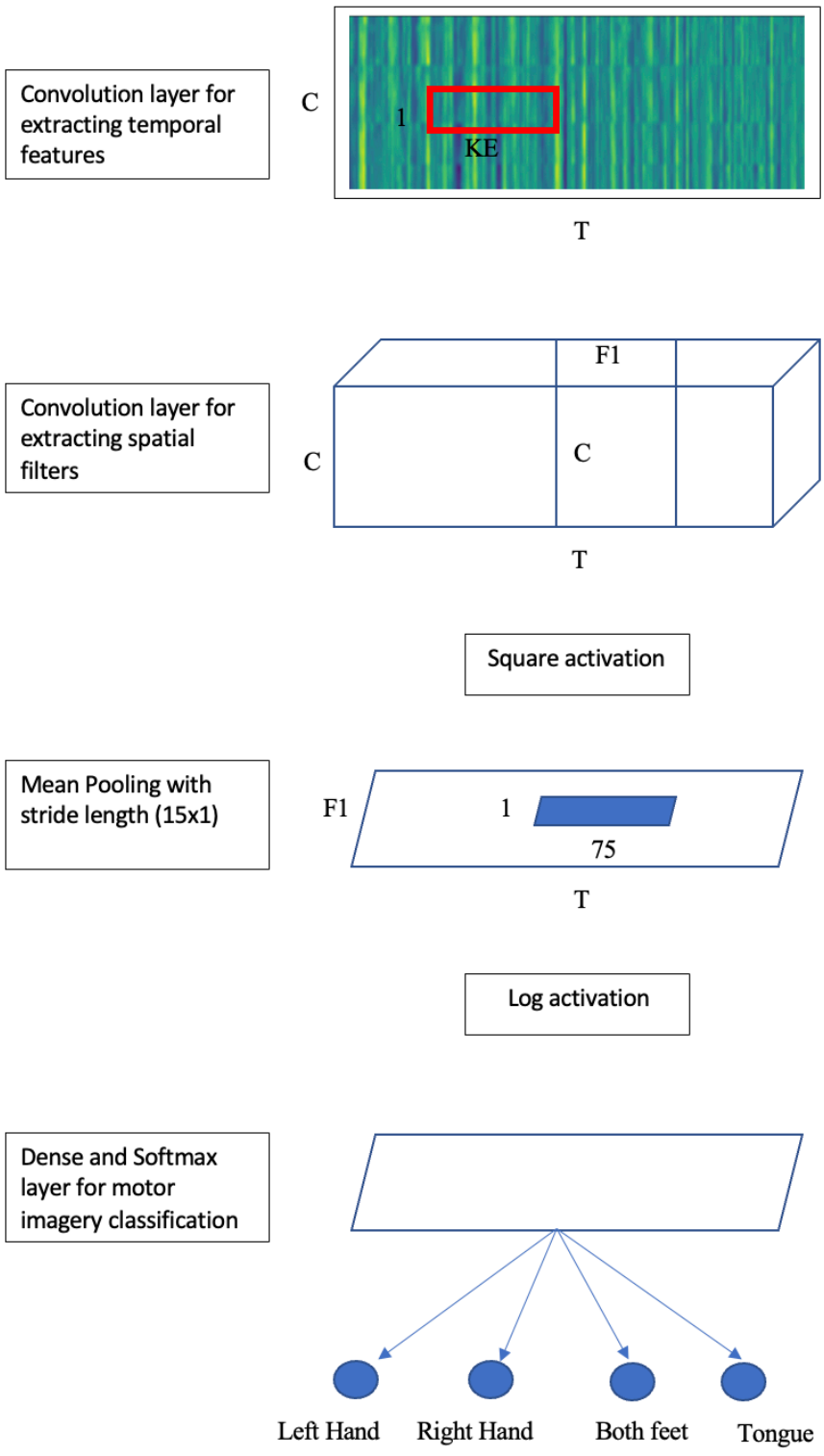

Figure 30 ShallowConvNeT model architecture and parameters where C=22, KE=25, T=1000 or 500 and F1=40.

3.7-DeepConvNet

DeepConvNet has the ability to extract multiple features from raw EEG signals and is not limited to a single feature. The first block is made up of two layers, the first of which performs temporal convolution and the second of which performs spatial convolution. Other 3 blocks of DeepConvNet consists of a single convolution layer and max pooling layer. Max pooling is used instead of square and mean pooling because max pooling sensitivity towards signal phase and power features. Finally, a dense layer with the SoftMax activation function. Each block consists of exponential linear unit (ELU) as activation function. The network architecture is shown in the figure 31. The model architecture and kernel size, pool size and stride length are based on the research paper by Schirrmeister et al. (2017). The code for the implementation of DeepConvNet is based on vlawhern. (2022). The input shape is (X,Y,Z), where X represents the number of channels as rows, Y as columns, and Z is set to 1 because the model has 2D convolution.

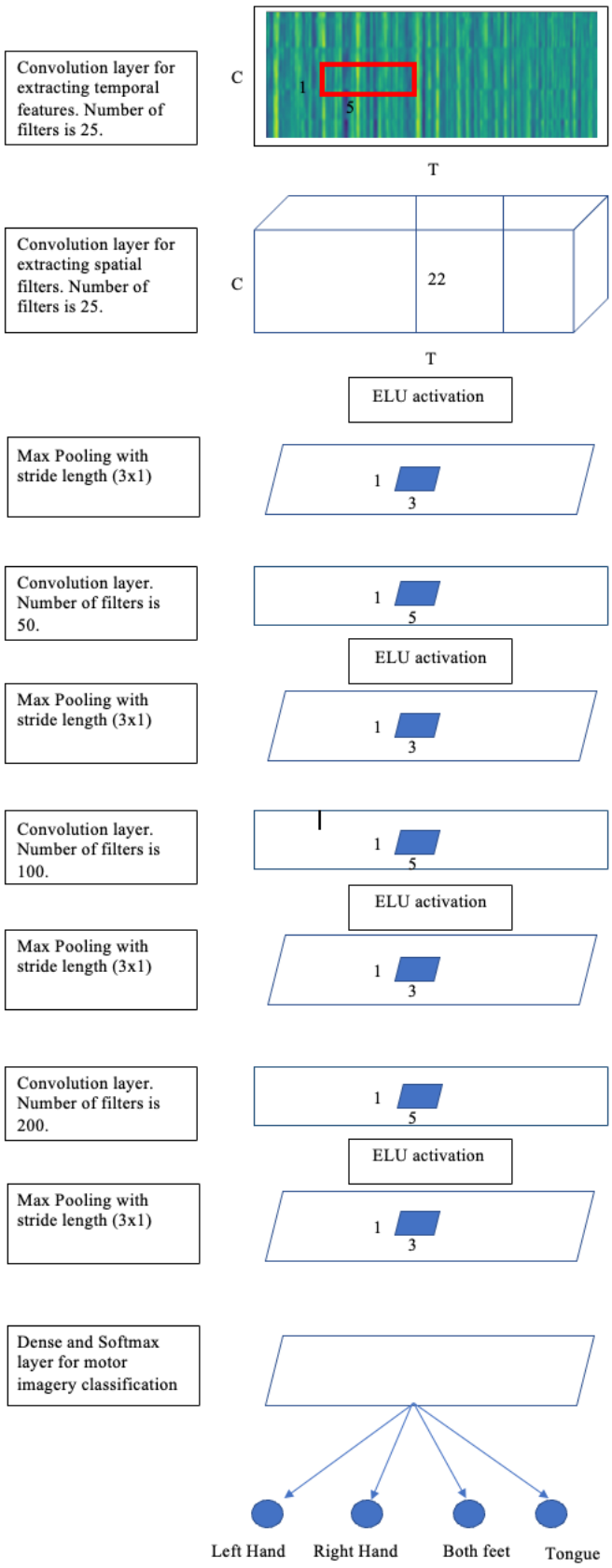

Figure 31 DeepConvNet Model architecture where C=22, T=1000 or 500.

3.8- EEGNet

The EEGNet architecture is divided into two blocks. There are two convolution steps in block 1. The filter size in the first step is (1, X), where X is half the sampling rate. As a result, X is 125 because the dataset's sampling rate is 250Hz. The capture of frequency information above 2Hz is facilitated by proving the filter size with half the sampling rate. The second step in the convolution process is a depth-wise convolution with filter size (1, C), where C is the number of channels and C=22. Depthwise convolution encourages spatial filter learning for each temporal filter extracted from the first convolution layer. This method promotes the efficient extraction of spatial filters tailored to each frequency. The depth multiplier (D) parameter in depthwise convolution is set to 2, allowing the model to learn two spatial filters for each feature map. Prior to using the ELU activation function, batch normalisation is used. Prior to the dropout layer, an average pooling layer is used, with the pool size set to (1,64), which reduces the sampling rate to 64Hz to reduce the parameters. A dropout of 0.5 is used to regularise the model. A separable convolution layer with a filter size of (1,64) is used in block 2, where the 64 represents the sampling rate of the data from block 1. The number of parameters to fit is reduced significantly by separable convolution, which reduces the computational load and increases the learning rate. Following the separable convolution is batch normalisation and the ELU activation function. An average pooling layer is introduced to further reduce the dimension, and the pool size is set to (1,16). Finally, the learnt features are fed to the SoftMax layer for the classification task. The network architecture is shown in the figure 30. The model architecture and kernel size, pool size and stride length are based on the research paper by Schirrmeister et al. (2017). The code for the implementation of EEGNet is based on vlawhern. (2022). The input shape is (X, Y, Z) where X represents the number of channels as rows, Y represents the samples as column and Z is set to 1 because the model has 2D convolution.

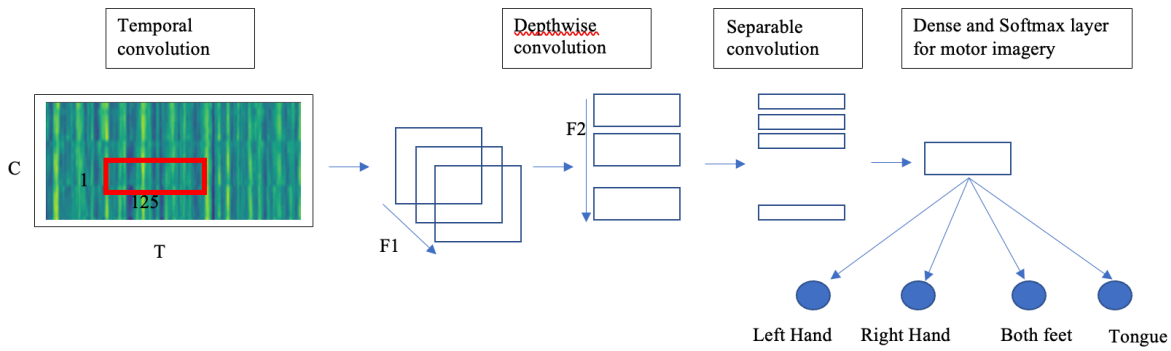

Figure 32 EEGNet model architecture. where C=22, T=1000 or 500,F1=8 and F2=16..

3.9-ResNet50

We used a pretrained CNN called ResNet-50 that was pre-trained on ImageNet to implement transfer learning. ImageNet is a dataset that contains roughly 14 million images. ResNet50 was thus pre-trained on 14 million images. ResNet50 classifies 1000 images by default using the softmax activation function. The classification layer of ResNet-50 is replaced in this project by two fully connected layers (FC1 and FC2) with 128 and 64 neurons, respectively, and four neurons for the output layer to classify four classes. The inspiration for the two fully connected layer prior to the output layer was taken from the research paper by Panachakel and Ganesan (2021). Architecture of this deep learning model is shown in the figure 33.

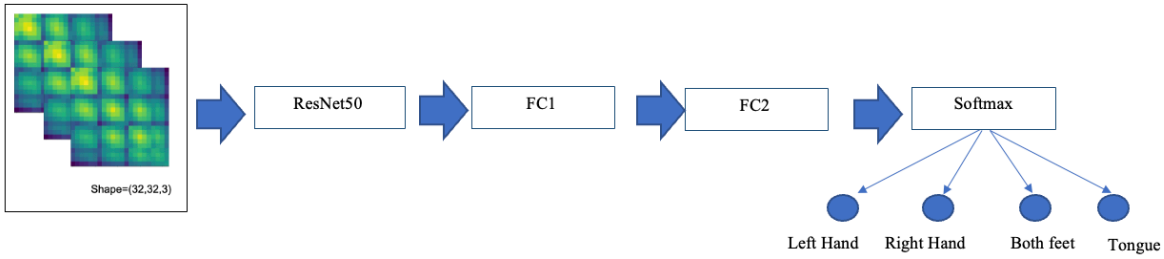

Figure 33 ResNet50 model architecture with the trainable layers.

The ResNet50 model's minimum input shape is (32 x 32 x 3). The EGG data input shapes are (22 x 1000) and (22 x 500), where 22 is the channel number and 1000 and 500 are the samples, respectively, depending on data augmentation methods. We used the Channel Cross Covariance Method to derive the ResNet50 input shape from the EEG data input shape. The Channel Cross Covariance (CCV) method reduces EEG data dimensionality while capturing information flow across multiple channels (Saha et al.,2019). The CCV matrix produces a square matrix of shape (number of channels x number of channels). The CCV matrix for this project has the following shape: (22 x 22). Each cell in the matrix represents the linear relationship between the amplitudes of the two channels (Saha et al.,2019). Furthermore, covariance between same channels equals variance (Saha et al.,2019). The function numpy.cov() returns the CCV matrix for each trail. Figure 34 depicts the CCV matrix of a trial. The number of elements in the x and y axes in figure 34 is 22, which validates the implementation of CCV.

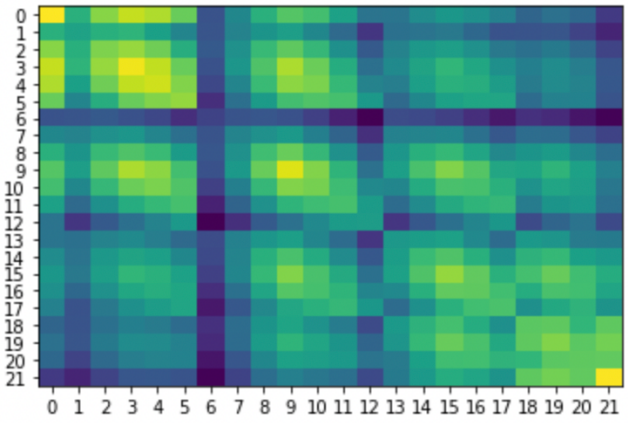

Figure 34 Cross Covariance Matrix of an EEG trial.

ResNet50 required the addition of a Z axis. We use the numpy.newaxis() function to add new axis to the CCV data. After addition of the third dimension, the shape of all the CCV data becomes (22,22,1). The ResNet50 needs 3 channels in the Z axis. To have the three image channels in the Z axis, we used the numpy.repeat() function to repeat the CCV data three times. After the use of numpy.repeat() function, the shape of all CCV data becomes (22,22,3). The ResNet50 requires 32 input shapes along the X and Y axes. As a result, we must increase the dimension of our CCV data from 22 to 32 along the X and Y axes. The Cv2.resize() function is used to resize the EEG data and the shape of CCV data becomes (32,32,3). The above process is shown in the figure 36 and the code snippet is shown in the figure 35. The above process is implemented on training, validation and testing data.

```
import cv2
#Cross covariance matrix
TrainData_cov=[]
for i in range(TrainData.shape[0]):
  cov=np.cov(TrainData[i])
  TrainData_cov.append(cov)
TrainData_cov=np.array(TrainData_cov)
print(TrainData_cov.shape)

#Adding new dimension and repeating the data (22x22) three times
TrainData_new_dim = np.repeat(TrainData_cov[..., np.newaxis], 3, -1)


#Creating an empty array with the dimension (number of trials,32,32,3)
TrainData_resize = np.ndarray(shape=(TrainData_new_dim.shape[0],32,32,3))

#Resize all the trial to shape (32,32,3) from (22,22,3)
for i in range(TrainData_new_dim.shape[0]):
    TrainData_resize[i] = cv2.resize(TrainData_new_dim[i], (32,32),interpolation = cv2.INTER_AREA)
print(TrainData_resize.shape)
```

Figure 35 Code snippet for altering the input shape of training data to suit the input shape of ResNet50. The code for resizing the input shape expect CCV is adapted from kaggle.com (n.d).

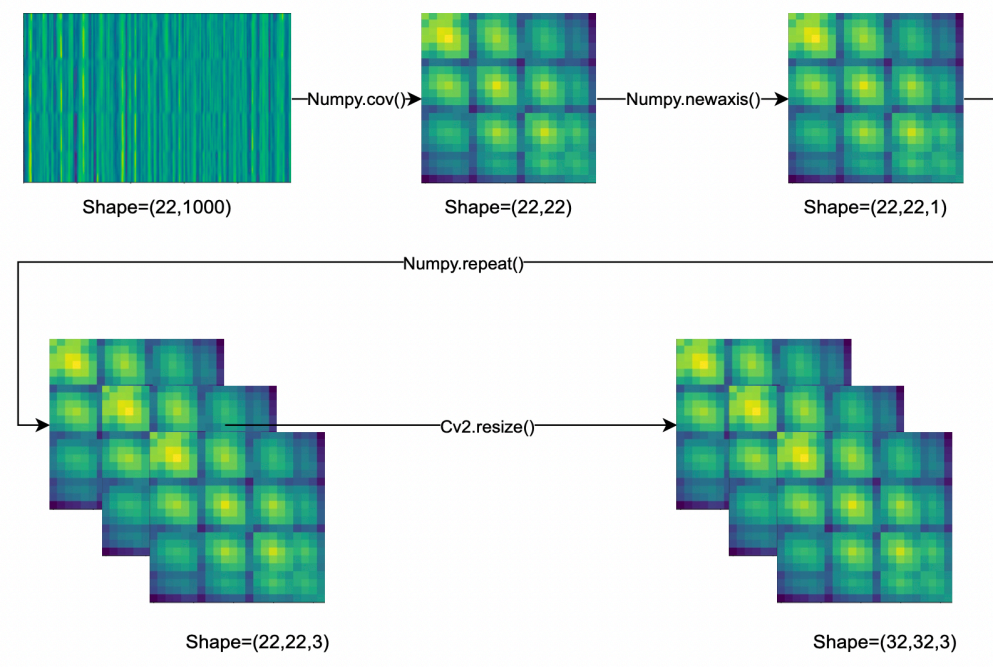

Figure 36 Process of changing the input shape for ResNet50.

4-Experimental results

We evaluated our deep learning model using accuracy metrics. Furthermore, the mean of classification accuracy across all subjects for each method is computed, as is the standard deviation. The accuracy metrics are shown in percentage in this project.

$$Accuracy = \frac{True\ positive\ +\ True\ negative}{True\ positive +\ True\ negative + False\ positive\ and\ False\ negative}$$

Each deep learning model has a different epoch number. DeepConvNet, for example, requires more epoch due to large parameters, whereas ShallowConvNet requires less epoch. The epoch number for a specific deep learning model is kept constant across all experimental methods. Table 6 displays the epoch number for the deep learning models. Across all deep learning models, the batch number is set to 64. Subject 4's data consists of trials for two classes rather than four classes. As a result, the subject 4 data was not taken into account in this project. As a result, for the remaining eight subjects, we evaluate our deep learning models.

Table 6 Deep Learning models and the respective epoch numbers.

Deep Learning Model	Epoch Number
ShallowConvNet	300
DeepConvNet	750
EEGNet	500
ResNet50	750

4.1- ShallowConvNet results for motor imagery classification

Table 7 displays the ShallowConvNet classification accuracy when data is augmented with the sliding window method. For the sliding window, the window size is set to 500 and step size is set to 250 and 500.

Table 7 Accuracy of ShallowConvNet using the augmented data by sliding window method.

Subject	Window=500; Step=250	Window=500; Step=500
S1	86	54
S2	70	66
S3	89	92
S5	39	54
S6	70	38
S7	83	93
S8	84	71
S9	70	71
Mean	73.875	67.375
STD	16.13724618	18.94305376

Table 8 displays the ShallowConvNet classification accuracy when data is augmented with the noise addition method. For the Gaussian noise, the mean is set to zero and standard deviation (STD) is set to 0.001,0.01,0.1,0.02,0.2 and 0.5.

Table 8 Accuracy of ShallowConvNet using the augmented data by noise addition method.

Subject	mean=0, STD=0.001	mean=0, STD=0.01	mean=0, STD=0.02	mean=0, STD=0.1	mean=0, STD=0.2	mean=0, STD=0.5
S1	84	75	75	75	75	92
S2	50	58	58	58	59	67
S3	84	92	92	92	83	84
S5	75	58	75	66	67	50
S6	42	42	50	50	33	33
S7	92	84	84	92	92	83
S8	84	66	84	83	67	75
S9	96	94	98	94	92	91
Mean	75.875	71.125	77	76.25	71	71.875
STD	19.5699296	18.35707337	16.309506	16.8417	19.50092	20.92461

Table 9 displays the ShallowConvNet classification accuracy using the non-augmented data.

Table 9 Accuracy of ShallowConvNet using the non-augmented data

Subject	No Data Augmentation
S1	92
S2	50
S3	83
S5	33
S6	50
S7	83
S8	75
S9	96
Mean	70.75
STD	23.63864874

According to table 7, the mean accuracy was high for data augmented with a window size of 500 and a step size of 250. According to table 8, the mean accuracy was high for data augmented with noise addition, with a mean of 0 and a standard deviation of 0.02. Except for the augmented data with window size 500 and step size 500, the augmented data provided higher mean accuracy than the non-augmented data (table 9).

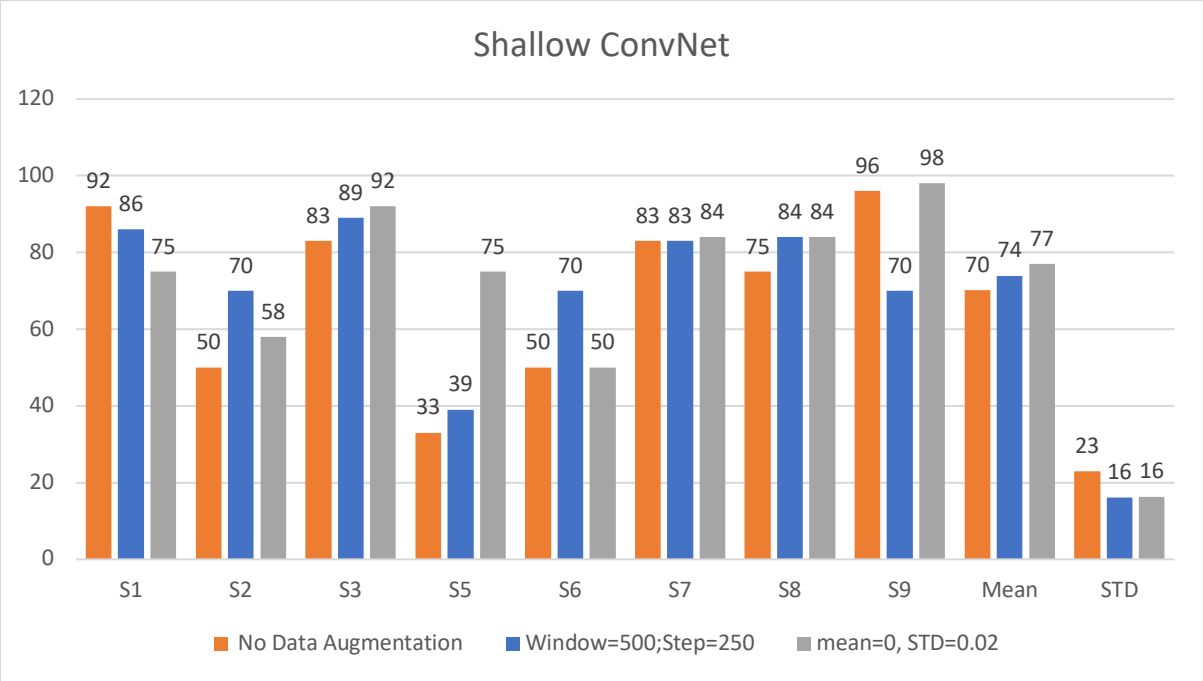

Figure 37 Graph of ShallowConvNet classification accuracy using non-augmented and augmented data. In the graph, the best optimal parameters for the augmented data were chosen from the sliding window and noise addition methods.

4.2- DeepConvNet results for motor imagery classification

Table 10 displays the DeepConvNet classification accuracy when data is augmented with the sliding window method. For the sliding window, the window size is set to 500 and step size is set to 250 and 500.

Table 10 Accuracy of DeepConvNet using the augmented data by sliding window method.

Subject	Window=500; Step=250	Window=500; Step=500
S1	67	63
S2	40	46
S3	73	58
S5	28	25
S6	25	22
S7	73	46
S8	73	67
S9	70	49
Mean	56.125	47
STD	21.33031043	16.44036844

Table 11 displays the DeepConvNet classification accuracy when data is augmented with the noise addition method. For the Gaussian noise, the mean is set to zero and standard deviation (STD) is set to 0.001,0.01,0.1,0.02,0.2 and 0.5.

Table 11 Accuracy of DeepConvNet using the augmented data by noise addition method.

Subject	mean=0, STD=0.001	mean=0, STD=0.01	mean=0, STD=0.02	mean=0, STD=0.1	mean=0, STD=0.2	mean=0, STD=0.5
S1	58	75	75	67	92	75
S2	58	42	42	67	42	43
S3	67	67	84	75	67	75
S5	33	42	50	42	42	33
S6	25	33	33	25	20	25
S7	58	59	68	67	67	58
S8	50	34	65	50	75	50
S9	75	75	84	83	92	92
Mean	53	53.375	62.625	59.5	62.125	56.375
STD	16.6818978	17.73565819	19.13813	19.07878	25.5870363	23.03994978

Table 12 displays the DeepConvNet classification accuracy using the non-augmented data.

Table 12 Accuracy of DeepConvNet using the non-augmented data

Subject	No Data Augmentation
S1	75
S2	59
S3	67
S5	34
S6	25
S7	67
S8	42
S9	83
Mean	56.5
STD	20.63284483

The mean accuracy for data augmented using the window size 500 and step size 250 was high in table 10. Table 11 shows that the mean accuracy was high for data augmented with noise addition, with a mean of 0 and a standard deviation of 0.02. The augmented data using noise addition method with the mean =0 and standard deviation =0.02 provided higher mean accuracy compared to the non-augmented data (table 12) and augmented data using sliding window (table 10).

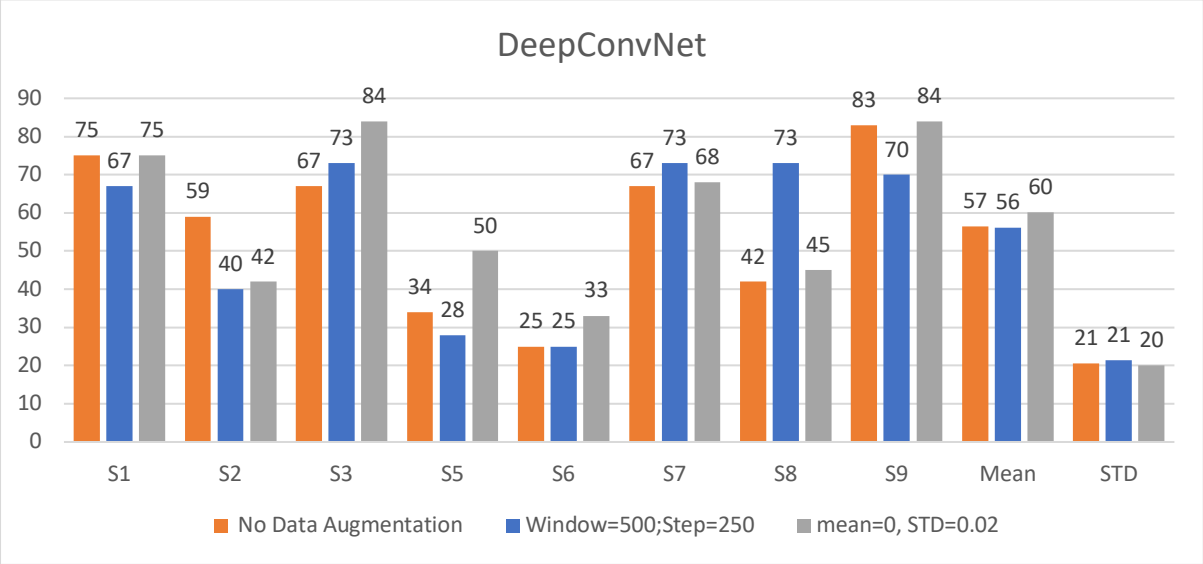

Figure 38 Graph of DeepConvNet classification accuracy using non-augmented and augmented data. In the graph, the best optimal parameters for the augmented data were chosen from the sliding window and noise addition methods.

4.3- EEGNet results for motor imagery classification

Table 13 displays the EEGNet classification accuracy when data is augmented with the sliding window method. For the sliding window, the window size is set to 500 and step size is set to 250 and 500.

Table 13 Accuracy of EEGNet using the augmented data by sliding window method.

Subject	Window=500; Step=250	Window=500; Step=500
S1	66	71
S2	58	50
S3	89	96
S5	59	67
S6	61	46
S7	70	63
S8	75	67
S9	73	71
Mean	68.875	66.375
STD	10.32939633	15.19339612

Table 14 displays the EEGNet classification accuracy when data is augmented with the noise addition method. For the Gaussian noise, the mean is set to zero and standard deviation (STD) is set to 0.001,0.01,0.1,0.02,0.2 and 0.5.

Table 14 Accuracy of EEGNet using the augmented data by noise addition method.

Subject	mean=0, STD=0.001	mean=0, STD=0.01	mean=0, STD=0.02	mean=0, STD=0.1	mean=0, STD=0.2	mean=0, STD=0.5
S1	92	83	75	67	75	75
S2	67	75	77	50	42	74
S3	100	100	100	100	100	84
S5	83	58	67	58	42	42
S6	42	33	62	75	41	42
S7	67	84	92	75	75	75
S8	92	75	84	83	67	92
S9	93	92	94	92	93	92
Mean	79.5	75	81.375	75	66.875	72
STD	19.442222	21.084863	13.458269	16.7161171	23.35709	19.8782

Table 15 displays the EEGNet classification accuracy using the non-augmented data.

Table 15 Accuracy of EEGNet using the non-augmented data

Subject	No Data Augmentation
S1	75
S2	67
S3	100
S5	58
S6	42
S7	83
S8	92
S9	91
Mean	76
STD	19.52288035

.

From the table 13, the mean accuracy was less for data augmented using the window size 500 and step size 250 compared with mean accuracy of non-augmented data from table 15. From the data obtained from table 13,14 and 15, the augmented data using noise addition method with the mean =0 and standard deviation=0.02 provided higher mean accuracy compared with the non-augmented data and augmented data using sliding window.

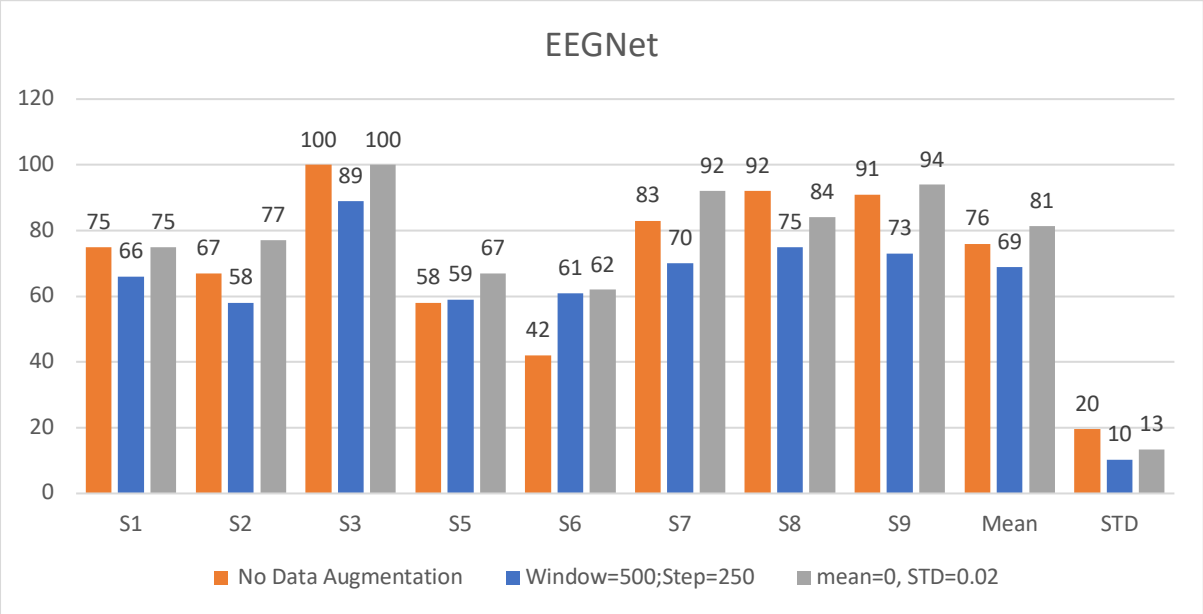

Figure 39 Graph of EEGNet classification accuracy using non-augmented and augmented data. In the graph, the best optimal parameters for the augmented data were chosen from the sliding window and noise addition methods.

4.4- ResNet50 results for motor imagery classification

Table 16 displays the ResNet50 classification accuracy when data is augmented with the sliding window method. For the sliding window, the window size is set to 500 and step size is set to 250 and 500.

Table 16 Accuracy of ResNet50 using the augmented data by sliding window method.

Subject	Window=500; Step=250	Window=500; Step=500
S1	30	67
S2	25	30
S3	52	37
S5	38	22
S6	25	25
S7	36	50
S8	50	50
S9	47	62
Mean	37.875	42.875
STD	10.86853256	16.92367319

Table 17 displays the ResNet50 classification accuracy when data is augmented with the noise addition method. For the Gaussian noise, the mean is set to zero and standard deviation (STD) is set to 0.001,0.01,0.1,0.02,0.2 and 0.5.

Table 17 Accuracy of ResNet50 using the augmented data by noise addition method.

Subject	mean=0, STD=0.001	mean=0, STD=0.01	mean=0, STD=0.02	mean=0, STD=0.1	mean=0, STD=0.2	mean=0, STD=0.5
S1	33	20	51	33	16	17
S2	25	17	33	15	33	33
S3	42	41	66	41	50	58
S5	34	50	50	25	58	33
S6	36	33	26	33	53	25
S7	51	49	50	58	25	41
S8	42	50	55	42	50	58
S9	32	35	25	25	33	33
Mean	36.875	36.875	44.5	34	39.75	37.25
STD	7.93612896	13.130526	14.7841614	13.1692282	15.0783667	14.5871764

Table 18 displays the ResNet50 classification accuracy using the non-augmented data.

Table 18 Accuracy of ResNet50 using the non-augmented data

Subject	No Data Augmentation
S1	41
S2	66
S3	48
S5	41
S6	33
S7	36
S8	48
S9	42
Mean	44.375
STD	10.155048

Table 16 shows that the mean accuracy of data augmented using sliding window methods was lower than the mean accuracy of non-augmented data in table 15. From the data in tables 17 and 18, the augmented data using the noise addition method with a mean of 0 and a standard deviation of 0.02 provided roughly comparable mean classification accuracy to the non-augmented data.

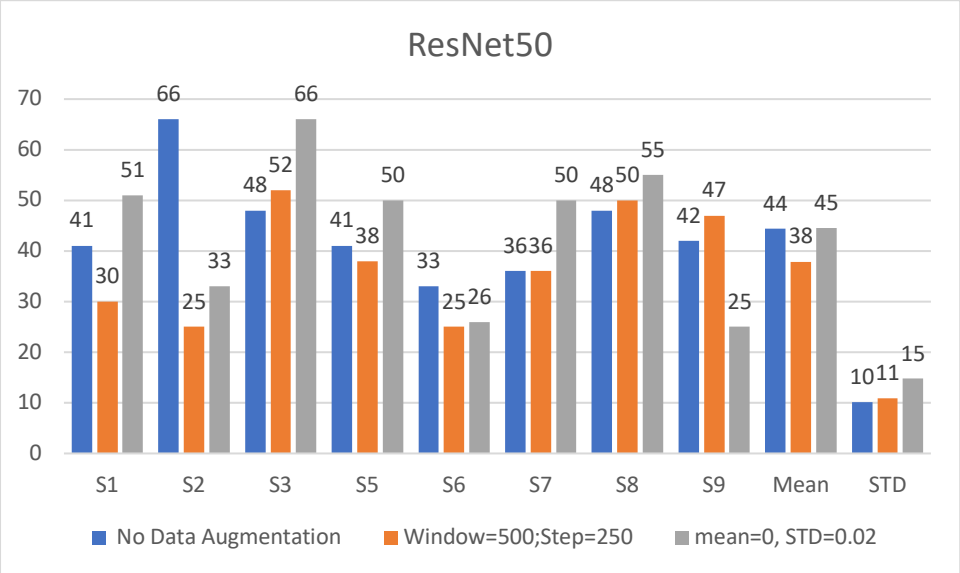

Figure 40 Graph of ResNet50 classification accuracy using non-augmented and augmented data. In the graph, the best optimal parameters for the augmented data were chosen from the sliding window and noise addition methods.

4.5- Results of deep learning models and data augmentation methods.

Table 19 and Figure 41 displays the mean classification accuracy of all deep models using augmented data and non-augmented data. For the noise addition method, the optimal value was chosen to reduce the complexity of the table.

Table 19 Mean classification accuracy across all the deep learning models in conjunction with data augmentation methods and no data augmentation.

Deep Learning model	Window=500; Step=250	Window=500; Step=500	Noise Addition (Mean=0 and Standard deviation=0.002)	No data augmentation
ShallowConvNet	74	67	77	70
DeepConvNet	56	47	63	57
EEGNet	69	66	81	76
ResNet50	38	43	44	44

.

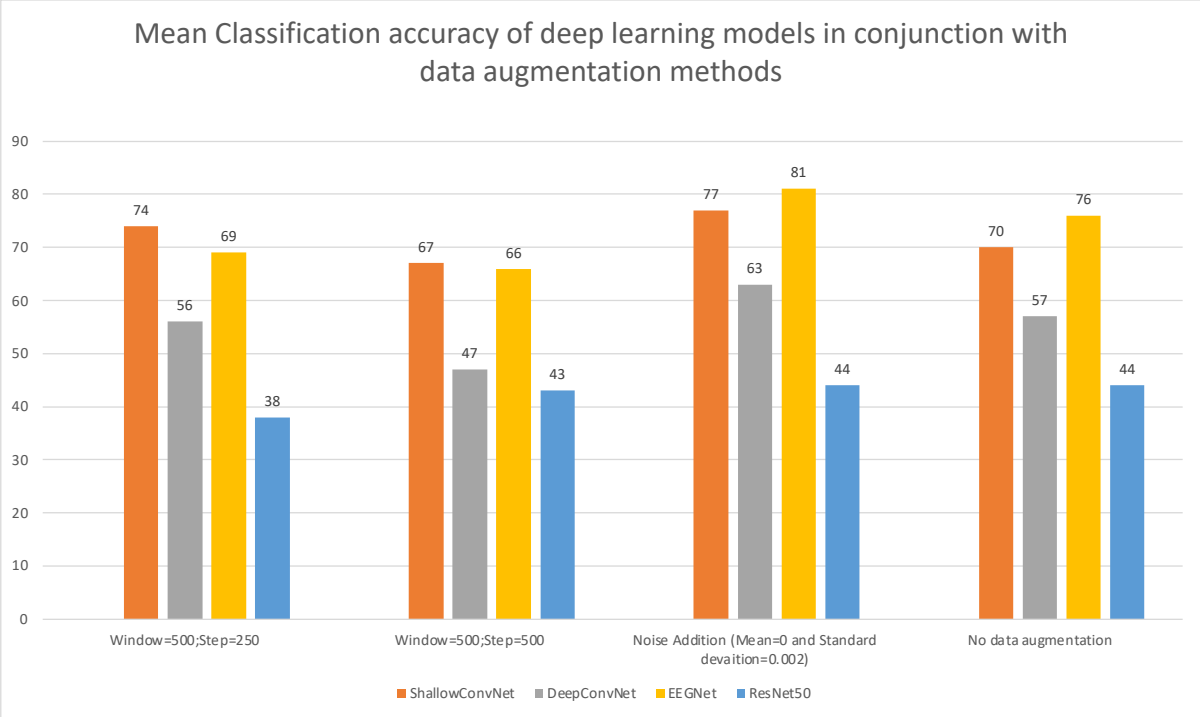

Figure 41 Graph represents mean classification accuracy across all the deep learning models in conjunction with data augmentation methods and no data augmentation.

5-Discussion

From the calculated results shown in Section 4, it can be seen that the noise addition method improved the classification accuracy across all deep learning models such as ShallowConvNet, EEGNet, DeepConvNet and ResNet50. The classification accuracy increased by 7%,6%,5.5% and 0.2% for the ShallowConvNet,EEGNet, DeepConvNet and ResNet50 respectively using the augmented data by noise addition method compared to the non-augmented data. The samples generated using gaussian noise with the highest standard deviation value of 0.5 produced worse mean classification accuracy across all deep learning models compared to data augmented using other standard deviation values used in gaussian noises (see table 8,11,14, and17). The samples generated with the lowest standard deviation value of 0.001 produced satisfactory mean classification accuracy across all deep learning models when compared to other standard deviation values used in gaussian noise (see table 8,11,14, and17). The samples generated with a standard deviation value of 0.02, as shown in tables 8, 11, 14, and 17, produced the highest mean classification accuracy across all deep learning models when compared to data augmented with other standard deviation values used in gaussian noises. Furthermore, a standard deviation value of 0.02 produced the lowest standard deviation value for EEGNet and ShallowConvNet classification results across all subjects. While the standard deviation value for the Gaussian noise appears to be low, the generated samples appear to be similar to the original samples, producing satisfactory classification results. The generated sample with a higher standard deviation, on the other hand, varies significantly more than the original samples producing the worst classification results. Moreover, the standard deviation values higher than the lowest value and lower than the highest value generated samples with little variation compared to the original samples and produced the highest classification accuracy across all the deep learning models used in this project. As a result, the optimal Gaussian noise's standard deviation value is 0.002 for augmenting the data.

Tables 7, 10, 13 and 16 show that data augmented with window size 500 and step size 500 produced worse mean classification accuracy across ShallowConvNet, EEGNet, and DeepConvNet, with the exception of ResNet-50, when compared to data augmented with window size 500 and step size 250. Despite the fact that the ResNet50 classification accuracy for the non-overlapping window is higher than the classification accuracy for the overlapping window, the difference is 3%, which is insignificant. The preceding fact demonstrates that the generated samples with no overlap between the windows had a negative impact on the classification result. Tables 7, 10, 13 and 16 show that data augmented with window size 500 and step size 250 produced lower mean classification accuracy across EEGNet, DeepConvNet,

and ResNet-50 except for ShallowConvNet when compared to non-augmented data. As a result, the sliding window method had a negative effect on classification accuracy across the deep learning models such as EEGNet, DeepConvNet, and ResNet-50. In addition, the sliding window method with overlap had a positive effect on ShallowConvNet's classification accuracy.

From table 9,12,15 and 18, without the data augmentation methods, the EEGNet and ShallowConvNet achieved mean classification accuracy of more than 70%, while the ResNet50 and DeepConvNet achieved mean classification accuracy of less than 60%. The ResNet50 model performed the worst, with a mean classification accuracy of 44%, while the EEGNet model outperformed all of the deep learning models used in this project, with a mean classification accuracy of 76%. After implementing the noise addition method, the DeepConvNet accuracy increased by 6%, whereas the ResNet50 accuracy had no effect after implementing both data augmentation methods. The ResNet50 does not appear to be able to learn features from the raw EEG signal and instead requires handcrafted features for better classification accuracy. The study by Panachakel and Ganesan (2021) demonstrates the aforementioned statement by achieving an average classification accuracy of 90% while using ResNet50 and handcrafted features such as Mean phase coherence (MPC) and Magnitude-Squared Coherence (MSC). DeepConvNet's accuracy appears to be unsatisfactory due to a lack of data, as the accuracy increased after using data augmentation methods. However, increasing the number of samples generated with the existing small data does not improve DeepConvNet classification accuracy significantly compared to ShallowConvNet and EEGNet. To outperform the ShallowConvNet and EEGNet, Choo et al (2020) stated that the DeepConvNet required new data augmentation methods rather than traditional data augmentation methods such as sliding window and noise addition or hyper tunning methods. After implementing the noise addition method, the mean classification accuracy of ShallowConvNet increased by 7%. The noise addition method with a standard deviation of 0.02 and a mean of 0 yielded the best mean classification accuracy of 77% among all tests using the ShallowConvNet. Without the use of augmented data, EEGNet nearly outperformed the best ShallowConvNet classification accuracy, achieving a mean classification accuracy of 76%. This demonstrated that the EEGNet can learn features effectively even with limited data. Following the implementation of the noise addition method, the mean classification accuracy of EEGNet increased by 5%. The noise addition method with a standard deviation of 0.02 and a mean of 0 and EEGNet produced the best mean classification accuracy of 81.4% across all tests in our experiment. The preceding

observations show that the best combination for classifying motor imagery tasks is augmented data using the noise addition method in conjunction with the EEGNet deep learning model.

6-Conclusion

We reviewed various research papers on EEG data pre-processing methods in this paper. We discovered that filtering EEG data between 4-40Hz removed eye-related artefacts while capturing higher brain activity during motor imagery. Furthermore, we investigated the channels to be used for classification of motor imagery task and concluded that using all of them provides better spatial resolution and captures information flow between multiple cortical regions. After reviewing various research papers, we proposed using three deep learning models to validate the motor imagery classification accuracy: ShallowConvNet, EEGNet, and DeepConvNet, due to their ability to learn features automatically from raw EEG data. We proposed using data augmentation and transfer learning to address the issue of data scarcity. The transfer learning method was implemented using the ResNet50 pre-trained CNN model. As a result, we validated four deep learning models in this project. We used a public dataset of four motor imagery classification tasks to validate deep learning models. Furthermore, we investigated various data augmentation methods before settling on noise addition and sliding window. To validate the efficacy of data augmentation methods, augmented and non-augmented data were used on all deep learning models in a comparison study. Following the implementation of deep learning models with augmented and non-augmented data, we concluded that the noise addition method improved classification accuracy across all deep learning models, whereas the sliding window method did not. The ResNet50 performed the worst with both augmented and non-augmented data, demonstrating that it was ineffective when learning features automatically from raw EEG data. ShallowConvNet, EEGNet, and DeepConvNet, on the other hand, were effective in learning features from raw EEG data. EEGNet proved to be an effective deep learning model for motor imagery classification compared to other deep learning models used in this project. In conclusion, from the findings in this project, we demonstrated that the noise addition method is the best data augmentation method for improving classification accuracy with limited data, and that EEGNet is an effective deep learning model for motor imagery task classification.

7-Future Work

After determining that the ResNet50 is incapable of learning features from raw data, we will investigate manual feature extraction methods in the future to see if the transfer learning methodology can address the data scarcity issue. Panachakel and Ganesan (2021) used Mean Phase Coherence and Magnitude-Squared Coherence for manual feature extraction. Mean Phase Coherence calculates the phase difference between two EEG channels and returns a value between 0 and 1, where 0 indicates that the phase difference is random and 1 indicates that the phase difference is related (Panachakel and Ganesan, 2021). The Mean Phase Coherence is widely used in sleep studies to extract features. Magnitude-Squared Coherence measures the linear relationship between two channels over all frequencies (Panachakel and Ganesan, 2021). Furthermore, various pre-trained CNNs can be used in future studies to implement transfer learning. VGGNet, GoogleNet, and ResNet-50 are three pre-trained CNN models proposed by Xu et al. (2019) for transfer learning in the field of BCI. As a result, we can evaluate whether all pre-trained CNNs are incapable of learning features from raw EEG data. In this project, the use of sliding window methods had a negative impact on the classification of motor imagery tasks. As a result, we are considering using GAN to augment data in future studies. The reason for this is that GAN was predominantly used in the EEG-based BCI application (Lashgari et al, 2020). We believe the GAN will work because it generates new samples in a manner similar to the noise addition method, which has been shown to improve classification accuracy in this project.

Reference

Aggarwal, S. and Chugh, N. (2019) "Signal processing techniques for motor imagery brain computer interface: A review," Array, 1–2(100003), p. 100003. doi: 10.1016/j.array.2019.100003.

Altuwaijri, G.A. and Muhammad, G. (2022). A Multibranch of Convolutional Neural Network Models for Electroencephalogram-Based Motor Imagery Classification. Biosensors, 12(1), p.22. doi:10.3390/bios12010022.

Al-Saegh, A., Dawwd, S.A. and Abdul-Jabbar, J.M. (2021). Deep learning for motor imagery EEG-based classification: A review. Biomedical Signal Processing and Control, 63, p.102172. doi:10.1016/j.bspc.2020.102172.

Ang, K.K., Chin, Z.Y., Zhang, H. and Guan, C. (2008). Filter Bank Common Spatial Pattern (FBCSP) in Brain-Computer Interface. [online] IEEE Xplore. doi:10.1109/IJCNN.2008.4634130.

Bitbrain. (2020). All about EEG artifacts and filtering tools. [online] Available at: https://www.bitbrain.com/blog/eeg-artifacts.

Choo, S., Park, H., Jung, J., Nam, C. (2020). Effectiveness of Interpolated Conditional Generative Adversarial Networks (icGANs) on EEG Data Augmentation.

Clayton, J, Wellington, S, Valentini-Botinhao, C & Watts, O 2020, Decoding imagined, heard, and spoken speech: Classification and regression of EEG using a 14-channel dry-contact mobile headset. in Proceedings Interspeech 2020. vol. 2020-October, International Speech Communication Association, pp. 4886-4890, Interspeech 2020, Virtual Conference, China, 25/10/20. https://doi.org/10.21437/Interspeech.2020-2745

Dai, M., Zheng, D., Na, R., Wang, S. and Zhang, S. (2019). EEG Classification of Motor Imagery Using a Novel Deep Learning Framework. Sensors, 19(3), p.551. doi:10.3390/s19030551.

Frølich, L., Winkler, I., Müller, K.-R. and Samek, W. (2015). Investigating effects of different artefact types on motor imagery BCI. [online] IEEE Xplore. doi:10.1109/EMBC.2015.7318764.

Nithianandam, G. (2022). Decoding Imagined speech using Electroencephalogram. Bath: University of Bath.

Lashgari, E., Liang, D. and Maoz, U. (2020) 'Data augmentation for deep-learning-based electroencephalography', Journal of Neuroscience Methods, 346, σ. 108885. doi: 10.1016/j.jneumeth.2020.108885.

Lawhern, V.J., Solon, A.J., Waytowich, N.R., Gordon, S.M., Hung, C.P. and Lance, B.J. (2018). EEGNet: a compact convolutional neural network for EEG-based brain–computer interfaces. Journal of Neural Engineering, [online] 15(5), p.056013. doi:10.1088/1741-2552/aace8c

Leeb, R., Brunner, C., Müller-Putz, G., Schlögl, A. and Pfurtscheller, G. (2008). BCI Competition 2008 -Graz data set B Experimental paradigm. [online] Available at: https://www.bbci.de/competition/iv/desc_2b.pdf

Liu, R., Zhang, Z., Duan, F., Zhou, X. and Meng, Z. (2017). Identification of Anisomerous Motor Imagery EEG Signals Based on Complex Algorithms. Computational Intelligence and Neuroscience, 2017, pp.1–12. doi:10.1155/2017/2727856.

Mulani, S. (2022). Using StandardScaler() Function to Standardize Python Data |DigitalOcean.[Online]Available at: https://www.digitalocean.com/community/tutorials/standardscaler-function-in-python

Nguyen, C.H., Karavas, G.K. and Artemiadis, P. (2017). Inferring imagined speech using EEG signals: a new approach using Riemannian manifold features. Journal of Neural Engineering, 15(1), p.016002

Panachakel, J.T. and Ramakrishnan, A.G. (2021). Decoding Covert Speech From EEG-A Comprehensive Review. Frontiers in Neuroscience, 15.

Panachakel, J. T. and Ganesan, R. A. (2021) 'Decoding Imagined Speech From EEG Using Transfer Learning', IEEE Access, 9, σσ. 135371–135383. doi: 10.1109/ACCESS.2021.3116196.

Roy, A.M. (2022). A CNN model with feature integration for MI EEG subject classification in BMI. doi:10.1101/2022.01.05.475058

Schirrmeister, R.T., Springenberg, J.T., Fiederer, L.D.J., Glasstetter, M., Eggensperger, K., Tangermann, M., Hutter, F., Burgard, W. and Ball, T. (2017). Deep learning with convolutional neural networks for EEG decoding and visualization. Human Brain Mapping, [online] 38(11), pp.5391–5420. doi:10.1002/hbm.23730.

vlawhern (2022). Introduction. [online] GitHub. Available at: https://github.com/vlawhern/arl-eegmodels/blob/master/EEGModels.py

Wang, F., Zhong, Sh., Peng, J., Jiang, J., Liu, Y. (2018). Data Augmentation for EEG-Based Emotion Recognition with Deep Convolutional Neural Networks. In: , et al. MultiMedia Modeling. MMM 2018. Lecture Notes in Computer Science(), vol 10705. Springer, Cham. https://doi.org/10.1007/978-3-319-73600-6_8

Xia, X. and Hu, L. (2019). EEG: Neural Basis and Measurement. EEG Signal Processing and Feature Extraction, pp.7–21.