FEATURE EXTRACTION FOR EEG MOTOR IMAGERY SIGNALS USING A DEEP NEURAL NETWORK

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

RIDVAN SOYSAL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

AUGUST 2023

Approval of the thesis:

FEATURE EXTRACTION FOR EEG MOTOR IMAGERY SIGNALS USING A DEEP NEURAL NETWORK

submitted by **RIDVAN SOYSAL** in partial fulfillment of the requirements for the degree of **Master of Science** i**n Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of the Department, **Electrical and Electronics Engineering** _____

Prof. Dr. Uğur Halıcı
Supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Engineering, METU _____

Prof. Dr. Uğur Halıcı
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Elif Vural
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Mehmet Dikmen
Computer Engineering, Başkent University _____

Assist. Prof. Dr. Serkan Sarıtaş
Electrical and Electronics Engineering, METU _____

Date: 22.08.2023

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name Last name: Rıdvan Soysal

Signature:

ABSTRACT

FEATURE EXTRACTION FOR EEG MOTOR IMAGERY SIGNALS USING A DEEP NEURAL NETWORK

Soysal, Rıdvan
Master of Science, Electrical and Electronic Engineering
Supervisor: Prof. Dr. Uğur Halıcı

August 2023, 103 pages

Compared to traditional machine learning methods, deep learning methods generally have better performance, and they are more able to handle complex data. Moreover, these models learn the features directly from the raw data, eliminating the need for additional feature extraction step. However, in order to benefit from these advantages, deep learning methods need high amount of data.

In this study we examined the use of deep learning methods in the field of EEG motor imagery signal (MI) classification. Although in recent years, many researchers have been applying deep learning methods in this area, we notice that EEG MI signal datasets that are highly used in these researches have insufficient amount of data for deep learning.

In this thesis, in order to benefit more from advantages of deep learning methods on EEG MI signal classification we looked for a solution to combine the datasets collected in different studies which cannot be combined directly due to variations in the protocols used in collecting data. After combining available datasets each having little amount of data, we created a larger dataset and used this mega-dataset to train

a convolutional autoencoder (CAE) based network that can be used as a deep feature extractor (DFE) for MI signals. Afterwards, trained DFE network is tested as a feature extractor for small datasets. Our experimental results show that using such DFE network improves the performance of EEG MI signal classification on these datasets.

Keywords: EEG MI Signal Classification, Convolutional Autoencoder, Deep Clustering, Feature Extractor, Brain Computer Interface (BCI)

ÖZ

DERİN SİNİR AĞI KULLANARAK MOTOR İMGELEME EEG SİNYALLERİ İÇİN ÖZNİTELİK ÇIKARIMI

Soysal, Rıdvan
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği
Tez Yöneticisi: Prof. Dr. Uğur Halıcı

Ağustos 2023, 103 sayfa

Derin öğrenme yöntemleri, geleneksel makine öğrenmesi yöntemlerine göre genellikle daha iyi sonuçlar verebilmekte, daha karmaşık yapıdaki veriler ile çalışmaya olanak sağlamaktadır. Ayrıca ham veri üzerine doğrudan çalışılabildiği için ilave bir öznitelik çıkarımı adımı ihtiyacı da bulunmamaktadır. Fakat derin öğrenmenin bu avantajlarından faydalanabilmek için yüksek miktarda veriye ihtiyaç bulunmaktadır.

Bu çalışmada, EEG motor imgeleme (MI) sınıflandırma alanında derin öğrenme yöntemlerinin kullanımını inceledik. Geçtiğimiz yıllarda, birçok araştırmacı bu konuyla ilgili derin öğrenme yöntemlerini uygulasa da, çalışmalarda sıkça kullanılan EEG MI sinyal veri setlerinin derin öğrenme yöntemleri için yeterli miktarda veri içermediğini gözlemledik.

Bu tezde, EEG motor imgeleme sinyalleri sınıflandırması üzerinde, derin öğrenme yöntemlerinin avantajlarından daha fazla faydalanabilmek için, veri toplama protokolleri birbirinden farklı olduğundan beraber kullanılamayan veri setlerinin birleştirilmesi için çözüm araştırdık. Mevcutta erişilebilir olan fakat az miktarda veri içeren sinyal veri setlerini birleştirerek daha büyük bir MI veri seti elde ettik ve bu

büyük seti, MI sinyalleri için bir öznitelik çıkarıcı olarak kullanmak üzere tasarladığımız evrişimsel otokodlayıcı (CAE) tabanlı derin öznitelik çıkarıcı (DFE) sinir ağını eğitmek için kullandık. Daha sonra, eğittiğimiz DFE sinir ağını, daha küçük veri setlerinde öznitelik çıkarıcı olarak kullandık. Deneysel sonuçlarımız, kullandığınız DFE ağının EEG MI sinyal sınıflandırma başarımını artırdığını göstermektedir.

Anahtar Kelimeler: EEG Mİ Sinyal Sınıflandırma, Evrişimsel Otokodlayıcı, Derin Kümeleme, Öznitelik Çıkarıcı, Beyin Bilgisayar Arayüzü (BCI)

to my son…

ACKNOWLEDGMENTS

I would like to express my special thanks and sincere gratitude to my supervisor Prof. Dr. Uğur Halıcı for her guidance and encouragement throughout my research. It was a great honor for me to work with her.

I would like to thank my company ASELSAN A.Ş. for providing me the opportunity of continuing my graduate education.

The numerical calculations reported in this thesis were partially performed at TÜBİTAK ULAKBİM, High Performance and Grid Computing Center (TRUBA resources) through Neuroscience and Neurotechnology Center of Excellence (NÖROM).

I also must express my gratitude to my beloved wife and my parents. They have shown patience and have supported me in this work, as they always do in my life.

Finally, praises and thanks be to the God for giving me ability, knowledge and strength to complete this academic journey. Without His continuous blessings, it would not be possible.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

[AE] Autoencoder

[BCI] Brain Computer Interface

[CAE] Convolutional Autoencoder

[CAHL] Clustering Assignment Hardening Loss

[CE] Cross Entropy

[CNN] Convolutional Neural Network

[CSP] Common Spatial Pattern

[DBN] Deep Belief Network

[DFE] Deep Feature Extractor

[DNN] Deep Neural Network

[EDF] European Data Format

[EEG] Electroencephalography

[ERD] Event Related Desynchronization

[ERS] Event Related Synchronization

[FC] Fully Connected

[FFT] Fast Fourier Transform

[GAN] Generative Adversarial Network

[GDF] General Data Format

[KL] Kullback-Leibler

[MI] Motor Imagery

[MLP] Multilayer Perceptron

[MSE] Mean Squared Error

[PCA] Principal Component Analysis

[RBM] Restricted Boltzmann Machine

[RELU] Rectified Linear Unit

[SAE] Stacked Autoencoder

[SF] Sampling Frequency

[STFT] Short Time Fourier Transform

[T-SNE] T-Distributed Stochastic Neighbor Embedding

[VAE] Variational Autoencoder

[WT] Wavelet Transform

CHAPTER 1

INTRODUCTION

1.1 Motivation of the Thesis

A Brain-Computer Interface (BCI) system helps individuals to communicate with a machine or other external devices using their brain signals. Researches into BCI began in the 1970s and since then numerous organizations around the world has been looking into this issue as stated in [2]. Recently, BCI has numerous potential uses in the fields of medicine, rehabilitation, advertisement, entertainment and gaming among others [2], [3].

A complete BCI system generally consists of four main parts: collecting and amplifying brain signals, recognizing and classifying these signals, sending commands to controlled equipment, and getting feedbacks from the equipment [4]. The study in [3] summarizes different methods to acquire brain signals which is the first component of a BCI system. Among different methods, electroencephalography (EEG) is the most widely used and convenient approach to collect brain signals. Within EEG signals, motor imagery (MI) signals are the most popular brain signal patterns that are used in BCI applications. This work is interested in EEG MI signals and mainly focuses on identifying and categorizing these signals. Controlling an equipment or getting feedbacks are beyond the scope of the research.

Developments in machine learning had a significant impact on the classification of EEG signals. Many techniques in machine learning applied for EEG signal classification which are reviewed in [4], until the advent of deep learning to this research area. [1] is one of the first researches that apply deep neural networks to the EEG MI field with a significant improvement in performance. It uses a convolutional neural network (CNN) cascaded with a stacked autoencoder (SAE). After that many

deep learning applications have been tried with EEG MI signals. Number of these applications and articles about this topic are growing rapidly over the last ten years. [29] and [30] make comprehensive reviews on these articles.

Despite the variety of deep learning methods applied, datasets used for EEG MI signal classification are limited to [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55] and [56]. Moreover, each of these datasets have limited amount of data. Besides, one cannot simply use datasets together due to different signal formats (sampling rates, used channels, bits per samples, file formats, etc...). This constraint lowers classification performance of deep learning applications owing to the requirement of large datasets in these applications. In order to overcome this problem, either larger datasets should be formed or need for large data should be reduced by using simpler neural networks.

Current situation of EEG MI datasets motivated us to focus on combining all available datasets to create a larger dataset and training a deep feature extractor (DFE) network using this mega-dataset. Once DFE network is trained, it can be used to extract features for smaller datasets, and with the help of these deep features, need for training deeper networks can be disappeared. One can use and train simpler networks such as MLP (Multi-Layer Perceptron) or a few layered CNN concatenated to our DFE network, in order to classify extracted features.

1.2 Scope of the Thesis

In this thesis, firstly, literature surveys about publicly available EEG MI datasets, deep learning methods used for EEG classification and convolutional autoencoders/deep clustering methods used with various datasets in various applications are presented.

In addition, challenges of combining all publicly available EEG signal datasets to obtain one unified image dataset are shown. Critical properties of EEG signals that are paid attention to, necessary signal processing steps, conversion of time series

signals to images, and further processing on images, such as whitening and data augmentation are explained.

After the explanation of the establishment of the combined dataset, proposed DFE networks that use this combined dataset are described. Six different DFE networks that can be used for feature extraction of EEG signals are presented along with the feature classifiers that use features extracted by DFEs and outputs classification results.

Lastly, experiments performed with DFEs and feature classifiers are shown. Details of training processes are given and classification results are shared to validate the quality of extracted features.

1.3 Contributions of the Thesis

Deep learning applications in EEG MI classification suffers from insufficient amount of available data. Although, it seems that there exist different datasets, as reviewed in [30], 60 percent of academic studies used only two of datasets: BCI Competition IV-2a [50] and BCI Competition IV-2b [51] where amount of data is quite limited for deep learning.

In order to overcome this problem, in this study we propose combining all available EEG MI datasets to train a deep feature extractor (DFE) network. This approach brings its own difficulties into the surface, due to dissimilarities in existing datasets. Each dataset is acquired using different equipment, with different acquisition parameters, adopting different experiment scenarios, and stored in different formats. Our first contribution is overcoming these difficulties and creating a combined dataset of all datasets.

Secondly we make our contribution by providing six different DFE networks and feature classifiers that are used to classify extracted features. Two of these networks seem to increase the classification performance.

1.4 Organization of the Thesis

This thesis work has 5 main chapters. The first chapter is Introduction, which includes motivation, scope and contributions of the thesis.

The second chapter contains literature survey that we have conducted, which includes three categories. First, we reviewed the existing publicly available datasets and their important properties. Second, we examined the existing EEG MI signal classification applications that use deep learning methods. We were also interested in, which datasets, which input formats and which deep learning methods are used in these works. Lastly, considering that deep clustering can be used for feature extraction we did researches about existing clustering algorithms based on deep learning.

Rest of the chapters are related to our designs and experiments. In chapter 3, how different EEG MI signal datasets are converted into one big unified dataset is explained. Signal processing, time signal to time-frequency image conversion, whitening application on images and data augmentation constitute the contents of this chapter.

In chapter 4, structure and details of six different DFE networks designed in this thesis work are given. Beside these DFEs, feature classifiers specific to each DFE, which use features extracted from DFEs and outputs classification results are explained in this chapter. In addition, a CNN structure which is used for performance comparison is also presented.

In the 5^{th} chapter, experimental works are shown and details of training processes are given. Then, results of experimental works are shared, analysis of successful and unsuccessful trials are presented and our results are compared to state of the art methods.

The last chapter concludes the study, where summary of the works are given, and discussions are provided.

CHAPTER 2

LITERATURE REVIEW

In this chapter, the current literature on EEG MI signal classification and deep clustering is presented. In the first section, we give the details of publicly available EEG MI datasets that are used by researchers. We summarize similarities and differences of these datasets. In the second section, we focus on deep learning applications on EEG signal classification. We examine scientific studies according to datasets used, frequency bands, input formulations and deep learning architectures. In the last section, we review previous works on deep clustering. We analyze existing deep clustering architectures and applications made on different datasets.

2.1 Publicly Available EEG MI Datasets

The Graz BCI dataset, published in 2001 by Gernot Müller-Putz and his group at the Graz University of Technology in Austria, was the first publicly accessible EEG signal dataset. In this dataset, brain signals of nine subjects were recorded while subjects are engaged in a variety of motor imagery activities, like imagining moving their left and right hands. The set is currently unavailable on the web.

In 2008, the BCI Competition IV dataset [50], [51] was released. This dataset was created for using in a competition to find the most effective machine learning classification algorithms for MI signals. Since then, with growing attention in machine learning and developments in deep learning, several other publicly available EEG MI signal datasets have been released. We give the list of publicly available sets in Table 2.1.

Table 2.1 List of publicly available datasets

Name	*Original Dataset Name*	*Year*	*Free Access*
Set-1	EEG datasets for motor imagery brain–computer interface [45]	2017	Available
Set-2	EEG Motor Movement/Imagery Dataset [46]	2009	Available
Set-3	Multi-channel EEG recordings during 3,936 grasp and lift trials with varying weight and friction [47]	2014	Available
Set-4	A large electroencephalographic motor imagery dataset for electroencephalographic brain computer interfaces [48]	2018	Available
Set-5	BCI Competition IV – 1 [49]	2007	Available
Set-6	BCI Competition IV – 2a [50]	2008	Available
Set-7	BCI Competition IV – 2b [51]	2008	Available
Set-8	High Gamma Dataset [52]	2017	Available
Set-9	Project BCI - EEG motor activity data set [53]	2007	Available*
Set-10	Planning Relax Data Set [54]	2012	Not Available**
Set-11	EEG dataset of 7-day Motor Imagery BCI [55]	2020	Available
Set-12	Motor Imagery Brain-Computer Interfaces: Random Forests vs Regularized LDA - Non-linear Beats Linear [56]	2014	Available

* Dataset available, but it is too small to use. Only two trials are available for only one person per each class.

** Only extracted features from EEG signals are available, no raw data available.

Sets that are included in the scope of this research does not have the same signal properties. This is because of using different BCI devices during the acquisition, and/or creating different experimental setups. As a result, using these datasets together without raw signal processing is impossible. In this part of the literature survey, we examine critical properties of these datasets.

2.1.1 Motor Imagery Labels (Classes)

Motor imagery (MI) is the mental representation or simulation of a movement without actually doing it. In each dataset, people who are making imaginations mentally, imagines different motor movements. For instance in Table 2.2, set-6 includes left/right hand imagery movements, feet movements and tongue movements, whereas set-8 includes only left/right hand imagery movements. In addition, some sets include, some real movements besides imagery ones. For example, set-1 contains left/right hand real movements as well as imagery ones.

“Labels” column in Table 2.2, gives which MI movements are included in the dataset. In the table, left, right, rest, fists, feet, left_leg, right_leg and tongue means imagining left hand movement, right hand movement, resting, making fist, feet movement, left leg movement and right leg movement, respectively. Moreover, thumb, index, middle, ring, pinkie labels are MI movements of the specified finger. Lastly 165g, 330g and 660g labels are used for object lifting motor imageries with corresponding weights. If a label starts with “real_” suffix, it shows that the movement is not imagery, it is really performed.

Table 2.2 Summary of dataset properties

Set	*Labels*	*# of Persons*	*Length (secs)*	*# of Channels*	*Sampling Freq.*	*File For.*
Set-1	left, right, real_left, real_right, rest	52	1040	64	512	MAT
Set-2	left, right, fists, feet, real_left, real_right, real_fists, real_feet, rest	109	1560	64	160	EDF
Set-3	real_165g, real_330g, real_660g	12	3500	32	500	MAT
Set-4	left, right, rest, thumb, index, middle, ring, pinkie, left_leg, right_leg, tongue	13	22450	22	200/1000	MAT
Set-5	left, right, feet	7	2960	59	1000	MAT
Set-6	left, right, feet, tongue	9	5350	22	250	GDF
Set-7	left, right	9	10500	3	250	GDF
Set-8	left, right, feet, rest	14	3775	128	500	EDF
Set-9	real_left, real_right, left, right, real_left_leg, real_right_leg	1	420	19	500	MAT
Set-10	-	1	-	8	256	-
Set-11	left, right, feet, rest	20	16236	26	500	NPZ
Set-12	right, feet	14	1765	15	512	MAT

2.1.2 Number of Persons and Signals Lengths

"# of Persons" and "Length (secs)" columns in Table 2.2 give information about amount of data. First one shows how many different people are included in preparing the dataset. In other words, each dataset includes data from a certain number of subjects, which is given on the "# of Persons" column. Smallest datasets, set-9 and set-10 only includes data from 1 subject, whereas, the largest dataset set-2 includes data from 109 subjects.

The next column defines the length of the signal in seconds. It shows how many seconds raw signal is available for each person.

2.1.3 Acquisition Scenarios

Researchers that built these datasets, have different acquisition setups. This leads to changes in scenarios during signal acquisition. However, in order to use these datasets together, we focus on some common properties.

Since each set consists of trials, we make our analysis trial based. A trial contains only a single MI event. Generally during a trial, signal acquisition starts before MI event and ends after MI event finishes. Therefore, we define each trial duration in three parts: pre MI event duration, MI event duration, post MI event duration. These durations differ between datasets. In addition, in some sets, MI cue that tells the subject which MI task he/she will perform, is given in pre MI event duration, whereas, in others it is given directly at the start of MI event.

We illustrate how these durations change between datasets in Figure 2.1. The time that cue is displayed is accepted as 0 point, in this graph. In addition to that, some of durations are not constant and change over trials. In that case, we take average time of different trials and demonstrate the mean value in the graph. Values with tilde (~) marks are results of averages.

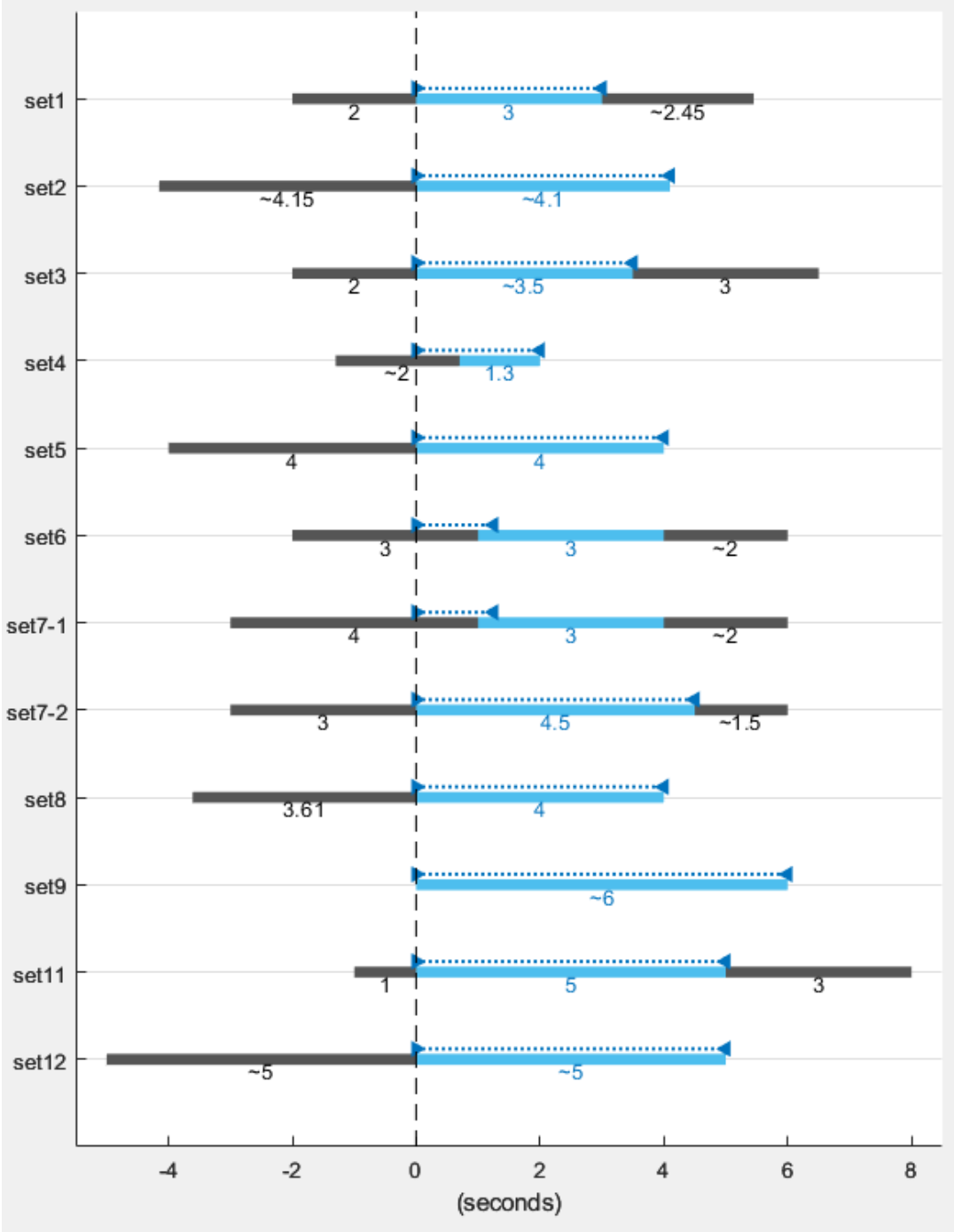

Figure 2.1. Acquisition scenarios of different datasets [45] … [56]

Figure notes: (blue bars: MI event durations, gray bars: pre/post MI event durations, right arrows: start times of cues, left arrows: end times of cues, dotted blue lines: active cue durations)

2.1.4 Number of Channels (Electrodes)

Since datasets are collected with different systems, number of channels also varies between datasets. Minimum number of channels is observed in Set-7. In this set, only three electrodes exists which are c3, c4 and cz. On the other hand, set-8 contains the maximum number of channels, where 128 electrodes are used to collect EEG signals. In Figure 2.2, received from [57], placement of 128 electrodes is shown. Other sets, includes 8 to 64 channels. In these sets, subset of given 128 electrodes are used, which are approximately equally spaced over the skull.

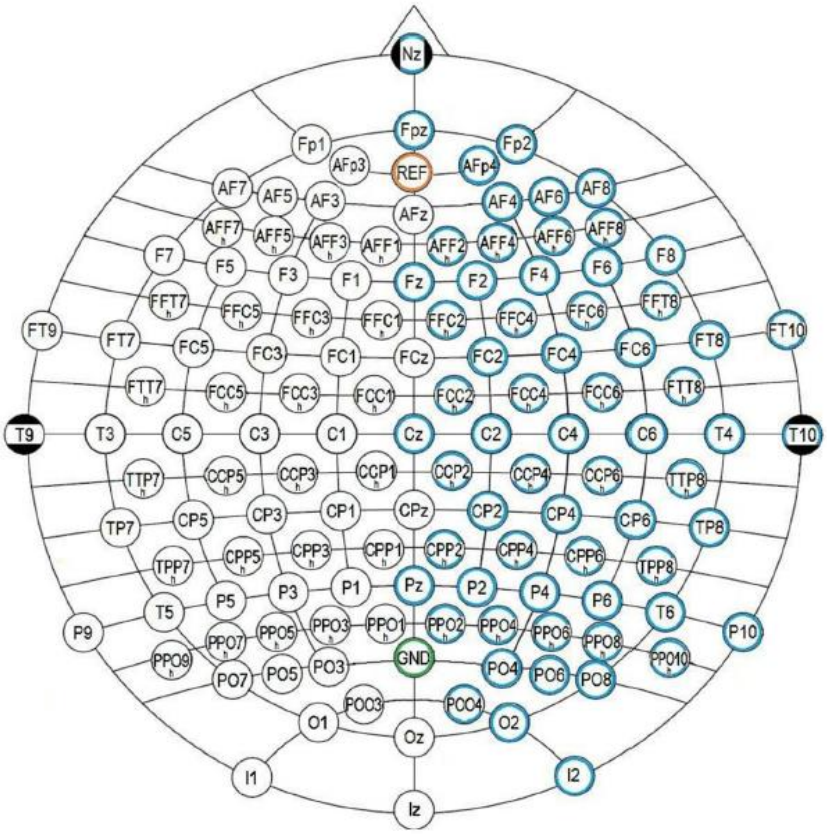

Figure 2.2. EEG channel location for 128 channel (10-5 system)

2.1.5 Sampling Frequency and Number of Bits

The sampling rate or frequency refers to the number of samples taken per second in a continuous signal. According to the Nyquist rate, highest frequency that can be represented in the signal can be half of the sampling rate. Besides from that, signals having different sampling frequencies handicap fair comparison. Datasets in our list, have different sampling rates as listed in Table 2.2.

The number of bits indicates the number of different amplitude levels that may be represented. The resolution of digitally stored signal is determined by the number of bits. Our datasets have various number of bits values. Therefore, resolution of signals varies from set to set.

2.1.6 File Format

After signals are recorded, they are saved to files to be used by other researchers. Most popular data storage file type is .MAT which is a file format used by MATLAB. However, there is no standard way of storing EEG signals in .MAT files. Each research team finds a suitable structure to store their recorded signals in .MAT files. This file structure information includes how different channels are stored, how trial start information is included, how cue information is attached, how sampling rate or other useful information are added, etc... Researchers have to discover the data structure before using the data.

Another file format used to store EEG signals is .NPZ which is a compressed archive format used in Python programming language for storing and exchanging NumPy arrays. Like .MAT files, .NPZ files do not have standard way of storing EEG signals.

Besides .MAT and .NPZ file formats, there are two more popular ways to store EEG signals: European Data Format (EDF) and General Data Format (GDF). These are standardized file formats for storing and exchanging biological signals like EEG.

‘File For.’ column in Table 2.2 gives the file formats of the related datasets.

2.2 EEG MI Signal Classification Deep Learning Applications

In recent years, deep learning has become increasingly popular across a variety of fields, including neuroscience and medical imaging. To be more specific, there has been a growing interest in applying deep learning techniques to analyze EEG MI signals. Number of academic studies on this topic has increased significantly over the last five years. In this part of the literature survey, we analyze recent studies and explore some critical decisions made in these researches.

2.2.1 Datasets Used

Two most popular datasets used in EEG-based classification studies are BCI Competition IV 2a and 2b datasets, which are mentioned as set-6 and set-7 in this study. Although, these are older datasets compared to others, they are still most popular choices of researchers. According to [30], approximately 60% percent of studies has used these two datasets. Popularity of these datasets arises from BCI Competition IV which is a series of international competitions in the field of BCI held in 2008-2009 and organized by the BCI Society.

Many studies make use of set-6 and set-7 together. [31], [32], [41], [42], [43], [58], [61] and some other studies evaluate their performance on both set-6 and set-7 datasets. Starting from [1], which is one of the first and most popular applications of deep learning in EEG MI classification field, recent studies, [33], [34], [35], [36], [37], [38], [39], [40], [44], [59], [60], [62] and [63] make use of set-7. Besides, other researches like [64], [65], [66], [67] and [68] make use of set-6.

Apart from set-6 and set-7, “EEG datasets for motor imagery brain-computer interface” also known as GigaDB, “EEG Motor Movement/Imagery Dataset (EEGMMIDB)” and “High Gamma Dataset (HGD)” which are mentioned as set-1, set-2 and set-8 respectively in this study are other popular datasets preferred by scientists. Furthermore, [69] and [70] use set-1, [71], [72], [73] and [74] use set-2, [52], [58] and [75] use set-8 in their researches.

In addition to the mentioned datasets, there are other scientific works that use different datasets. Some of these datasets are publicly available, whereas others are not available for public use.

2.2.1.1 Electrode Preferences

As explained in 2.1.4, number of electrodes varies between datasets. Number of electrodes used in studies depends on available electrodes in datasets. However, we observe that almost all studies have used all present channels in datasets. Although it is known that c3, cz and c4 electrodes give great amount of information about the motor signal, studies do not restrict usage of other electrodes.

2.2.1.2 Class Preferences

Our examination shows that most of the studies concentrate on left and right hand MI tasks. One reason is that set-7, which is one of the most popular datasets, only has these two labels. However, it is not the only reason. Studies working with locally collected data which are not published mostly prefers to work with left and right hand MI classification. Additionally, some studies focus on only these two labels, although there are more classes presented in the set.

2.2.1.3 Data Distributions

Data distribution is another important topic about the usage of datasets. Because, we notice that although different researches may use the same dataset, how they distribute data between training and evaluation may still vary.

One alternative way to deal with data is hold-out strategy. In hold-out evaluation, the dataset is split into two parts: a training set and a testing set. Usage of this approach also has some alternatives. How and in which proportion data will be split depends on the researcher’s choice.

The second alternative way is cross-fold validation. This method involves dividing the dataset into k equal-sized parts, or folds, and training the model k times, each time using k-1 folds for training and the remaining fold for validation. The model's performance is measured by calculating the average performance across the k iterations. While using this approach, researcher has to decide how many folds will there be during evaluation.

Selection of data distribution over the training and test sets affects the classification performance crucially. While many researches do not pay attention to this issue, [44] notices the problem, creates different results for different data distributions and compares its results accordingly. One of the datasets they use is set-7, and they point out to three different data distributions while using the set. Set-7 has five sessions recorded. First three sessions were published as a training set, and latter two sessions were published as a test set some time later. According to [44], studies divide into three groups. First group use all five sessions and randomly split data into training and test sets. Second group use first three sessions as training set and last two sessions as test set. Third group use first three sessions only and prefers cross fold validation. Besides from findings of [44], in our investigation we also see that [63] use only first two sessions of the dataset for both training and test purposes.

2.2.2 Frequency Bands

In a healthy human brain, as stated in [2] signals with frequencies from 0.5 Hz to 100 Hz are generated. Depending on brain activity, there are some distinguishing rhythms of EEG signals. These rhythms are defined slightly different by different authors. Here we give the EEG signal rhythms and their frequency bands as described in [2].

There are six EEG rhythms that are discovered by scientist while exploring EEG signals. Delta rhythms, with frequencies from 0.5 to 4 Hz, are seen during deep sleep. Theta rhythms have frequencies between 4 and 8 Hz and they are seen during light

sleep, or in state of hypnosis. The mu is a motor rhythm that has a frequency range of 8 to 12 Hz. Alpha rhythms are visible in the absence of visual stimulation and have frequencies between 8 and 13 Hz. Beta rhythms, with frequencies from 12 to 30 Hz, are observed during focusing. Finally, gamma rhythms, with frequencies between 30 and 100 Hz, are observed during motor activities. We create a chart in Figure 2.3 that shows the EEG rhythms aligned along the frequency axis. Note that in the plot, frequency axis is in logarithmic scale.

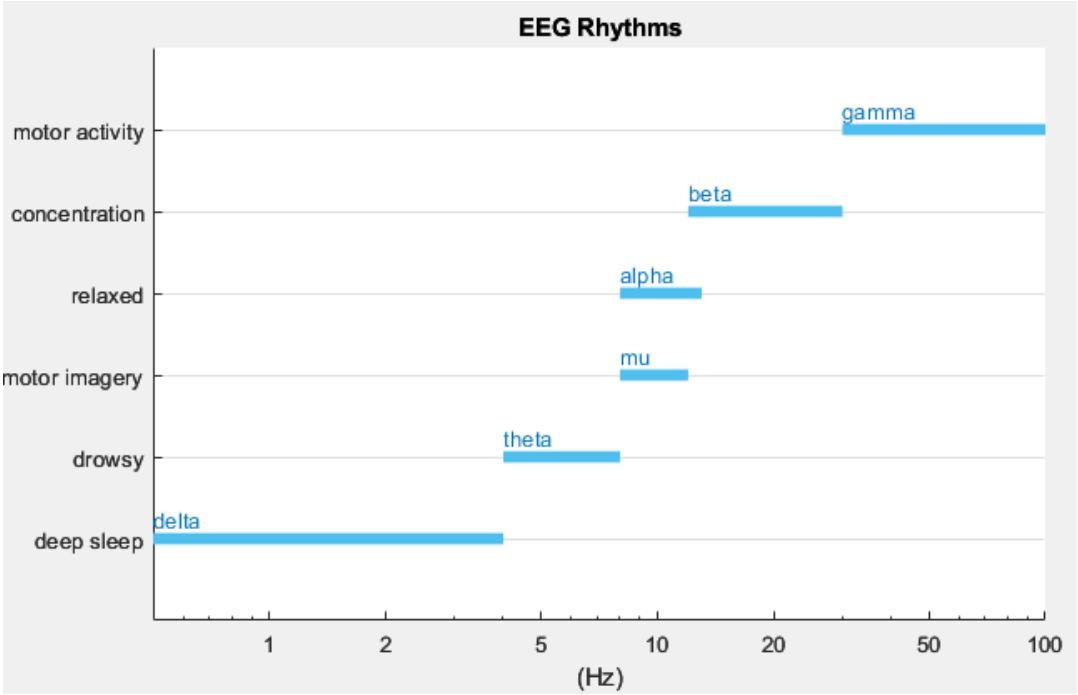

Figure 2.3. Rhythms occur in EEG signals

Many other researches and also [2] explains that different brain parts are activated during different MI tasks. Some tasks decrease certain EEG rhythms in particular areas of brain. This phenomena is called event related desynchronization (ERD). Inverse situation also exists, in which MI tasks increase certain EEG rhythms in some brain areas. This situation is called event related synchronization (ERS). To give an example, when a person imagines moving his/her left hand, it leads to specific changes in the EEG signal in different regions of the brain. These changes include ERD in the contralateral (right) motor cortex also known as C4 region, and ERS in

the ipsilateral (left) motor cortex also known as C3 region. Inversely, when a person imagines moving his/her right hand, it leads to ERD in C3 region and ERS in C4 region.

Researches also show that ERD and ERS effects change according to frequency bands. MI events mainly affects mu and beta bands. Generally a MI task causes an ERD in mu band, and an ERS in beta band. Knowing this information, almost all studies focus on these frequency bands. Nevertheless, there are some researches, [31], [64] and [68], that use all frequency spectrum (0.5-100 Hz).

2.2.3 Input Formats

When we examine existing deep learning applications, we see that there are three popular types of feeding EEG MI signals to deep neural networks. Since deep neural networks (DNN) have made significant breakthroughs in image tasks like classification, segmentation or recognition, first input format that comes to mind is image.

There are a few ways of converting EEG signals into images in literature. Most used conversion is Short Time Fourier Transform (STFT) which is also known as spectrogram. STFT approach breaks down a signal into a series of overlapping short-time segments and calculates the Fourier Transform of each segment. This allows for analyzing the frequency content of the signal over time. Many researches including [1], [33], [34], [35], [36], [37], [38], [39], [40] and [44] use STFT to transform EEG signals into images. Besides STFT, there are also other methods for converting time signals to images like Continuous Wavelet Transform (CWT) which is used in [59].

Second common way to use EEG signals in DNNs is directly using raw values. As it is known, EEG signals are time series signals and they can be directly fed into neural networks without any conversion. [31], [32], [41], [42], [43], [58], [65], [68], [71], [72] and [73] use this type of input format

Lastly, EEG signals can be used as DNN inputs after some features are extracted using methods like Principal Component Analysis (PCA), Common Spatial Pattern (CSP) and Wavelet Transform (WT). This feature extraction methods have been widely used in machine learning. Still they are preferred in deep learning applications. Among recent studies, [62], [63], [69], [70], [75] use CSP, [60] uses PCA and [74] uses WT.

2.2.4 Deep Learning Methods on EEG MI Classification

Majority of studies in EEG MI task classification applications use CNN models as deep learning architecture. Besides using CNN alone, some studies build cascaded structures with CNN models. CNN+SAE structure is considered in [1], CNN+VAE (Variational Autoencoder) is experimented in [38] and CNN+GAN (Generative Adversarial Network) is used in [35]. In addition to CNN based architectures, [60] uses DBN-RBM (Deep Belief Network with Restricted Boltzmann Machines) and [69] uses LSTM (Long Short-Term Memory) which is a type of RNN (Recurrent Neural Network).

2.3 Deep Clustering Applications

As mentioned in earlier chapters, in this study, we want to design a deep feature extractor (DFE). Therefore, in this part of literature survey we examine existing deep clustering methods used in various applications.

Deep clustering is a machine learning technique that combines deep learning and clustering algorithms to perform unsupervised learning tasks. It aims to discover hidden structures and patterns in unlabeled data by automatically grouping similar samples into clusters. Traditional clustering algorithms frequently use manually created features or distance measurements in order to group data points, whereas in deep clustering feature representations are directly obtained from the raw input data.

By using deep neural networks, complex features can be learned and extracted automatically.

As implied in [8], [10], [11] and [78], deep clustering methods consist of two main parts, one of which is representation part and the other is clustering part. In representation part, embedding features are extracted and in clustering part according to these features clustering is implemented. These two parts are examined by [8] on existing studies and put in three categories.

The first category is named multi-step sequential deep clustering. The studies in this category ([16], [77] and [79]) trains representation and clustering sections separately. First representation part is trained. After representation training is finished, clustering part starts training.

Second category is named joint deep clustering. In joint training, representation and clustering parts are trained simultaneously. To do so, a combined loss function is defined, and network is trained accordingly. Also, in many studies representation part is trained separately before joint training. This separate training period is called pre-training. Studies [14], [15], [19], [80] and [81] use joint deep clustering method.

Third category is defined as closed-loop multi-step deep clustering. It is also known as iterative deep clustering. The studies in this category ([5] and [82]) trains representation and clustering sections iteratively. In other words, two steps are trained alternately instead of feedforward linear fashion.

2.3.1 Representation Learning

Representation learning, also can be referred as feature learning, is a technique that concentrates on automatically learning rich and practical representations or features from raw input data. The aim is to convert the data into a more condensed and useful representation that captures the underlying structure and patterns in the data.

The most popular approach in representation learning part is Autoencoders (AEs). An AE is a type of neural network that learns to compress and reconstruct input data. It consists of an encoder that compresses the data into a lower-dimensional representation and a decoder that reconstructs the original data from the compressed representation. During training, the autoencoder aims to minimize the difference between the input data and the reconstructed output, effectively learning a compact representation of the data. Convolutional Autoencoders (CAEs) are a variant of AEs that employ convolutional neural networks as their underlying architecture. In [7], [14], [15], [16], [18], [19], [20], [21], [22] and [24] AEs or CAEs are used for compact representation of data.

Generative models are another branch of deep unsupervised representation learning. Normally in other methods, an embedded representation is generated from input data, whereas in generative methods, network learns how to generate input data from an embedding representation. Among generative models, the most common is Variational Autoencoders (VAEs). It is used in [23] and [25]. Generative Adversarial Networks (GANs) are also used for representation [83]. In addition, [17] uses VAE and GAN together.

Contrastive learning has become more popular, in the past few years. Contrastive learning is a representation learning approach that trains a model to maximize similarity between similar samples and minimize similarity between dissimilar samples. By mapping input samples into an embedding space, the model learns to cluster similar samples together. Studies [26], [80] and [81] use contrastive techniques for representation learning.

Lastly most practical method to obtain embedding representation is using pretrained convolutional networks such as AlexNet, VGG16/19, DenseNet, ResNet, etc... [5] and [9] use pretrained networks for image representation. Even, [13] tries using multiple pretrained networks together, and combines features generated from these networks.

2.3.2 Clustering Part

For clustering part, diversity of methods is less than representation part. Majority of the studies use k-Means algorithm in clustering part for the sake of simplicity. However, methods differ in choice of the loss function [11]. Clustering Assignment Hardening Loss (CAHL), K-Means Loss [84], Balanced Assignment Loss and Locality Preserving Loss are popular loss functions used for clustering tasks.

In clustering algorithms, CAHL [14] is a technique that is used to obtain better soft assignments. This is achieved by introducing an auxiliary distribution that enforces stricter probabilities, measuring the distance between the auxiliary and original distributions using the Kullback-Leibler (KL) divergence and finally improving cluster purity and achieving confident assignments.

The Balanced Assignments Loss [19] is also used as a loss function in clustering algorithms to achieve balanced cluster assignments. It is formulated as the KL divergence between the uniform distribution and the probability distribution of a data point belonging to each cluster. By minimizing this loss, the goal is to achieve a uniform probability in the distribution of data points among clusters.

The Locality Preserving Loss [85] is a mathematical formulation that aims to preserve the locality of clusters by enforcing nearby data points to be closer, and the k-Means Loss [84] is used to ensure that data points are equally distributed around cluster centers, the loss function minimizes the distance between each data point and its assigned cluster center, achieving a balanced distribution.

2.3.3 Datasets Used

Deep clustering applications are mostly used on image datasets. Studies [7], [13], [14], [15], [16], [17], [18], [21], [27] and [76] use image datasets like MNIST, USPS, Fashion-MNIST, COIL20, COIL100, CIFAR-10 etc. [19] also tries face recognition databases like FTGC, YTF, CMU-PIE besides image sets. [5] tries deep clustering

on a much larger dataset: ImageNet. There are also some articles [6], [12] that work on speech or audio signals. In addition to publicly available popular datasets, some case studies on real data are also made. The study in [9] works with soldering data classification and [24] practices bearing fault classification in induction motors.

During our literature survey, we didn’t come across to a research that implements a deep clustering method in the field of EEG MI signals.

CHAPTER 3

UNIFICATION OF EEG MI DATASETS

In literature, there are three popular input formats to use EEG signals with deep learning applications, as discussed in section 2.2.3. Since it gives detailed information on time and frequency of the EEG signals, in this study STFT of the EEG signals are preferred as input format. STFT conversion produces two dimensional data, which helps us to represent the converted data as images. In the rest of the thesis we will alternatively use the terms “STFT of EEG signal” or “STFT image”.

In section 2.1, we present publicly available EEG MI datasets and their properties. Our literature research reveals that during the collection of datasets, researchers has used different equipment and followed different protocols. As a result of this difference, we cannot combine datasets directly. Number of classes, signal length, acquisition procedure, number of electrodes, sampling frequency, number of bits used and file format vary among these datasets. In this chapter, we present details of how these challenges are overcome and how datasets collected by different works are unified in one dataset. All phases throughout this chapter is implemented in MATLAB 2022b.

3.1 Signal Processing

We use all datasets whose properties are summarized in Table 2.2 to create a unified EEG MI dataset. As can be seen from the table, some critical properties of these datasets differ. In order to reduce these differences we do some signal processing before applying STFT. Signal processing part consists of four steps as shown in Figure 3.1, which are filtering, resampling, amplitude normalization and time segment selection.

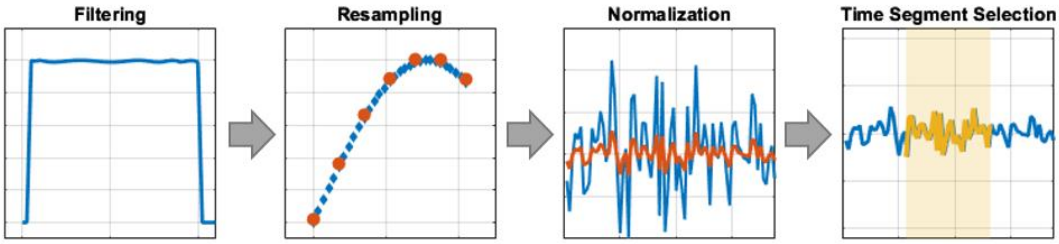

Figure 3.1. Signal Processing Steps

3.1.1 Filtering

According to Nyquist's theorem, the maximum frequency that can be accurately represented in a digital signal is half the sampling frequency. When we examine sampling frequencies (SF) of interested datasets, we see that sampling frequencies differ among them. This means that signals from different datasets contains information from different frequency ranges. For instance, SFs of set-5 and set-6 are 1000Hz and 250Hz respectively. Therefore, considering Nyquist's theorem signals in set-5 contain information up to 500Hz, where signals in set-6 contain information up to 125Hz. Regarding this fact, we want to limit higher frequency information in these datasets. Since human brain generates signals up to 100Hz, we define upper frequency limit as 100Hz.

As for the lower frequencies, we observe high level of noise around the 0 Hz in some of the datasets. However, we want to filter out this noisy information without disrupting useful information. As elaborately explained in section 2.2.2, useful information MI events generate, is mostly located in mu and beta bands. Therefore, we define lower frequency limit as 3Hz, which is high enough for filtering noisy information and low enough for saving useful MI information.

To sum up, we create a bandpass filter with cutoff frequencies 3Hz and 100Hz, and apply this filter as the first step of signal processing. There is only one exception for this filtering: SF of set-2 is 160Hz, which means that the highest frequency

information represented in this dataset is 80Hz. So, we apply only low pass filter with cutoff frequency 3Hz to signals of this dataset, we don’t filter high frequencies.

3.1.2 Resampling

SF determines the highest frequency that can be represented in the signal. In addition to that 1/SF equals to the time between two samples of a signal. For instance, if we examine set-5 and set-6 again, time duration between two samples of a signal in set-5 is equal to 1ms (SF: 1000Hz) and time duration between two samples of a signal in set-6 is equal to 4ms (SF: 250Hz). It leads to inaccurate synchronization of signals if sampling rates are not matched and it can significantly impact the quality and reliability of signal analysis, comparison, and processing.

In this study, in order to overcome sampling frequency mismatch, we apply resampling to all signals. Resampling is the process of altering a digital signal's SF. It involves modification of a signal's discrete samples to increase/decrease the SF while retaining the fundamental properties of the original signal. Paying regard to Nyquist’s theorem, we choose the desired sampling frequency 250Hz.

In conclusion, we convert SF of all signals to 250Hz by resampling. As can be investigated from Table 2.2, no resampling is required for set-6 and set-7. Also, resampling involves increasing SF (upsampling) in set-2, and it requires decreasing SF (downsampling) in other sets.

3.1.3 Amplitude Normalization

Different signals of different datasets were recorded with different systems. As a result of this, amplitude of signals vary among datasets. Some samples are saved between values -255 and 255, some other samples are saved between -1 and 1, etc… In addition to that some signals have a nonzero average that may be arising from

recording setup. In brief words, signals of different datasets have different min/max and mean values.

Although we do not expect a change in the overall shape or pattern of the converted images, we equalize mean and min/max values of signals for the sake of neatness. Equalizing mean values includes adding or subtracting an offset value (Equation 3.1) to signal to make the average value equal to zero. Equalizing min/max values includes rescaling signal by dividing to a factor (Equation 3.2) such that it fits to a defined minimum and maximum range. As a result, we make all signals in the same amplitude range and zero average as given in Equation 3.3.

$$offset = \sum_{s=1}^{n} y_s \tag{3.1}$$

$$scale\_factor = \frac{\max(|y_s|)}{expected\_max} \tag{3.2}$$

$$y_{s-normalized} = \frac{y_s - offset}{scale\_factor} \tag{3.3}$$

3.1.4 Selecting Time Segments

We discuss different signal acquisition setups and scenarios in 2.1.3. There are three common sections in trials of different datasets. We summarize these parts as MI event, pre MI event and post MI event. Also, there is a duration that a cue of MI task is displayed which is overlapping with one or more of these three parts.

In this study, as [1] suggested, we extract 2 seconds of time signal from each trial and ignore rest of the time series data. Similar to the same work, we define the extracted 2-second signal range starting from 0.5s after the cue is first displayed and finishing at 2.5s after the cue is first displayed.

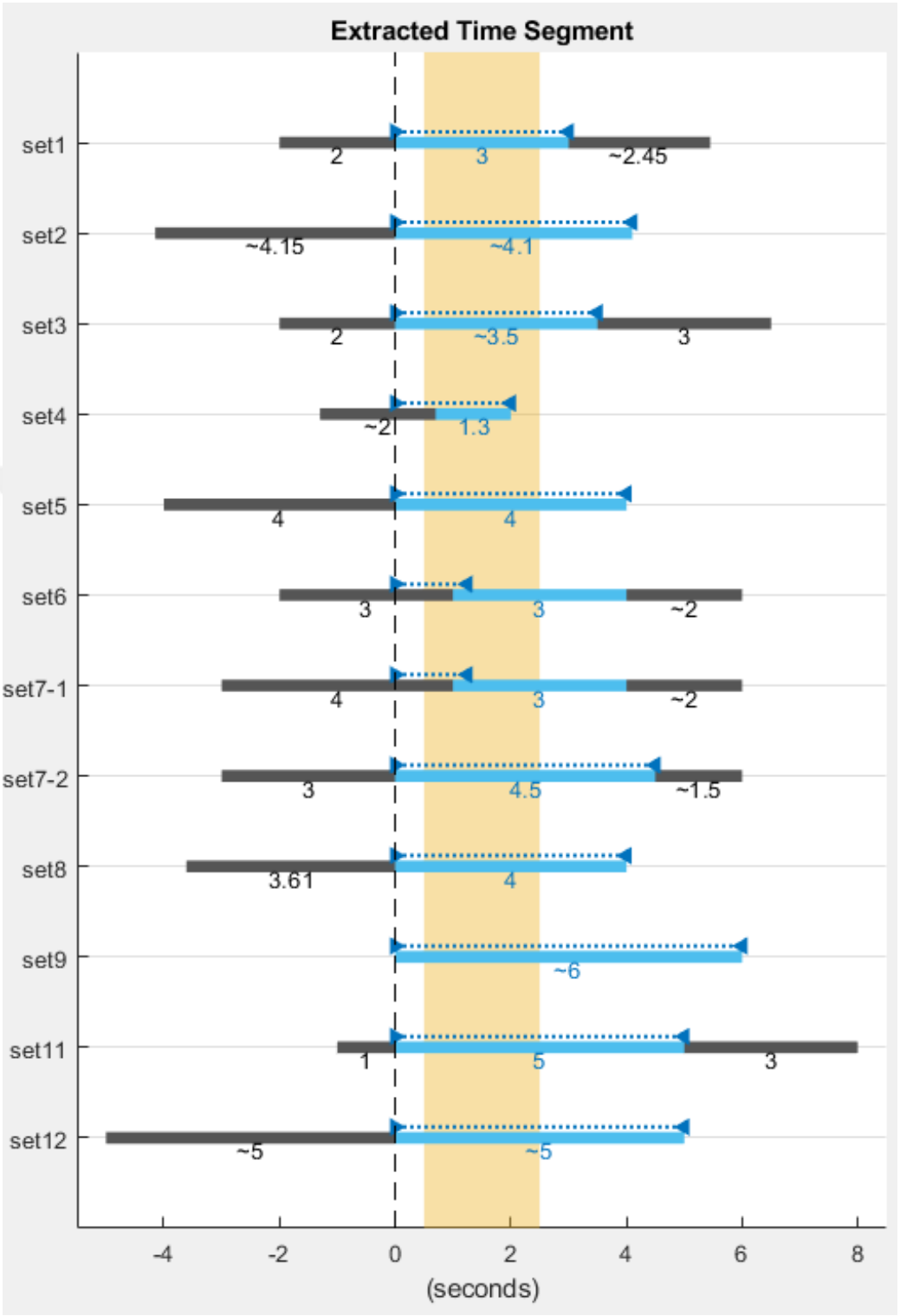

Figure 3.2. Acquisition of different datasets

3.2 Input Image Format

In this study, the STFT of the EEG signals are represented as images and the input image format from [1], [28] are used with some minor modifications. In the following sections, details of how preprocessed signals are converted to time-frequency images are presented.

3.2.1 Time Signal to Time-Frequency Image

After preprocessing part, 2-second signals are extracted from time series data as explained in section 3.1.4 in detail. These 2-second time series data undergoes STFT transformation. For a signal having SF of 250Hz, 500 samples are extracted. We compute STFT with window size of 64 as [1] suggested and starting from sample-1 this window is shifted towards sample-500 with time step of 14 samples. This results in STFT calculation of 32 windows, and the last window covers up to sample-498. Samples 499 and 500 are ignored in this conversion.

For each window, FFT (Fast Fourier Transform) length is chosen as 512. Since signals are not complex valued (real valued), resulting FFTs are symmetrical along the frequency axis. Therefore, only one side of the FFT spectrum is used. As a result, with 512 FFT points, resulting spectrum has 257 points including zero. This means that for 1 time window, STFT returns a 257-point vector. For 32 time windows, STFT returns a 257x32 sized image. 257 and 32 are the pixel counts along the frequency and time axes. Figure 3.3 (left) shows image obtained after STFT operation.

MI ERD and ERS effects are mostly visible on mu and beta bands. Therefore instead of using all frequency information, we only prefer including mu and beta bands in our image dataset. On Figure 3.3 (right) frequency bands that we are interested are shown.

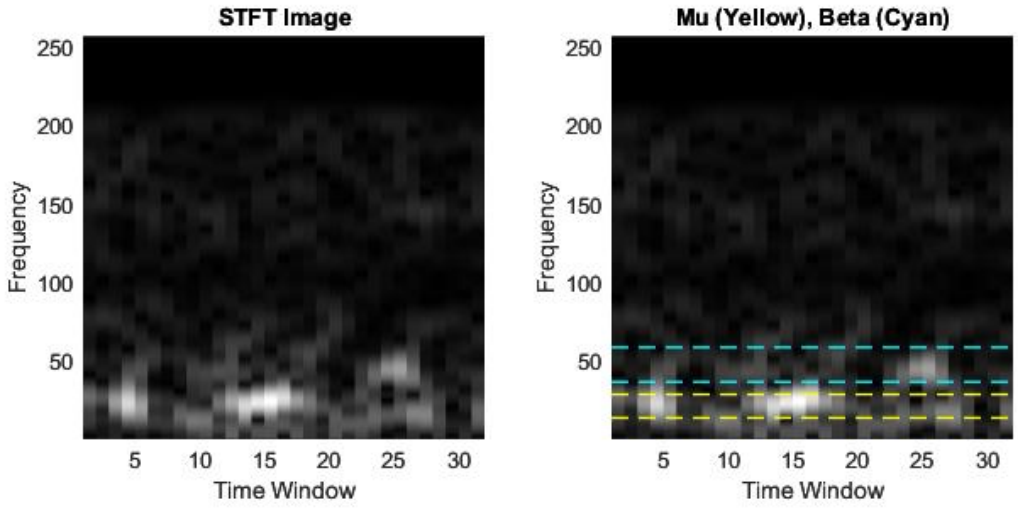

Figure 3.3. STFT image (on left), interested frequency bands (on right)

As discussed in 2.2.2, limits of mu and beta bands vary according to different researches. In this research, we take mu band as 6-14 Hz range and beta band as 17.5-28.5 Hz range. In the STFT image, this corresponds to 16 pixels in mu band and 23 pixels in beta band along frequency axis.

While extracting interested frequency regions, we use 16 frequency points of mu band directly. However, while using beta band, in order to maintain consistency between the effect of mu and beta bands, we rescaled frequency axis with cubic interpolation so that beta band frequency points are reduced to 15 points. After extracting mu and beta bands, the rest of the image is ignored and the mu and beta images extracted are put on top of each other. The figure given below shows the extracted image of frequency bands.

Up to now, only frequency and time information is discussed. In EEG signals, there is also one more dimension. It is the spatial information. During the same MI event, data is recorded with different electrodes located on different places of the skull. Signals from different electrodes are saved to different channels in datasets. Considering our dataset pool, the least number of electrodes is in set-7. This set contains only 3 electrodes (c3, c4 and cz). Even so, [2] states that MI event ERD and

ERS effects are distinctly visible on these three electrodes. Therefore, in order to use all datasets together in this work, we prefer using c3, c4 and cz electrodes.

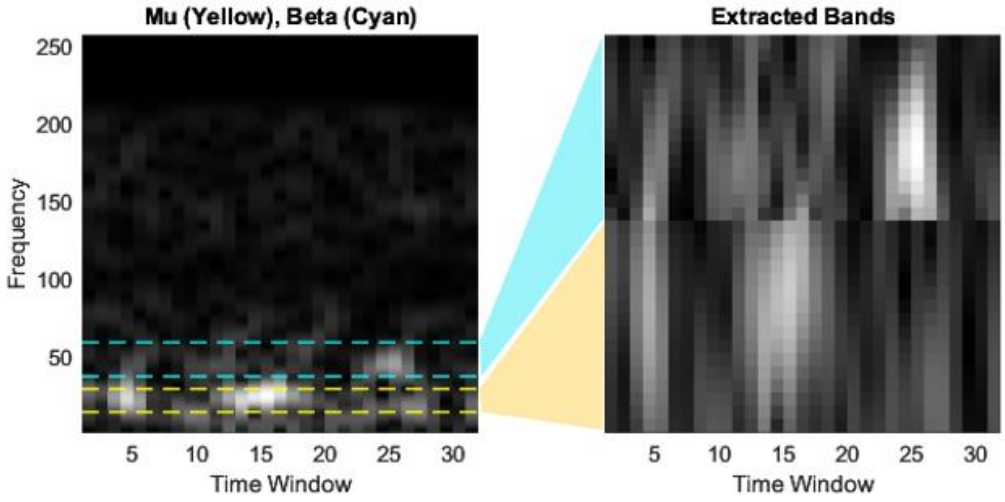

Figure 3.4. STFT image mu and beta band extraction

To study our methods, we prepared two different image datasets. First dataset contains one electrode information per image, whereas the second dataset contains combined three electrode information per image. The following two sections give details of these datasets.

3.2.2 Single Electrode Image Dataset

In this image set, all images are in size of 31x32 and containing information from only one electrode (Figure 3.4 right image). Total number of images for each electrode is given in Table 3.1 As can be seen from the table for each electrode 170314 images are generated. 72828 of these images belong to left and right hand MI events whereas the remaining images are corresponding to other MI events such as feet, tongue, etc. Since we include three electrodes (c3, c4 and cz) in our study, grand total number of images are 510942.

Table 3.1 Images extracted per electrode from datasets

Set No	*Left/Right Hand Labeled Image*	*Total Labeled Image*
Set-1	10520	14316
Set-2	4918	26214
Set-3	0	3528
Set-4	22447	63892
Set-5	1200	1400
Set-6	2592	5184
Set-7	6520	6520
Set-8	6742	13484
Set-9	4	18
Set-10	0	0
Set-11	16765	33518
Set-12	1120	2240
Total For Each Electrode	**72828**	**170314**
Total Number In Dataset	**218484**	**510942**

3.2.3 Combined 3-Electrode Dataset

This dataset is the combined version of the first dataset. Instead of adding images obtained from c3, c4 and cz separately to the dataset, they are combined in one image. Therefore, each image in this dataset is in size of 93x32 and total number of images in this dataset is one third of the first dataset, which is 170314.

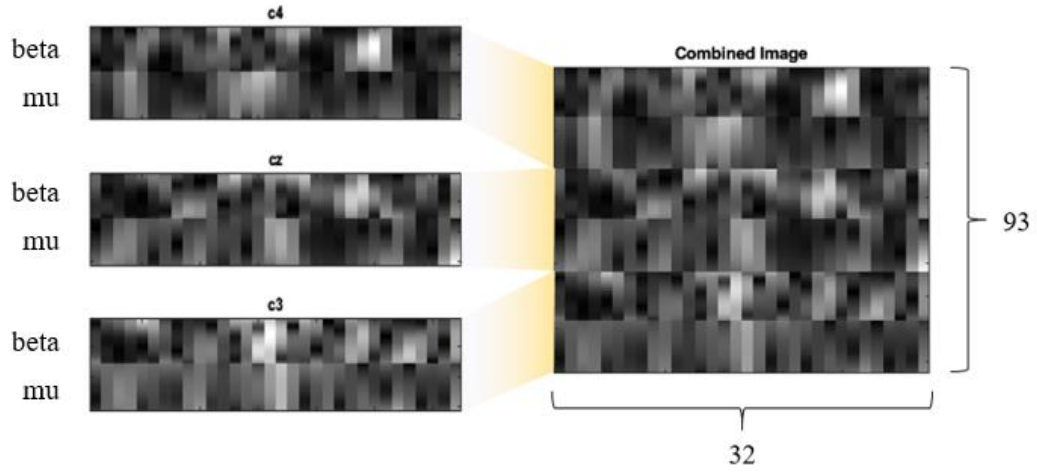

Figure 3.5. Combined STFT image

3.3 Whitening

We make some preprocessing in raw signal level in order to reduce the differences between the datasets and so combine them. We also expect that these signal processing steps reduce the information specific to subject or specific to dataset, while boosting the information related to the motor imagery event. However, when we examine the data obtained after signal processing and STFT transformation, we notice that information specific to subject or dataset is more apparent than information specific to MI event. In Figure 3.6, average of images labeled as right hand and left hand MI events that belong to two subjects are shown. On top row left and right hand images of subject 1 are shown, on the bottom row, those for subject 2 are shown. When we examine these images, we see that “beta c4” and “beta cz” regions of subject 1 are darker compared to other regions. Also, “mu c4” and “mu c3” regions of subject 2 are darker independent of the related MI event task. This dark areas are features that belong to specific subject. Although we seek features that reflect MI event, from these images it can be easily said that instead of having similarities between images of same MI events, we have similarities between images of same subjects.

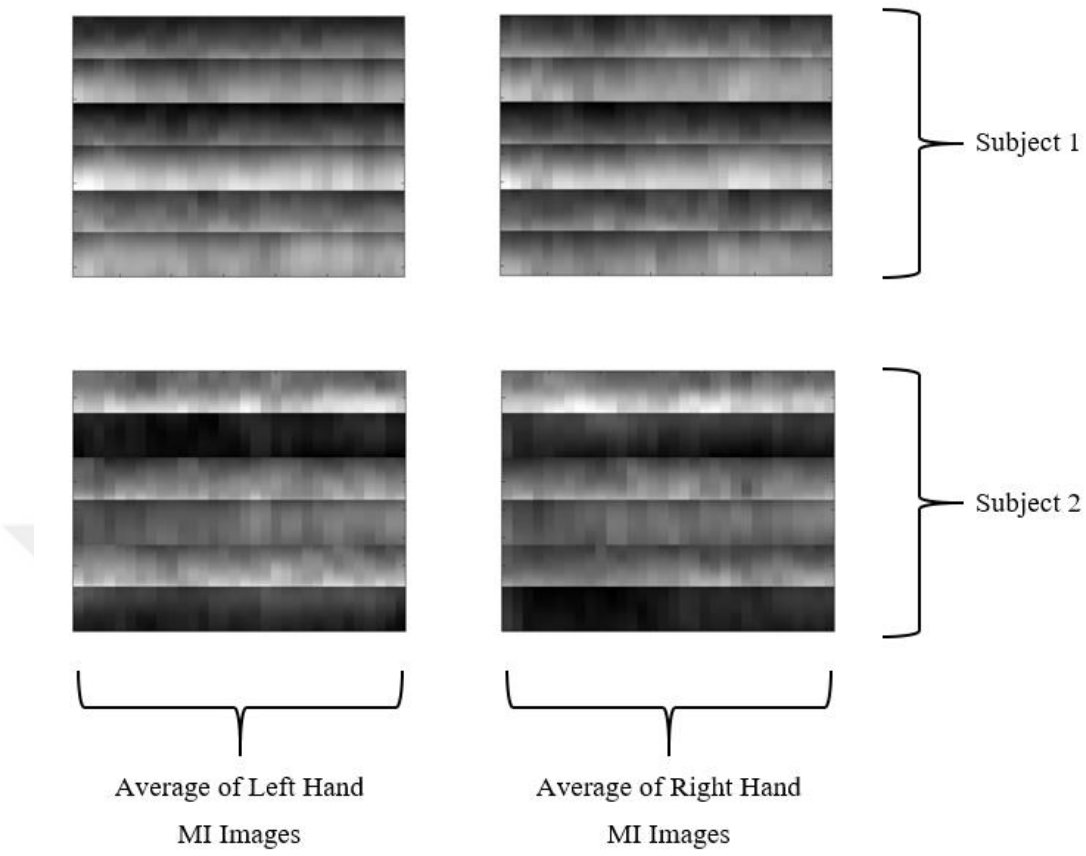

Figure 3.6. Left/Right hand MI event image averages

To overcome this issue we apply whitening to images. Whitening refers to the statistical procedure that includes decorrelating and normalizing data to have zero mean and unit variance in order to facilitate further analysis on data. In our case, we redefine whitening. Our whitening is a row based whitening that ensures average value of each row is equal to 0 and standard deviation of each row is equal to 1. When we are calculating averages and standard deviations we include all images of a specific subject.

On the figure below, we recalculate average values of left and right hand MI events for two subjects after whitening is applied. It can be clearly seen that whitening enhances features carrying MI event information, whereas it degrades features carrying information specific to subject or specific to dataset information. When we compare Figure 3.6 and Figure 3.7, we see that before whitening, images on the same

row are more similar which belong to same subject and after whitening, images on the same column are more similar which belong to same MI event.

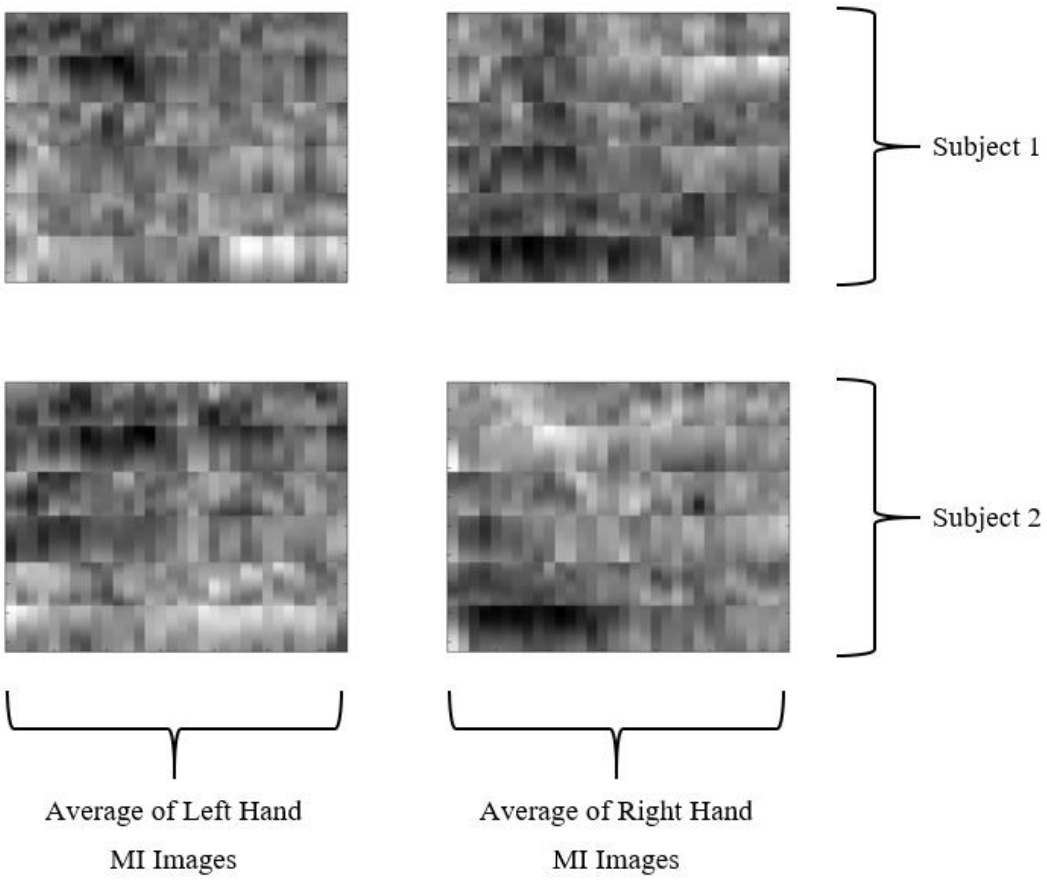

Figure 3.7. Left/Right hand MI event image averages after whitening

3.4 Data Augmentation

Data augmentation is important in deep learning because it boosts the quantity and diversity of training data, which helps models to generalize more effectively, to perform better by avoiding overfitting, and to improve overall accuracy of the trained model.

Popular data augmentation techniques that are used for images are random cropping, flipping, rotation, scaling, brightness/contrast adjustment, adding noise, blurring, and color jittering. However, we evaluate that most of these techniques are not

suitable for our unified EEG MI dataset containing STFT images. Because, images in our dataset has frequency information on vertical axis, and flipping, rotating or scaling obviously distorts the frequency information. In addition to that, we do not find color or brightness altering techniques useful in our case due to gray scale and whitened characteristics of images in our set.

We find two methods of augmentation beneficial in our study. The first one is gaussian noise, and the second is circular shift. We also try using these methods jointly. In the following two subchapter we explain details of these methods.

3.4.1 Gaussian Noise

Adding Gaussian noise is a common approach used in data augmentation to improve the generalization and robustness of deep learning models. It includes adding noise generated from a Gaussian distribution, commonly referred to as normal distribution, to the input data in order to introduce random fluctuations to them.

Our augmentation method slightly differs from standard gaussian noise implementation. Firstly, we create a normal distributed noise with 0 mean and 0.005 variance. Number of samples in distribution is equal to number of samples in one row of the image. Then, we add this calculated noise to one row of the image. Then we repeat this process for every row. In Figure 3.8, an image from dataset and augmented image on which gaussian noise added are shown together.

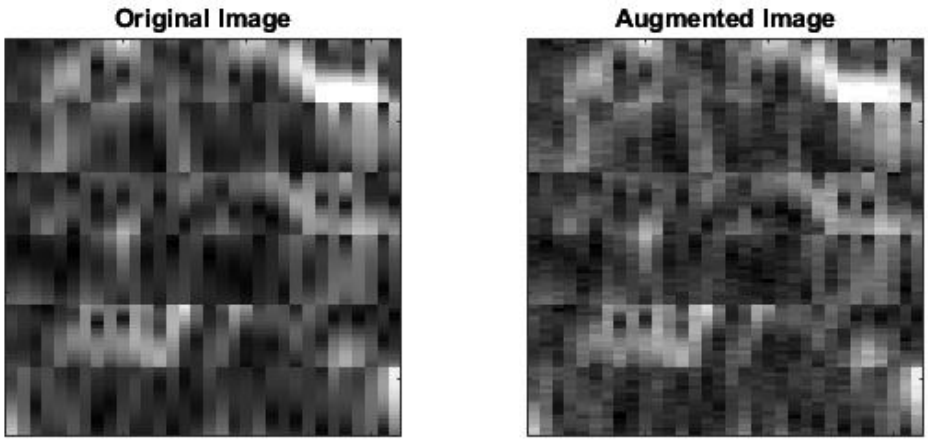

Figure 3.8. Gaussian noise augmentation

3.4.2 Circular Shift

We mention in 3.4 that augmentation methods like random cropping, flipping, rotation and scaling distort frequency information which is crucial in our set and should not be disrupted. Therefore, we propose another method which preserves frequency information. Since frequency information is stored along vertical axis, we do not want to change the image along the vertical axis. In addition, during the acquisition of signals, reaction times of different subjects to motor imagery cues differ, which results in diversity in starting moments of actual motor imagery events in the order of seconds. Therefore, we find applying augmentation on horizontal axis meaningful. In order to apply augmentation only to horizontal axis, we use horizontal shift. In addition, to preserve image shape, we make horizontal shift circularly which means that pixels on the rightmost side of the image enter again to the image from the leftmost side during shifting operation. In Figure 3.9, original image is shown on the leftmost, the other four images are obtained by circular shift augmentation.

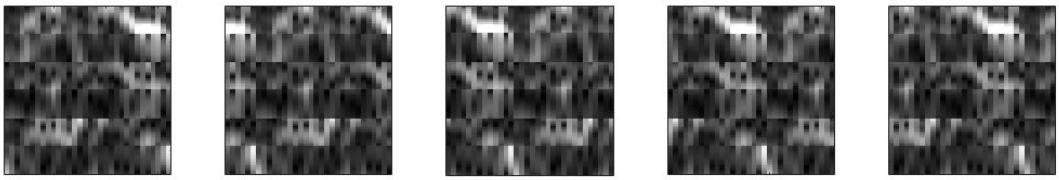

Figure 3.9. Circular shift augmentation

3.4.3 Joint Augmentation

In our experiments, we use gaussian and circular shift augmentations jointly. While shifting images circularly along the horizontal axis, we also add gaussian noise. In the preparation of this dataset, there are two parameters to optimize. One parameter is how many shifted image will be included in the augmented dataset. The other parameters is what will be the variance of the gaussian noise. These parameters are optimized during experiments and is explained in further chapters.

CHAPTER 4

DEEP FEATURE EXTRACTOR

After creating EEG image datasets, details of which are explained in the previous chapter, we design DFE (Deep Feature Extractor) networks. These DFEs use EEG STFT images as inputs and create feature vectors from them. In addition, in order to measure the quality of extracted features, we design feature classifiers that use extracted features from DFEs and output classification results. For the sake of simplicity as most of the other studies do, we work on EEG MI images that belong to left and right hand MI event group in this study. This chapter presents implementation details of all used networks. All the implementation is done using PyTorch 1.13.

Firstly, we introduce a CNN implementation that directly gets input images and outputs binary classification results for left/right hand MI events. This network is used for benchmarking and the success of DFEs are determined by comparing to this network. Then we give implementation details of 6 different DFEs and their feature extractors.

3 different methods with 2 different dataset structure form the basis of 6 different DFEs. The methods used are “reconstruction and clustering”, “reconstruction and classifying” and “reconstruction only”, while the datasets considered are single electrode and combined 3-electrode datasets which were previously described in sections 3.2.2 and 3.2.3 respectively. Table 4.1 shows how we name the experimented DFE networks.

Table 4.1 Experimented DFE methods

The loss functions used for DFE construction	*Single Electrode Dataset*	*Combined 3-Electrode Dataset*
Reconstruction and Clustering	DFEv1S	DFEv1C
Reconstruction and Classifying	DFEv2S	DFEv2C
Reconstruction Only	DFEv3S	DFEv3C

4.1 CNN Implementation

As mentioned earlier, CNN will be used to compare results of our DFEs. We adopt CNN implementation from [1] where 1D convolution is applied. As explained in this study, vertical position of activation or deactivation is crucial in classification performance since this axis holds the frequency and electrode information. Contrary to vertical location, horizontal location of the activation or deactivation is not that important since it is the time axis. As a result, convolution on frequency-electrode axis is not desired, only convolution on time axis wanted. In other words, 1D filtering is applied along the horizontal axis.

CNN network accepts inputs in size of 93x32 from combined 3-electrode dataset and it consists of 1 convolutional, 1 max pooling, 1 flattening, and 1 fully connected (FC) layer as shown in Figure 4.1.

In convolutional layer, 30 filters with size of 93x3 are applied. On this layer, no padding is applied. Since filter height is equal to input image height, only in horizontal direction filters are slid. After convolution, max pooling is performed. This layer reduces the width dimension of the feature maps 10 times. As a result, 30 feature maps with size 1x3 are obtained. All these feature maps are flattened before the fully connected layer that takes the flattened feature maps and produces logits, which represent the scores for each class.

For the training, cross entropy loss is used with Adam optimizer. Learning rate is set to 0.0002 and mini batch training is made. Batch size is adjusted to 50. With these settings, network is trained with a lower limit of 100 epochs (empirically 100 epochs are enough for the network to converge). After 100 epochs, loss change is tracked to stop the training process. To monitor the trend in loss function, first we calculate the average loss of last 15 epochs as the “recent average” and we calculate the average loss of 15 epochs that come before the other 15 epochs as the “historical average”. Then we calculate the ratio of the “recent average” to the “historical average” When this ratio goes down below 0.97, training is stopped. This ensures that training continues until the network stops learning.

After training, the network is used for testing. Network produces 2x1 outputs corresponding to two classes (Actually one node at the output layer is enough for binary classification, but to remain loyal to original study, we did not change the structure). In testing, maximum value in this layer is selected as the predicted class. For the overall accuracy on testing data, mean values of these accuracies are used.

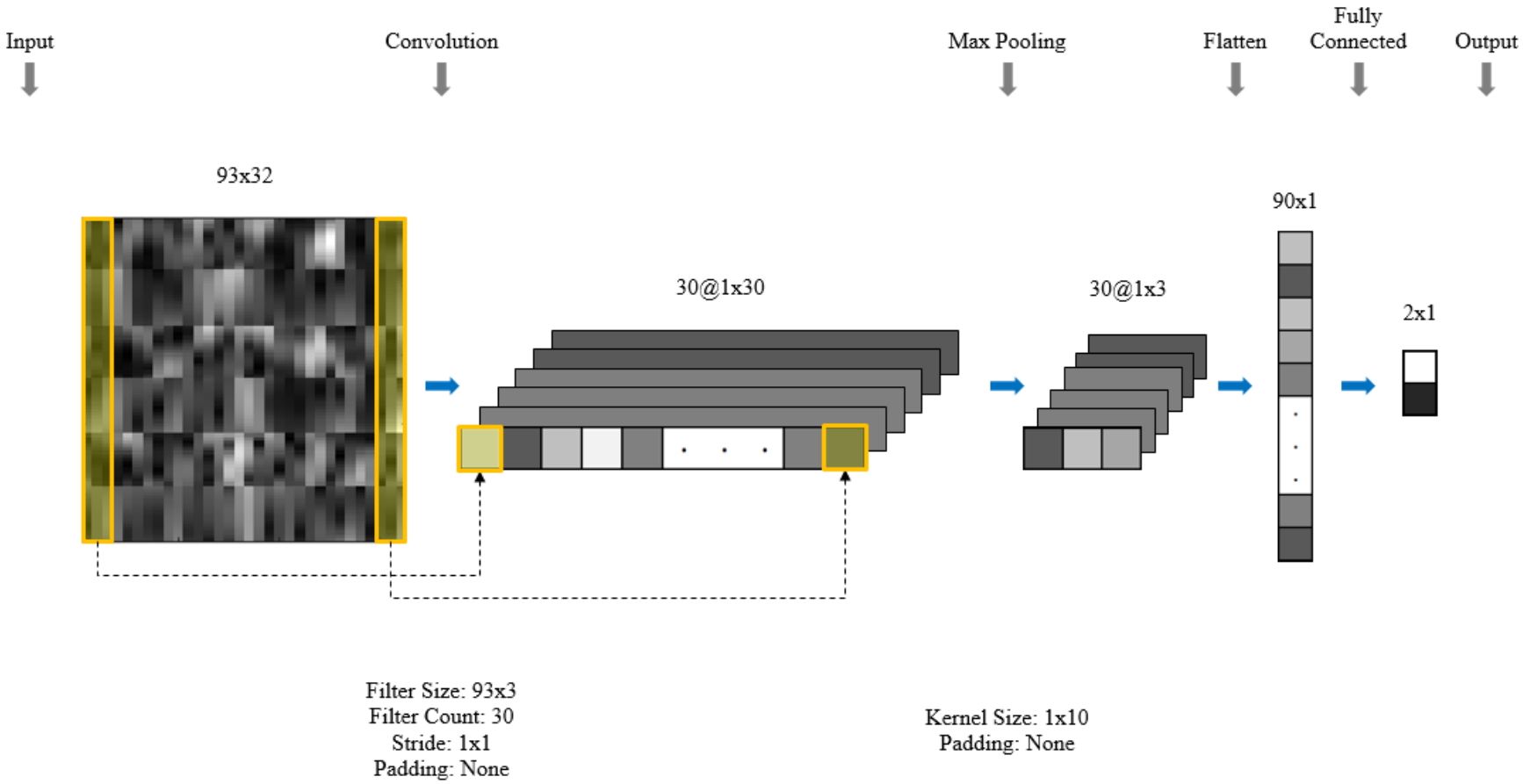

Figure 4.1. CNN structure

4.2 DFE with Reconstruction and Clustering Losses

4.2.1 General Structure

First version of DFE (DFEv1) uses reconstruction and clustering method together. This method is inspired from [14]. It includes forming embedded representations of input images through a convolutional autoencoder (CAE) and using these embedded features for cluster training in unsupervised manner.

In Figure 4.2, the general structure of DFEv1 is given. As can be seen from the figure, input image x_i is encoded to form embedded features y_i, then the image is reconstructed ($\tilde{x}_i$) from these embedded features. This is called the reconstruction part. At first, only reconstruction part of the network is trained which is referred as pre-training. During pre-training, MSE loss is used. We define MSE loss as follows:

$$L_R = MSE = \frac{1}{n}\sum_{p=1}^{n}(x_{ip} - \tilde{x}_{ip})^2 \tag{4.1}$$

MSE measures the average squared difference between the original image and the reconstructed image. Here n denotes the number of pixels in input image, x_{ip} is the p^{th} pixel of i^{th} input image and $\tilde{x}_{ip}$ is the corresponded pixel in the reconstructed image.

After pre-training is finished and adequate representations of input images in the embedding layer are obtained, clustering part of the network is introduced. To do so, first thing is initializing cluster centers μ_j. Cluster centers are vectors in the same size with embedded features y_i and they are initialized by applying k-means on embedded features. K-means is only applied once to initialize cluster centers.

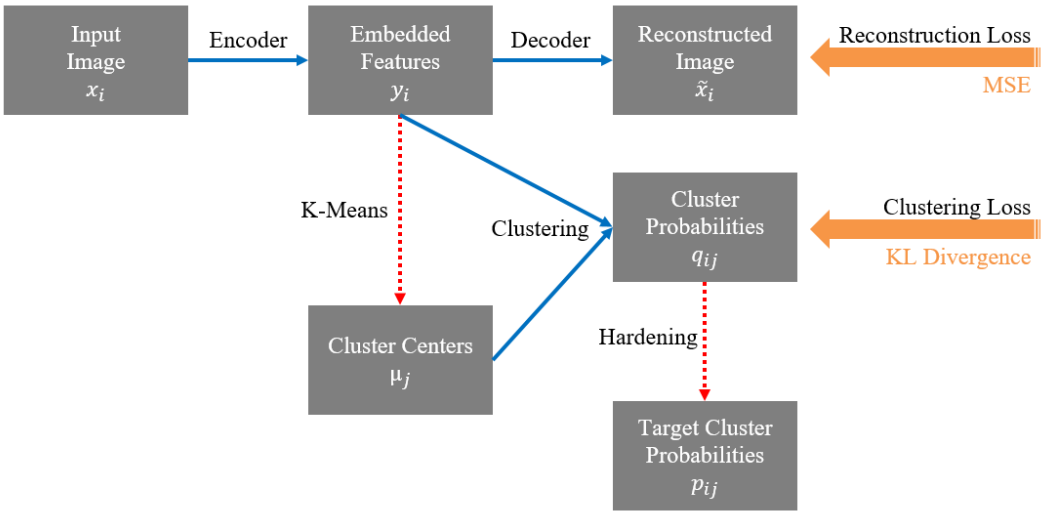

Figure 4.2. General structure of DFEv1

In the clustering part of the network, clustering probabilities Q are calculated which shows the probability of embedded features y_i belonging to j[th] cluster (μ_j). These probabilities are calculated by Student's t-distribution:

$$q_{ij} = \frac{(1 + \|y_i - \mu_j\|^2)^{-1}}{\sum_j (1 + \|y_i - \mu_j\|^2)^{-1}} \tag{4.2}$$

For the existing distribution Q we calculate a target distribution P which is the hardened form of Q. Hardening means the probability distribution of one image (x_i) is less scattered among clusters (μ_j). In other words, it tries to maximize the probability of an input belonging to one cluster and minimize the probabilities of belonging to other clusters. Target distribution P is calculated and updated as follows at every T iterations:

$$p_{ij} = \frac{q_{ij}^2 / \sum_i q_{ij}}{\sum_j (q_{ij}^2 / \sum_i q_{ij})} \tag{4.3}$$

Training objective in clustering part is to draw Q closer to P by learning cluster centers and encoder weights. Therefore clustering loss is defined as the difference

between these two distributions. To calculate the difference between distributions, Kullback-Leibler (KL) divergence is used. KL-divergence is calculated as follows:

$$L_c = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \quad (4.4)$$

As we mentioned, during pre-training only reconstruction loss is used. After pre-training is finished reconstruction and clustering losses are used jointly. In Equation 4.5, joint loss function is given. Here, γ is used to balance reconstruction and clustering losses and its value is determined empirically while adjusting gradually decreasing, smooth and eventually converging loss curves for both loss functions.

$$L = L_R + \gamma L_c \quad (4.5)$$

Idea of training reconstruction and clustering jointly is inspired from [14], but necessary changes are made to apply it to our case. Firstly, in [14] 2D convolution is preferred, however, as mentioned earlier in this chapter, convolution along vertical axis is not desired regarding the nature of our dataset. Therefore, we apply 1D convolution in DFEv1S. However, while using combined dataset, this condition differs. Although vertical axis contains frequency information, it also contains spatial information obtained from different electrodes. This property of images allows us to apply convolution along the vertical axis, too. In DFEv1C, we choose filter length 31 in vertical axis so that it does not convolves frequency information, but it convolves spatial (electrode) information. Another difference from [14] is that instead of 3 convolutional layers, 2 convolutional layers are used in encoder/decoder. This is because of the larger filter size we choose in the first layer leads to smaller second layer and reduces the need for another layer. Furthermore, we do not apply flattening between encoder and decoder as [14] does. Instead we apply flattening before clustering. This is owing to that we intuitively want to encode only frequency information in CAE and our experiments verifies our intuition. Last difference of our DFEs is that we introduce max pooling before flattening layer which is not used in [14]. We make this change considering our input format and CNN implementation (described in 4.1) both of which are adopted from [1]. In this network, max pooling

is preferred after convolutional layer and it performs well with the preferred input format. Therefore, we also add a max pooling layer, which is not used in [14].

There are two different subversions of DFEv1 depending on the dataset it uses. DFEv1C and DFEv1S use single electrode image dataset (3.2.2) and combined 3-electrode dataset (3.2.3), respectively.

4.2.2 DFEv1C: Using Combined 3-Electrode Dataset

Encoder of DFEv1C consists of 2 convolutional layers. First convolutional layer accepts input images in size of 93x32 from combined 3-electrode dataset. 32 filters in size of 31x3 are applied in this layer. We choose filter size in vertical axis 31 to cover all frequency information in one filter and filters are shifted 31 by 31 in this axis in order for jumping to other electrode information. Filters stride by ones in horizontal axis and as a result, 3x30 outputs are obtained. After convolution, ReLU is used for nonlinearity. In the second convolutional layer, 64 filters in size of 3x1 are applied with a stride 1x1 which produces 1x30 sized outputs and again ReLU is used after convolution.

Decoder of DFEv1C consists of 2 deconvolutional layers. These layers are reverse of the convolutional layers implemented in encoder. After the second deconvolutional layer sigmoid function is preferred to ensure that pixel values are in between 0 and 1 in the reconstructed image. MSE loss is defined between the reconstructed image and original input image. Optimization of reconstruction loss updates the weights of encoder and decoder layers.

In order to obtain embedded features, first max pooling is applied to 1x30 sized 64 images and then flattening is applied. Since the kernel size is 1x10 in max pooling, resulting size becomes 192x1. We expect clustering part to learn left and right hand movement MI signals from these embedded features in an unsupervised manner. Therefore, we define 2 clusters corresponding to these 2 classes. Each of these clusters is the same size with embedded features. As a result, clustering centers

contain 192x2 points and these are trainable parameters. At the beginning of the clustering training, these clustering centers are initialized with K-means applied to all embedded features extracted from all input images.

Using both embedding features and clustering centers according to Student's t-distribution, cluster probability distributions are calculated for all images. As mentioned in 4.2.1, from this distribution a target distribution is calculated. Target distribution is not calculated in every epochs. Since target distribution is calculated from cluster probability distribution and it is used to update cluster probability distribution, updating target distribution in every epochs bring on instability. To avoid this instability, target distribution is calculated in every 7 epochs, which is large enough to avoid instability and small enough to prevent clustering loss function to converge early. KL divergence loss is defined between the target and existing distributions. Optimization of clustering loss updates the weights of encoder and clustering centers. In DFEv1C γ value in equation 4.5 is set to 1 empirically.

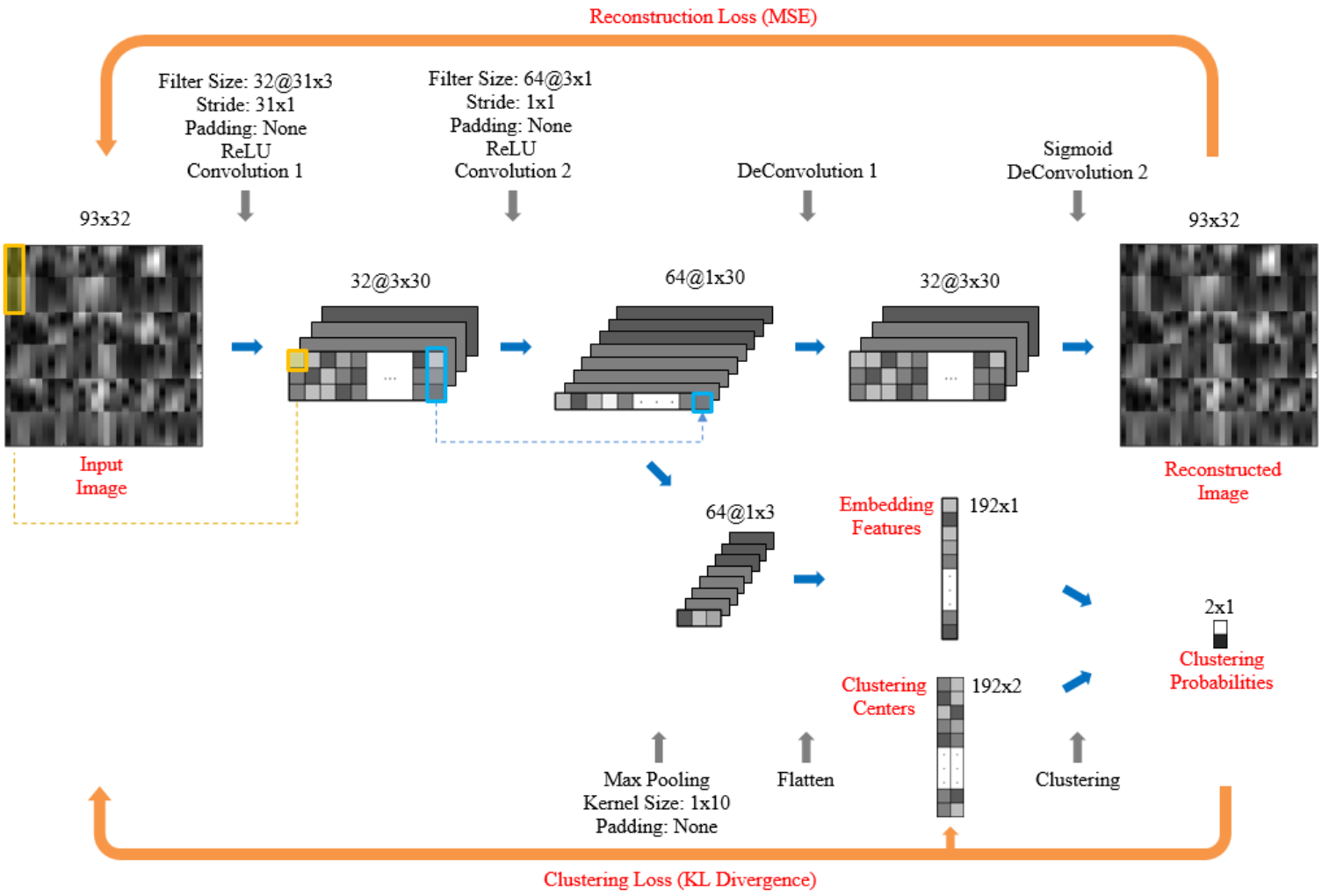

Figure 4.3. DFEv1C implementation

After DFEv1C training is completed, it can be used as a feature extractor for EEG MI signals. DFEv1C takes 93x32 images as input and produces 192x1 embedded features. In order to measure the quality of these features, another network is designed that takes embedded features as inputs and classifies them. The classifier is shown in Figure 4.4. It is a three layered MLP network with 64 and 32 nodes in hidden layers. It produces soft labels at output layer and cross entropy loss is used to train this network:

$$L = CE = -\sum_{c=1}^{2} t_{ic} \log s_{ic} \tag{4.6}$$

Here c is the class number and t_{ic} denotes the true label for the i^{th} input. It is equal to 1 if i^{th} input belongs to c^{th} class, 0 otherwise. s_{ic} is the softmax output, and it is the probability produced by the network showing the probability of i^{th} input belonging to c^{th} class.

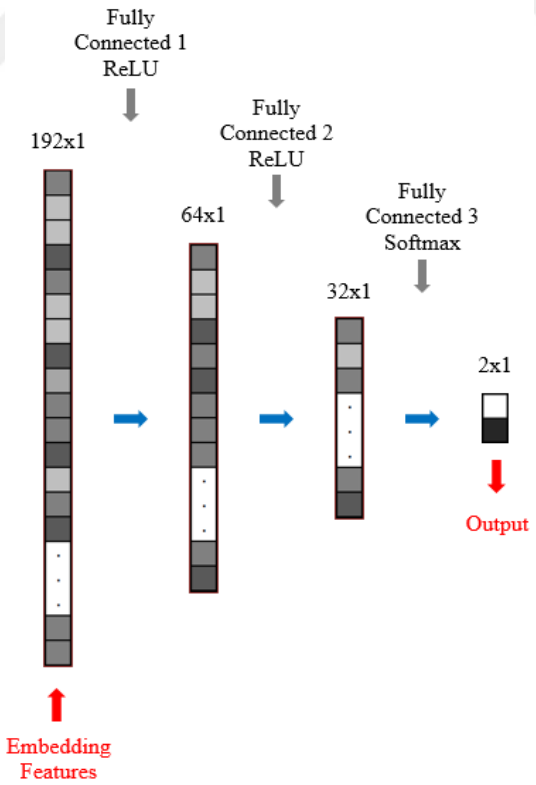

Figure 4.4. MLP classifier using features generated with DFEv1C/DFEv2C

4.2.3 DFEv1S: Using Single Electrode Dataset

DFEv1S is similar to DFEv1C, but is uses single electrode dataset as input. We give differences of DFEv1S from DFEv1C in this section.

First convolutional layer of DFEv1S accepts input images in size of 31x32 from single electrode dataset. 16 filters in size of 31x3 are applied in this layer. We choose filter size in vertical axis 31 so that filters stride only along horizontal axis. Also in horizontal axis 1 unit zero padding is added. As a result, 1x32 sized outputs are produced in this layer. In the second convolutional layer, 32 filters in size of 1x3 are applied with a stride 1x1 which produces 1x30 sized outputs.

In the clustering part of the network, we choose number of clusters 3. Because, we expect that in one cluster c3 left hand images and c4 right hand images will gather due to symmetry. In the second cluster c4 left hand images and c3 right hand images will gather and in the last cluster cz left/right images will gather. By choosing number of clusters 3 and having 96x1 sized embedded features, clustering centers become in size of 96x3 and the output layer becomes in size of 3x1. In DFEv1S γ value in equation 4.5 is defined 10.

After training is completed, DFEv1S can be used as a feature extractor. However, features extracted by this network contain information only from one electrode. In order to measure the quality of these features, like DFEv1C a classifier network is designed. This time we designed a CNN (Figure 4.6) that uses embedded features of 3 electrodes. Embedded feature size is 96x1, and when we combine features from 3 electrodes the input size of the classifier becomes 96x3. Since horizontal axis of these inputs carries spatial information, in the first layer convolution is applied with filters in size of 1x3. Output of this layer is in size of 96x1. Then we flatten outputs of the 8 filters and obtain 768x1 vector. Rest of the classifier network contains 3 fully connected layers with 128 and 32 nodes in hidden layers. Like DFEv1C feature classifier, CE loss is used during training.

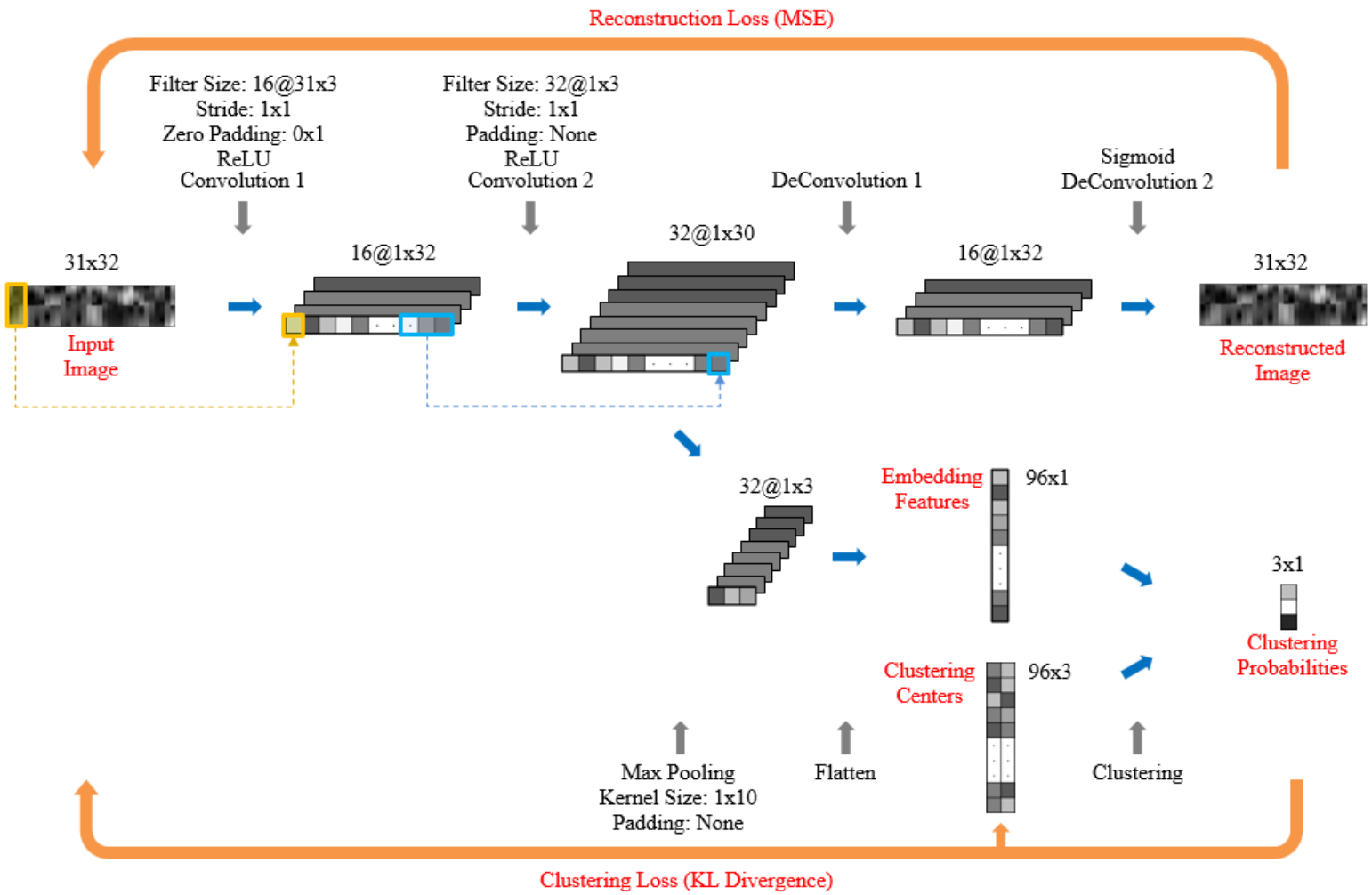

Figure 4.5. DFEv1S implementation

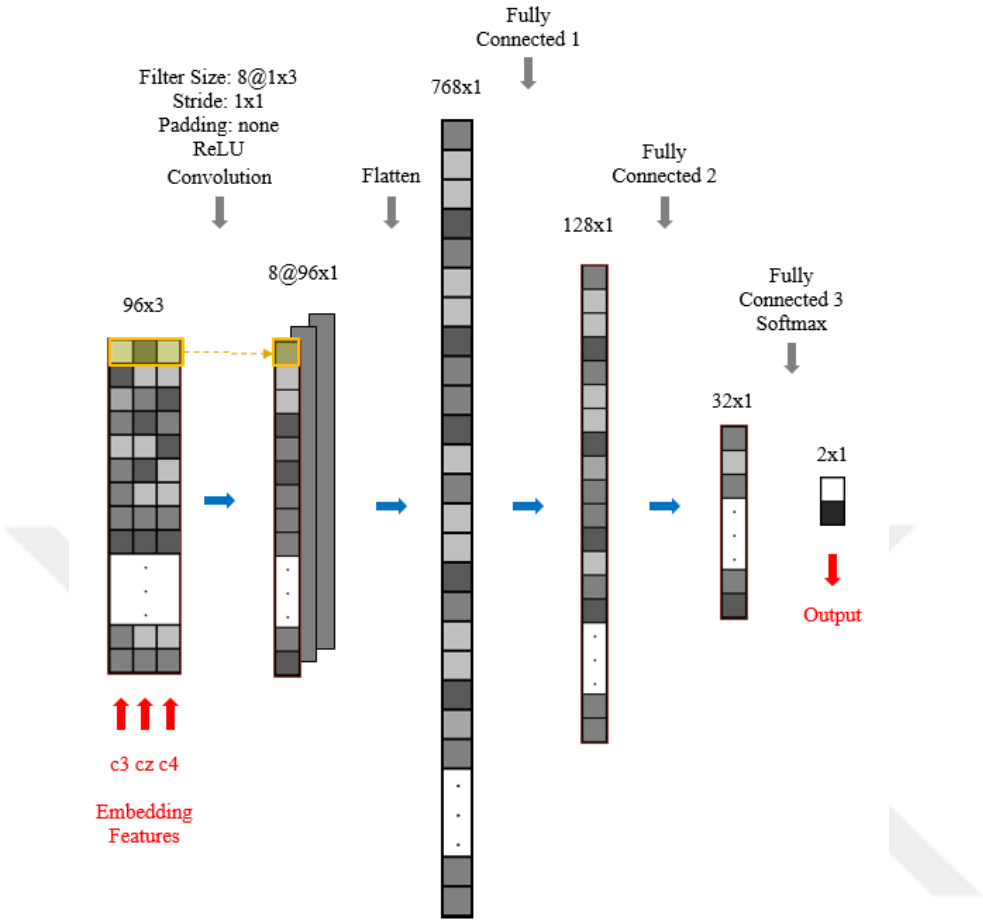

Figure 4.6. CNN classifier using features generated with DFEv1S/DFEv2S

4.3 Reconstruction and Classifying

4.3.1 General Structure

Second version of DFE (DFEv2) uses reconstruction and classifying method. In this method we replace clustering in DFEv1 with classifying. It includes forming embedded representations of input images through a convolutional autoencoder (CAE) and then classifying these embedded features in supervised way.

In Figure 4.7, general structure of DFEv2 is given. Like DFEv1, input image x_i is encoded to form embedded features y_i, then the image is reconstructed ($\tilde{x}_i$) from

these embedded features in the reconstruction part. Same as DFEv1, only reconstruction part of the network is trained first in pre-training with MSE loss function. After pre-training is finished, we introduce classification loss and network continues training while optimizing two losses jointly. Outputs of the classification part are soft label predictions (s_i) and we choose CE for classification loss as defined in the formula (4.6).

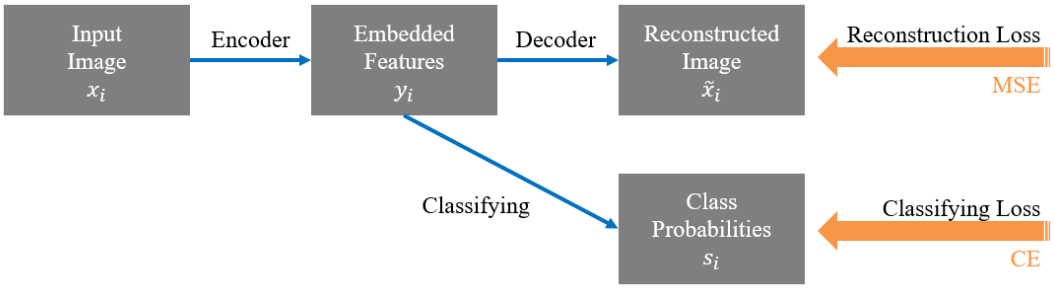

Figure 4.7. General structure of DFEv2

There are two different subversions of DFEv2 depending on the dataset it uses. DFEv2C and DFEv2S use single electrode image dataset (3.2.2) and combined 3-electrode dataset (3.2.3) respectively.

4.3.2 DFEv2C: Using Combined 3-Electrode Dataset

Reconstruction part of the DFEv2C is exactly the same as DFEv1C. Max pooling and flatten layers which produce embedded features are also the same as DFEv1C. Therefore, embedding layer contains 192x1 points. In DFEv2C, instead of using trainable cluster centers and unsupervised clustering layer, we use a FC layer that outputs soft classification results. Soft classification results provide a probability distribution of input images belonging to each class. Classification part of the network is trained with CE loss and it updates the weights of the encoder and FC layer. In order to balance reconstruction and classification losses, we take $\gamma = 10$.

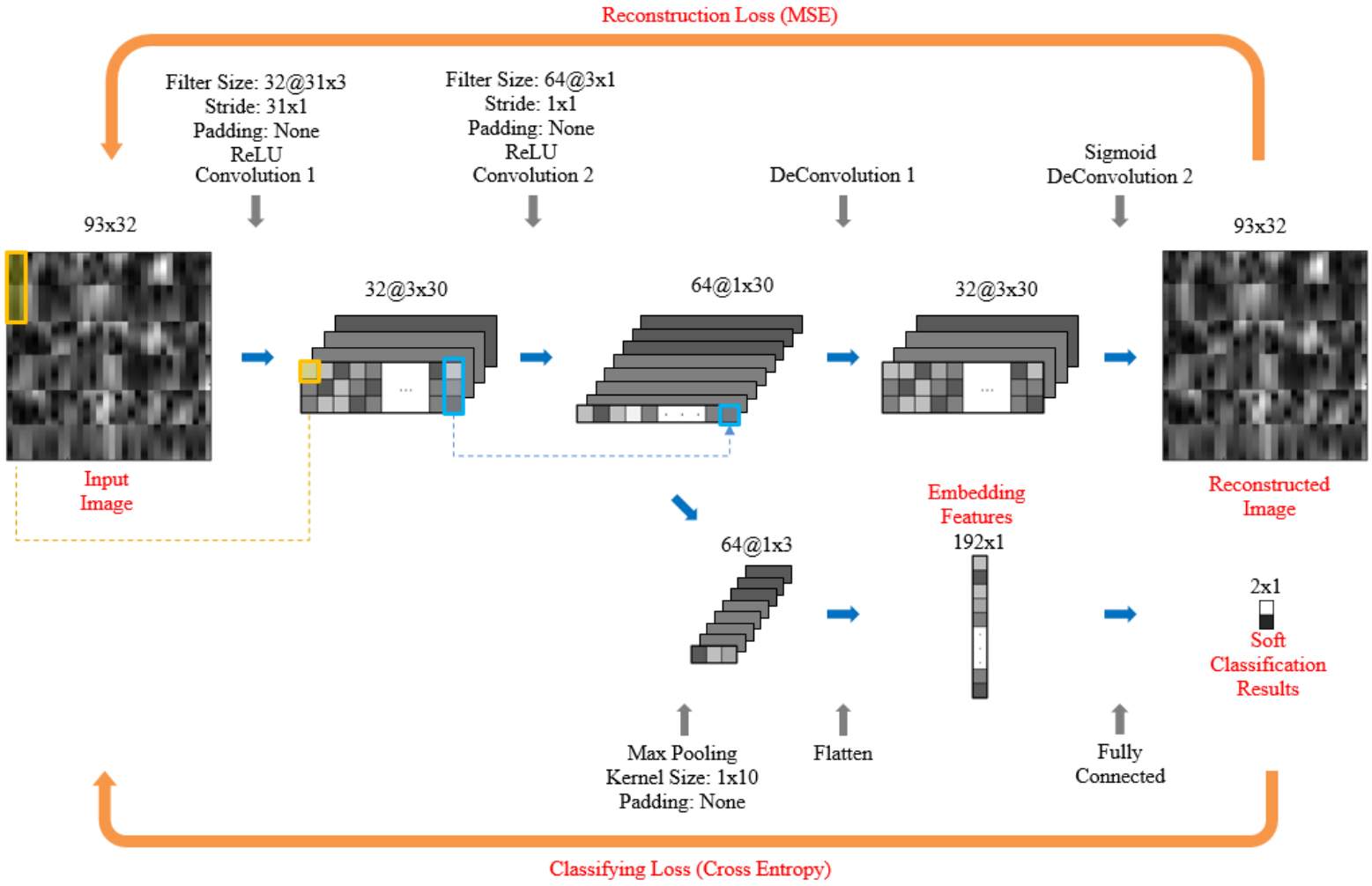

Figure 4.8. DFEv2C implementation

Since the size of the embedded feature vector is the same size as DFEv1C, MLP classifier given in Figure 4.4 can also be used for DFEv2C. The feature quality will be measured according to classification performance of this classifier.

4.3.3 DFEv2S: Using Single Electrode Dataset

Reconstruction part of the DFEv2S is exactly the same as DFEv1S. Max pooling and flatten layers which produce embedded features are also the same as DFEv1S. Therefore, embedding layer contains 96x1 points. In DFEv2S, instead of using trainable cluster centers and unsupervised clustering layer, we use a FC layer that outputs soft classification results. For the classification, we define six labels: c3 left hand, c3 right hand, cz left hand, cz right hand, c4 left hand and c4 right hand. Different from DFEv1S which has three clusters, we choose the number of classes as six. Because, during our experiments with DFEv1S, we see that clustering cannot distinguish these classes, and adding more clusters makes clustering task even more difficult, and finally lowers the performance. However, for DFEv2S, training is held in supervised manner, which increases network's learning capacity and the ability of distinguishing smaller classes. Therefore, we define more classes in DFEv2S compared to clusters in DFEv1S. This part of the network is trained with CE loss and it updates the weights of the encoder and FC layer. In order to balance reconstruction and classification losses, we choose $\gamma = 1$.

Size of the embedded features that DFEv2S produces is 96x1 like DFEv1S. Therefore CNN classifier given in Figure 4.6 can also be used to classify the embedded features of DFEv2S. The feature quality will be measured according to classification performance of this classifier.

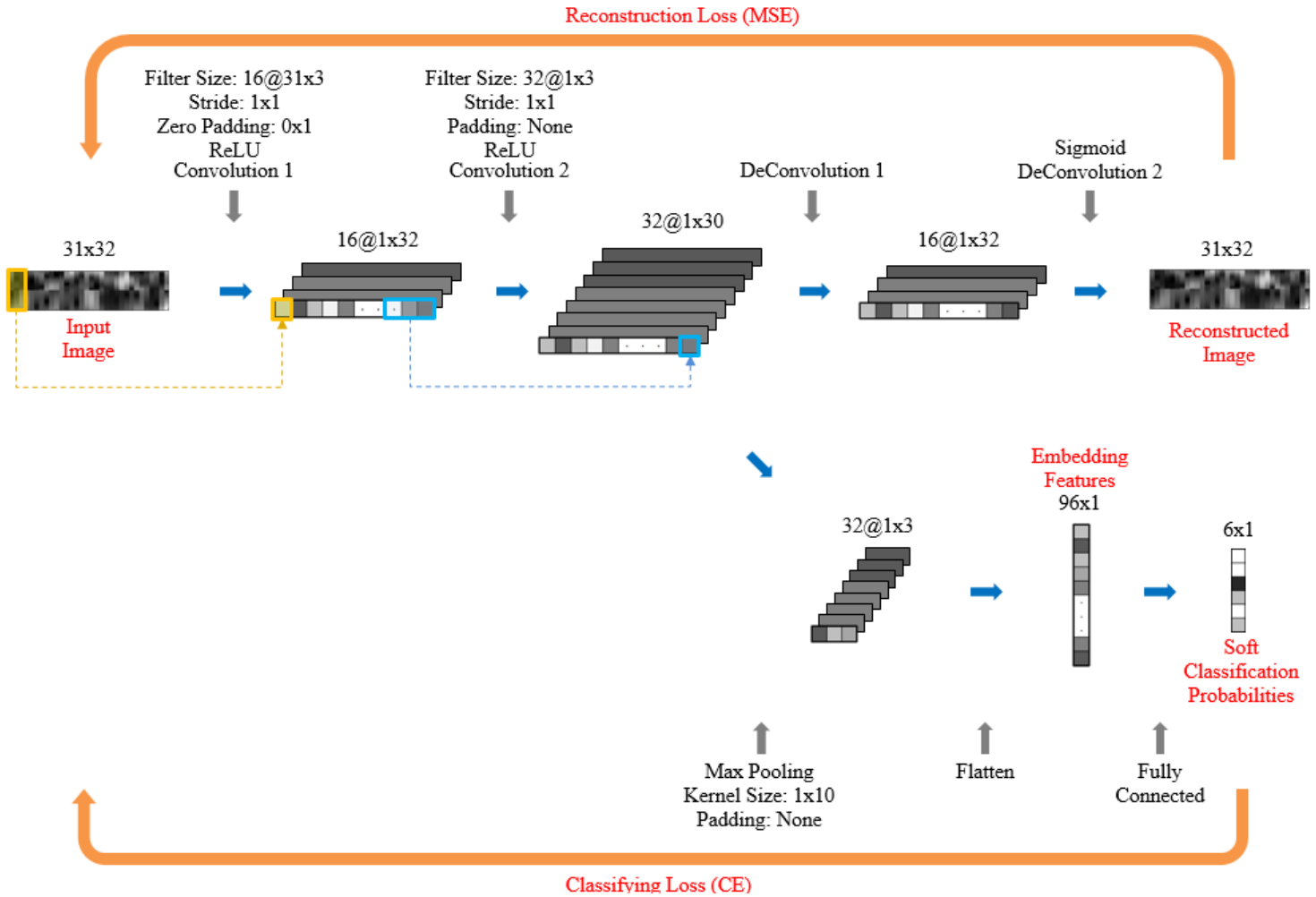

Figure 4.9. DFEv2S implementation

4.4 Reconstruction Only

4.4.1 General Structure

Third version of DFE (DFEv3) uses only reconstruction method. It includes forming embedded representations of input images through a convolutional autoencoder (CAE).

In Figure 4.10, general structure of DFEv3 is given. Like DFEv1, input image x_i is encoded to form embedded features y_i, then the image is reconstructed ($\tilde{x}_i$) from these embedded features.

Figure 4.10. General structure of DFEv3

There are two different subversions of DFEv3 depending on the dataset it uses. DFEv3S and DFEv3C use single electrode image dataset (3.2.2) and combined 3-electrode dataset (3.2.3), respectively.

4.4.2 DFEv3C: Using Combined 3-Electrode Dataset

Encoder of DFEv3C consists of 2 convolutional layers. First convolutional layer accepts input images in size of 93x32 from combined 3-electrode dataset. 32 filters in size of 31x3 are applied in this layer with 1 zero padding in horizontal axis. This convolutional filtering produces 3x32 sized outputs. After convolution, ReLU is used for nonlinearity. In the second convolutional layer, 64 filters in size of 3x1 are applied with a stride 1x1 which produces 1x32 sized outputs and again ReLU is used after convolution.

Decoder of DFEv3C consists of 2 deconvolutional layers. These layers are reverse of the convolutional layers implemented in encoder. After the second deconvolutional layer sigmoid function is preferred to ensure that pixel values are in between 0 and 1 in the reconstructed image. MSE loss is defined between the reconstructed image and original input image. Optimization of reconstruction loss updates the weights of encoder and decoder layers.

Size of the embedded features produced by DFEv3C is 1x32. However, different feature vectors are obtained with 64 different filters. To be able to use these features in the classifier network we combined 64 channels and obtain feature matrices in size of 64x32 as shown in Figure 4.12. In this figure we give the details of the classifier that use features extracted with DFEv3C. The classifier network is a CNN with 1 convolutional, 1 max pooling, 1 flattening and 1 fully connected layers. In the convolutional layer 64x3 sized 30 filters are applied and 1x30 sized outputs are generated, then after max pooling layer with kernel size 1x10, this size is reduced to 1x3. When we flatten the resulting layer, size of the final layer before FC layer becomes 90x1. Fully connected layer with softmax function produces soft classification outputs. CE loss is defined between the soft classification results and true labels.

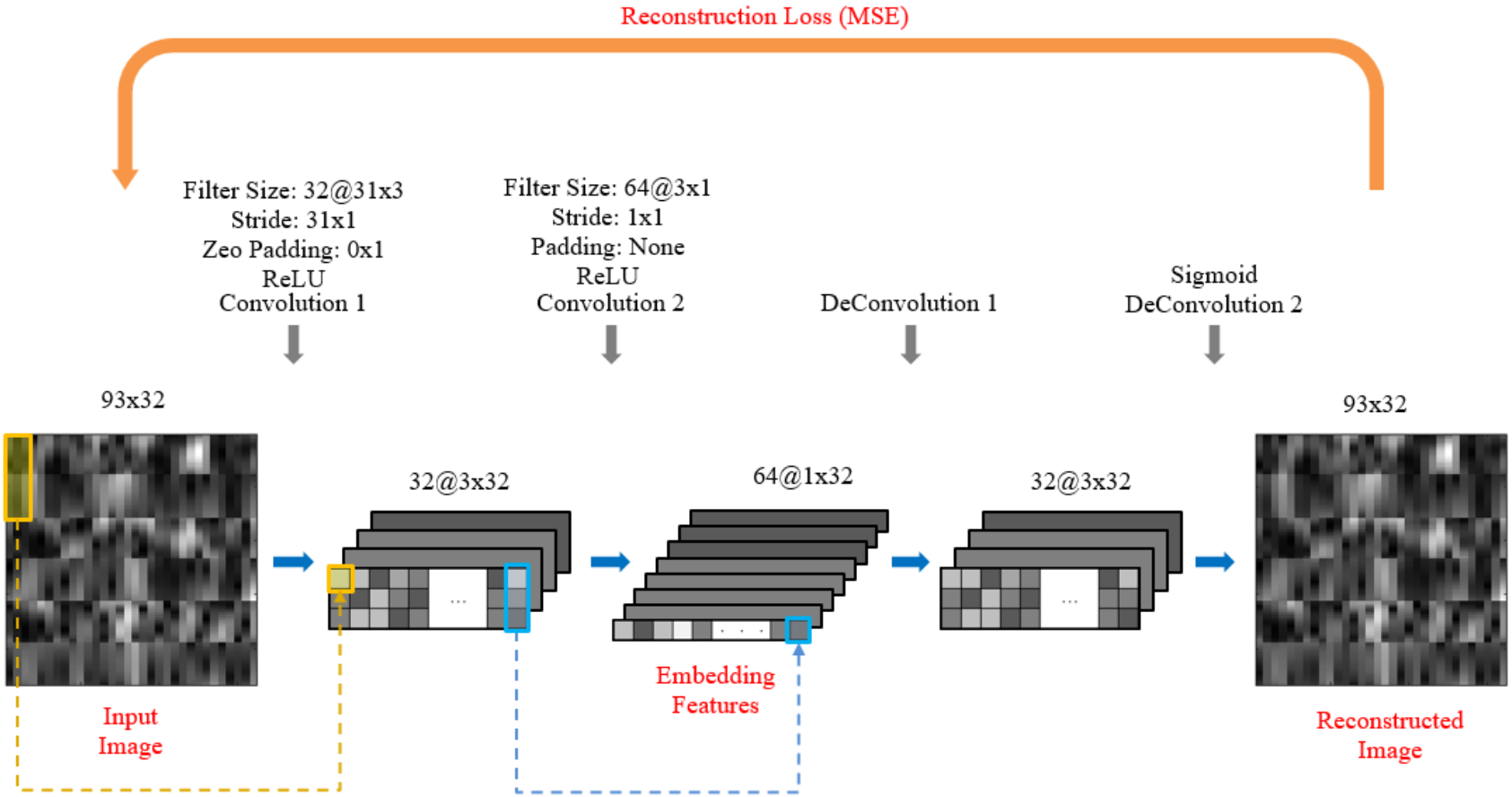

Figure 4.11. DFEv3C implementation

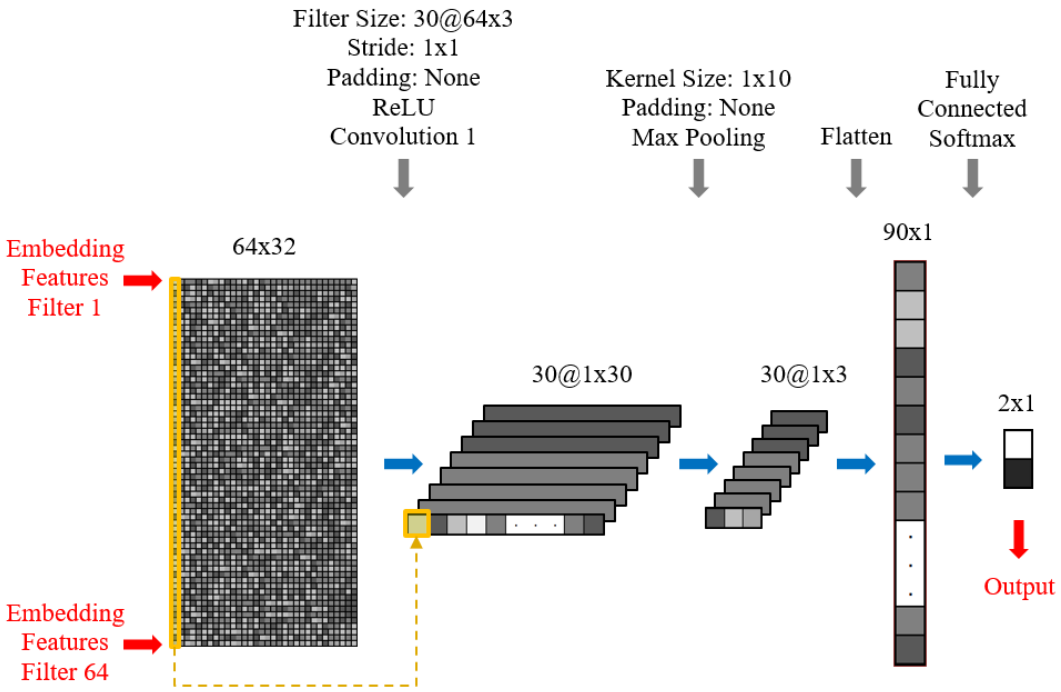

Figure 4.12. CNN classifier using features generated with DFEv3C

4.4.3 DFEv3S: Using Single Electrode Dataset

Encoder of DFEv3S consists of 2 convolutional layers. First convolutional layer accepts input images in size of 31x32 from single electrode dataset. 8 filters in size of 31x3 are applied in this layer with 1 zero padding in horizontal axis. This convolutional filtering produces 1x32 sized outputs. After convolution, ReLU is used for nonlinearity. In the second convolutional layer, 16 filters in size of 1x3 are applied with a stride 1x1 which produces 1x30 sized outputs and again ReLU is used after convolution.

Decoder of DFEv3S consists of 2 deconvolutional layers. These layers are reverse of the convolutional layers implemented in encoder. After the second deconvolutional layer sigmoid function is preferred to ensure that pixel values are in

between 0 and 1 in the reconstructed image. MSE loss is defined between the reconstructed image and original input image. Optimization of reconstruction loss updates the weights of encoder and decoder layers.

Size of the embedded features produced by DFEv3S is 1x30. However different feature vectors are obtained with 16 different filters. Since the features generated from this network only contains one electrode information, we combined the features of 3 electrodes in the classifier network as in Figure 4.14. It can be seen from the figure that we preserve the channel separation during concatenation of these features. Therefore size of the input layer of the classifier becomes 3x30 with 16 channels. The classifier network is a CNN with 1 convolutional, 1 max pooling, 1 flattening and 1 fully connected layers. In the convolutional layer 3x1 sized 32 filters are applied and 1x30 sized outputs are generated, then after max pooling layer with kernel size 1x10, this size is reduced to 1x3. When we flatten the resulting layer, size of the final layer before the FC layer becomes 96x1. Fully connected layer with softmax function produces soft classification outputs. CE loss is defined between the soft classification results and true labels.

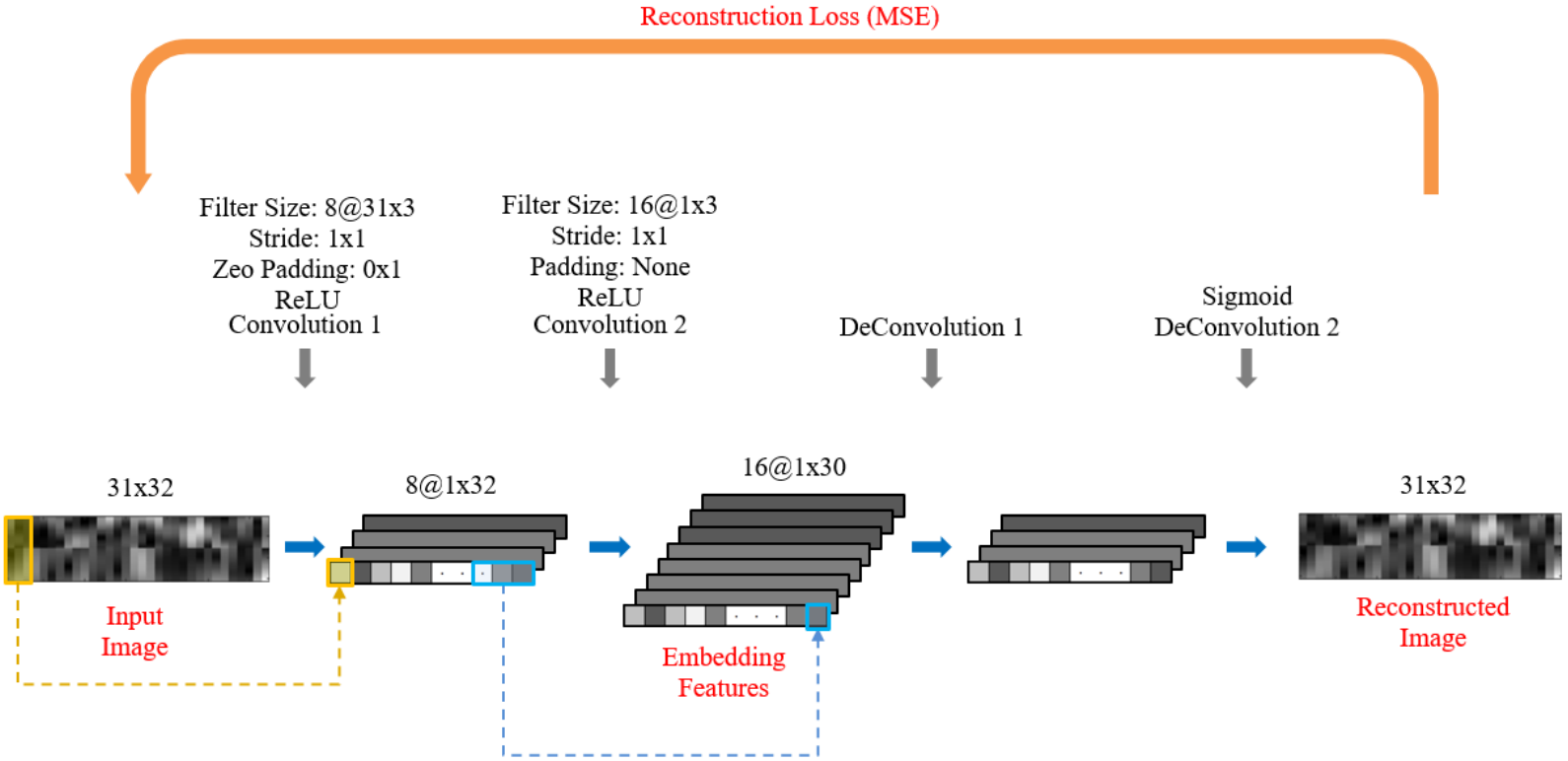

Figure 4.13. DFEv3S implementation

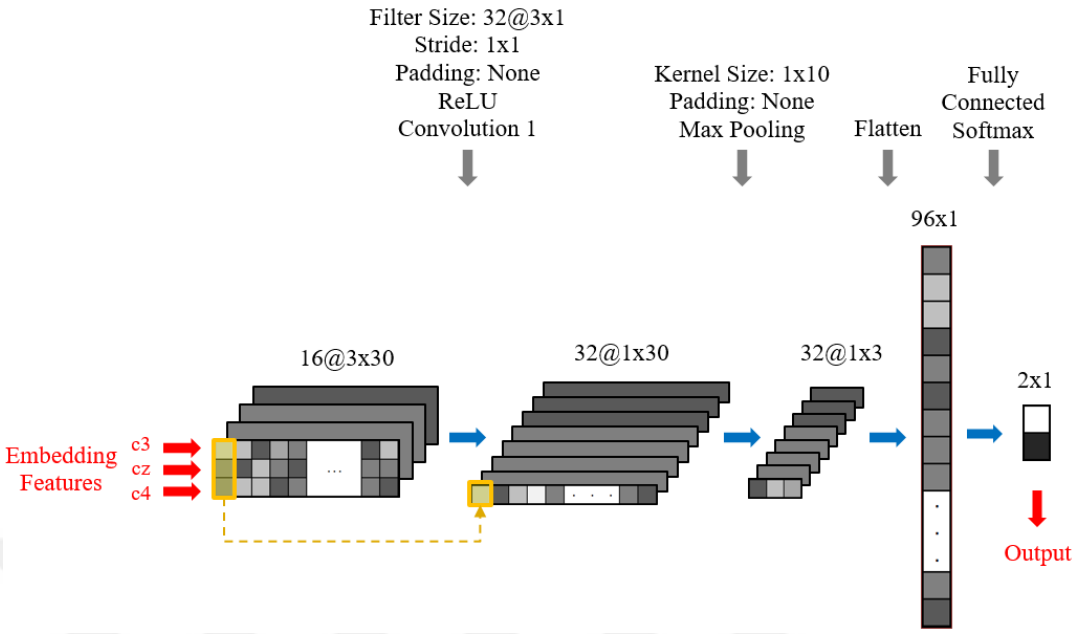

Figure 4.14. CNN classifier using features generated with DFEv3S

CHAPTER 5

EXPERIMENTS

Chapter 3 gives detailed information about datasets we use in this study. Chapter 4 gives the deep learning structures that we design for feature extraction and classifier networks that use these features. In this chapter, we present experiments that are held using these datasets and feature extractors (DFEs). This chapter is organized as follows: First data distributions are explained, then details of the training processes are given. After that we share results obtained from all DFEs and analyze their success rates. Beside from these these, we also give how whitening and augmentation affects the results. Finally we compare our results to state of the art solutions for EEG MI event classification.

5.1 Data Distributions

Table 3.1 gives how many images are generated from publicly available datasets we found for EEG MI signals. In our experiments, we split this data into 3 groups: First group of data is used for the training of DFE network. After DFE network is trained, it is used as feature extractor. Second group of data is used for the training of the classifier which uses features extracted by DFEs and third group of data is used to measure classification performance of the feature classifiers trained with extracted features.

As mentioned in 2.2.1, the most popular dataset used in EEG MI classification is set-7. Therefore, in our experiments, we prefer showing the performance of DFEs using set-7. In 2.2.1.3 we explain that there are different approaches how researchers distributes the data in set-7 among training and test purposes. In this work, we adopt two different data distributions. These distributions are summarized in Table 5.1.

Table 5.1 Data distributions

	Distribution 1	*Distribution 2*
DFE Training Data	All sets except set-7	All training sessions of set-7
Classifier Training Data	Training sessions of a subject in set-7	Training sessions of a subject in set-7
Classifier Test Data	Test sessions of a subject in set-7	Test sessions of a subject in set-7

5.1.1 Distribution 1

In distribution 1, we train DFE networks with left/right hand MI images that are obtained from all sets except set-7 (sets 1, 2, 3, 4, 5, 6, 8, 9, 11, 12). Total number of images that are used from these datasets is 66308 without augmentation (This number is given for combined 3-electrode dataset. For single electrode dataset this number triples). With augmentation this number goes much higher.

After training DFE, this trained DFE is used as a feature extractor on set-7. In this set, there are 9 subjects and the classifier is trained and tested separately for each of these subjects. There are five sessions of recordings for each subject. First three of these sessions are referred as training sessions, while other two sessions are referred as test sessions. During the training of the classifier, training sessions are used and the performance of the classifier is tested with test sessions. In Table 5.2, training and test image counts for each subject are given.

5.1.2 Distribution 2

In distribution 2, we train DFEs with images that are obtained from training sessions of all subjects in set-7 (3680 images without augmentation). Data distribution during training of the classifier is the same as distribution 1.

Table 5.2 Training and test image counts for different subjects of set-7

Subject ID	*Training image count*	*Test image count*
ID 1	400	320
ID 2	400	280
ID 3	400	320
ID 4	420	320
ID 5	420	320
ID 6	400	320
ID 7	400	320
ID 8	440	320
ID 9	400	320
Total	3680	2840

5.1.3 Augmentation

As explained in 3.4, we apply gaussian noise and circular shift augmentation jointly. There are two hyperparameters to be optimized in augmentation: gaussian noise variance and circular shift count. In our experiments, we get the best results when gaussian noise variance is equal to 0.005 and circular shift is equal to 5. Details of this tuning are given in 5.3.4.

We summarize the total image counts used in DFE training with/without augmentation is applied in Table 5.3. During the training of the feature classifier no augmentation is applied. Number of data used in feature classifier is summarized in Table 5.2 for each subject.

Table 5.3 Image counts used in DFE training

	Combined Set	*Single Electrode Set*
Distribution 1 w/o Augmentation	66.308	198.924
Distribution 1 with Augmentation	331.540	994.620
Distribution 2 w/o Augmentation	3.680	11.040
Distribution 2 with Augmentation	18.400	55.200

5.2 Training and Test Processes

As mentioned earlier, we have DFE networks to be trained. After DFE trainings are completed, we use DFEs for feature extraction. With extracted features we train feature classifier networks. In this subchapter, we give details of training process for these two types of networks.

5.2.1 DFE Training

In this section, we present some techniques used in training of the DFEs for faster and better convergence of training processes along with the tuned parameters related to given techniques.

5.2.1.1 Mini Batch Training

To train models on large datasets, machine learning and deep learning algorithms frequently use mini-batch training. In this method, the input is split into smaller subsets known as mini-batches, and instead of feeding inputs into the model all at once (batch training) or just one by one (online training), inputs are fed as mini-batches. This makes it possible for the model to be updated more frequently, which may speed up convergence and enhance generalization. In our experiments we set the mini batch size to 32.

5.2.1.2 Warm-Up Epochs

Pre-training part starts with warm-up epochs. The model's parameters are typically far from their ideal values in the early stages of training, and a high learning rate could lead the parameters to diverge quickly or make large, unstable changes. Warm-up solves this problem by slowly increasing the learning rate over a predetermined number of iterations or epochs from a small value to the desired rate. In DFE networks, during warm-up period, learning rate starts with 0.00005 and it increases to 0.0005 over 10 epochs.

5.2.1.3 Learning Rate Decay

During the training, we use Adam optimizer. Besides from using Adam, we decay the learning rate over time. The purpose of decreasing the learning rate over time is to improve convergence and fine-tune the model's performance as training progresses. To decrease learning rate, we define two variables: no_improvement_count and patience. We compare the loss value after each epoch with the lowest achieved loss during training. If the loss value of the current epoch is greater than the minimum loss, then no improvement occurs and we increase the no_improvement_count value by 1. If the loss value is smaller than the minimum loss, we decrease the no_improvement_count and we set the new minimum loss value to current loss. If no_improvement_count value is 0, we do not decrease the value to negative values. When learning gets slower and loss value stops decreasing, at some point no_improvement_count reaches the predetermined patience value. Each time this condition occurs, we set the learning rate value to half of itself and we reset no_improvement_count value. For the pre-training parts of DFEv1, DFEv2 and for DFEv3, patience value is set to 3. After pre-training of DFEv1 and DFEv2, patience value is set to 5. These patience values are obtained experimentally and give the optimum learning rate decay points for our network structures.

Training of DFEv2 networks includes updating target distribution as explained in 4.2. We update the target distribution at every 7 epochs. Since updating target distribution increases loss value immediately, we reset the value of no_improvement_count at these epochs.

5.2.1.4 Training Stop Condition

For DFEv1 and DFEv2 networks, training contains two parts: Pre-training (reconstruction) and training (clustering or classifying). In these networks, pre-training finishes when learning rate decays for the first time, or in other words, when no_improvement_count reaches the patience value for the first time.

After pre-training, training continues until one of these two conditions happens: 1) Training reaches 500 epochs, 2) Learning rate goes under 0.00001. For DFEv3 networks, there is no pre-training. Training continues until one of these two conditions is satisfied.

5.2.1.5 Loss Curves

In this section, we examine the loss curves obtained during the training of the DFEs and we explain particular patterns observed on these curves with techniques described in previous chapters.

5.2.1.5.1 DFEv1

In Figure 5.1, loss curves of DFEv1C and DFEv1S are given for both distribution 1 and distribution 2. These networks train reconstruction and clustering losses together. On the plots, we give loss curves of these losses separately. Blue series show reconstruction losses with axis values on the left side whereas red series show clustering losses with axis values on the right side. As can be seen from the plots that

clustering loss is introduced later in the training. In the graphs, “epochs” axis starts from 10. We avoid showing loss curve during warm-up period for the sake of clarity.

The effect of learning rate decay is visible on series (especially on blue series). When learning rate is decreased to half value, a sudden decrease occurs on loss values. This is because with smaller learning rate, better fine tuning can be done. Learning rate decay effect is more apparent in distribution 1.

Also the effect of target distribution updates can be seen on clustering loss (red) series. Since we update clustering loss target every 7 epochs, at this points sudden increases occurs in clustering losses. This is because we set the target distribution to much further at each update, and new target always becomes more difficult to reach. After each update, loss starts to decay until next update. This makes clustering loss curve to appear like a sawtooth waveform.

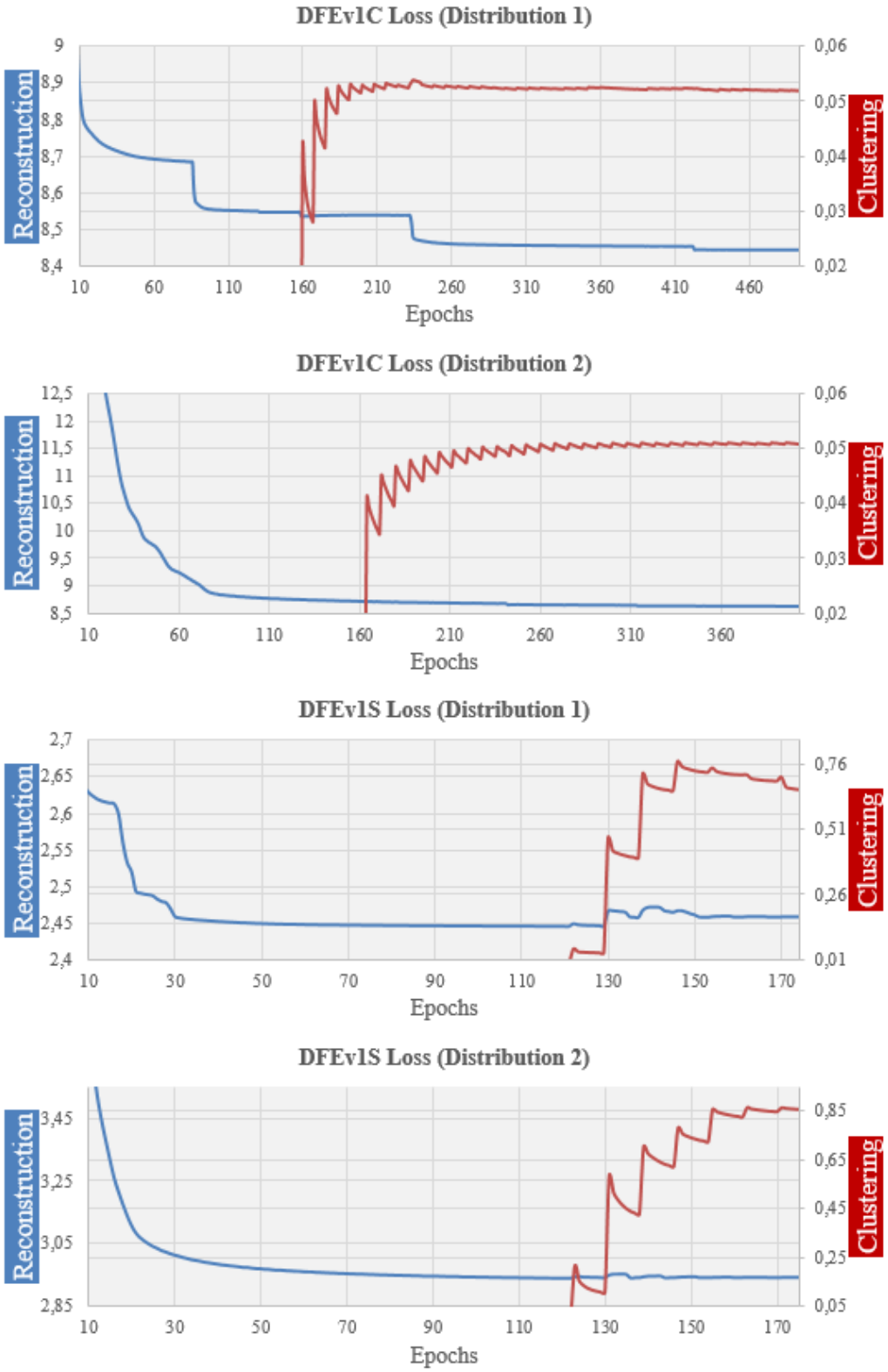

Figure 5.1. Loss change during DFEv1 training

5.2.1.5.2 DFEv2

In Figure 5.2, loss curves of DFEv2C and DFEv2S are given for both distribution 1 and distribution 2. These networks train reconstruction and classifying losses together. On the plots, we give loss curves of these losses separately. Blue series show reconstruction losses with axis values on the left side whereas red series show classifying losses with axis values on the right side. As can be seen from the plots that classifying loss in introduced later in the training. Like in previous subchapter, “epochs” axis starts from 10. We avoid showing loss curve during warm-up period for the sake of clarity.

Like explained in 5.2.1.5.1, the effect of learning rate decay is visible on loss series. Different from DFEv1, this time we observe slight increases in reconstruction loss, following the sudden decreases occurred at learning rate changing times. This increase is not surprising, considering the joint training of reconstruction and classifying. Total loss obtained from these two loss functions still decreases over epochs.

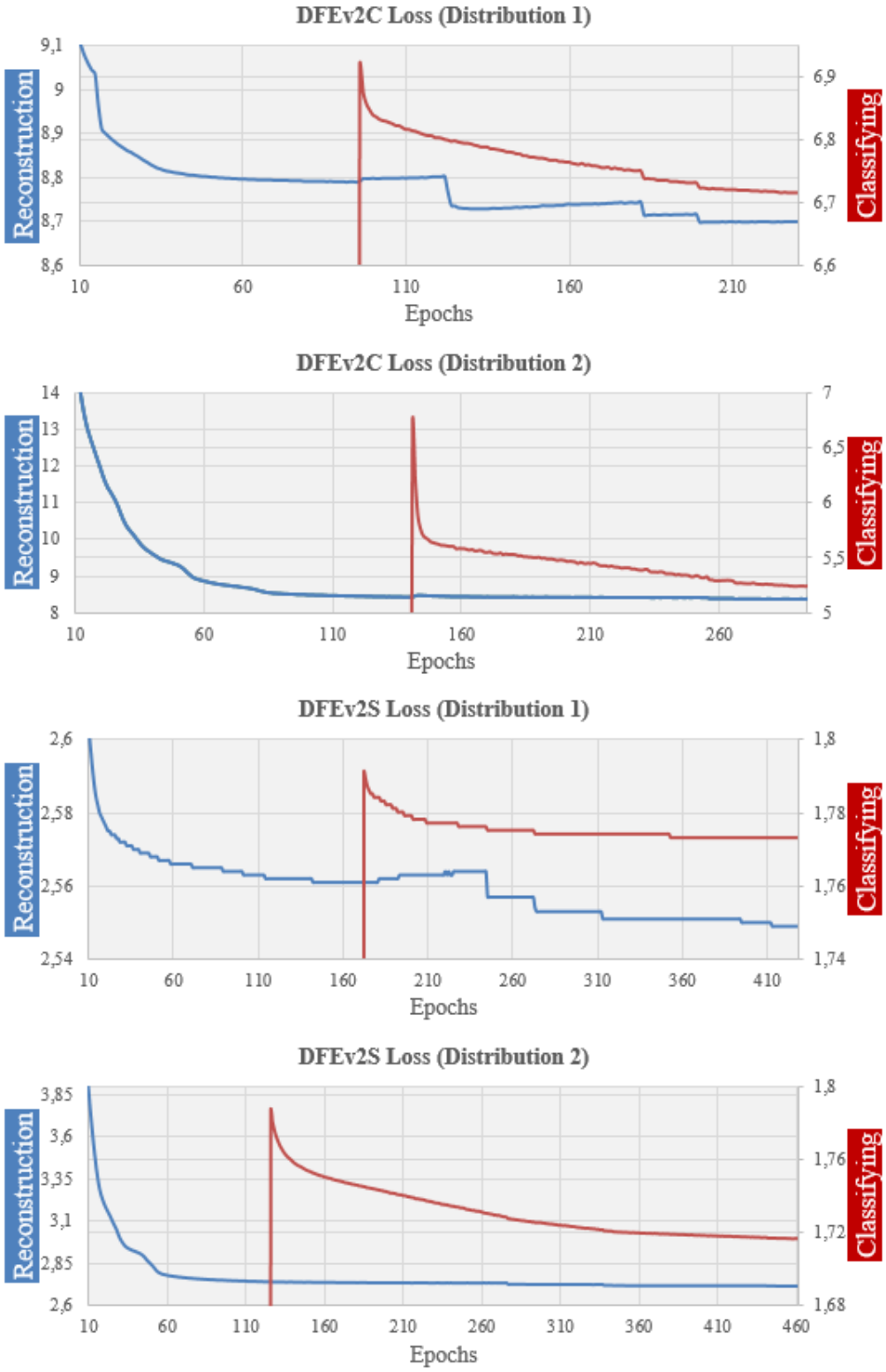

Figure 5.2. Loss change during DFEv2 training

5.2.1.5.3 DFEv3

In Figure 5.3, loss curves of DFEv3C and DFEv3S are given for both distribution 1 and distribution 2. These networks train only reconstruction loss. Like in previous subchapters, “epochs” axis starts from 10. As explained in 5.2.1.5.1, the effect of learning rate decay is visible on loss series.

5.2.2 Feature Classifier Training and Testing

In set-7, there are 9 subjects and as most of the studies working with this dataset, we train and test feature classifiers separately. For each of these 9 subjects we repeat training/testing 10 times and we give average results in 5.3. This approach is useful for obtaining a more robust estimate of the network's performance by reducing the impact of random variations or fluctuations in the training process.

During training sessions, we prefer using cross entropy loss function. We use Adam optimizer and we apply learning decay as in DFE trainings. We set initial learning rate to 0.0005 and final learning rate to 0.00001 with patience value 5. Training continues at least 100 epochs. After 100 epochs, if learning rate decays below 0.00001 or 500 epochs are reached, then the training stops. We implement mini batch training with batch size equals to 50.

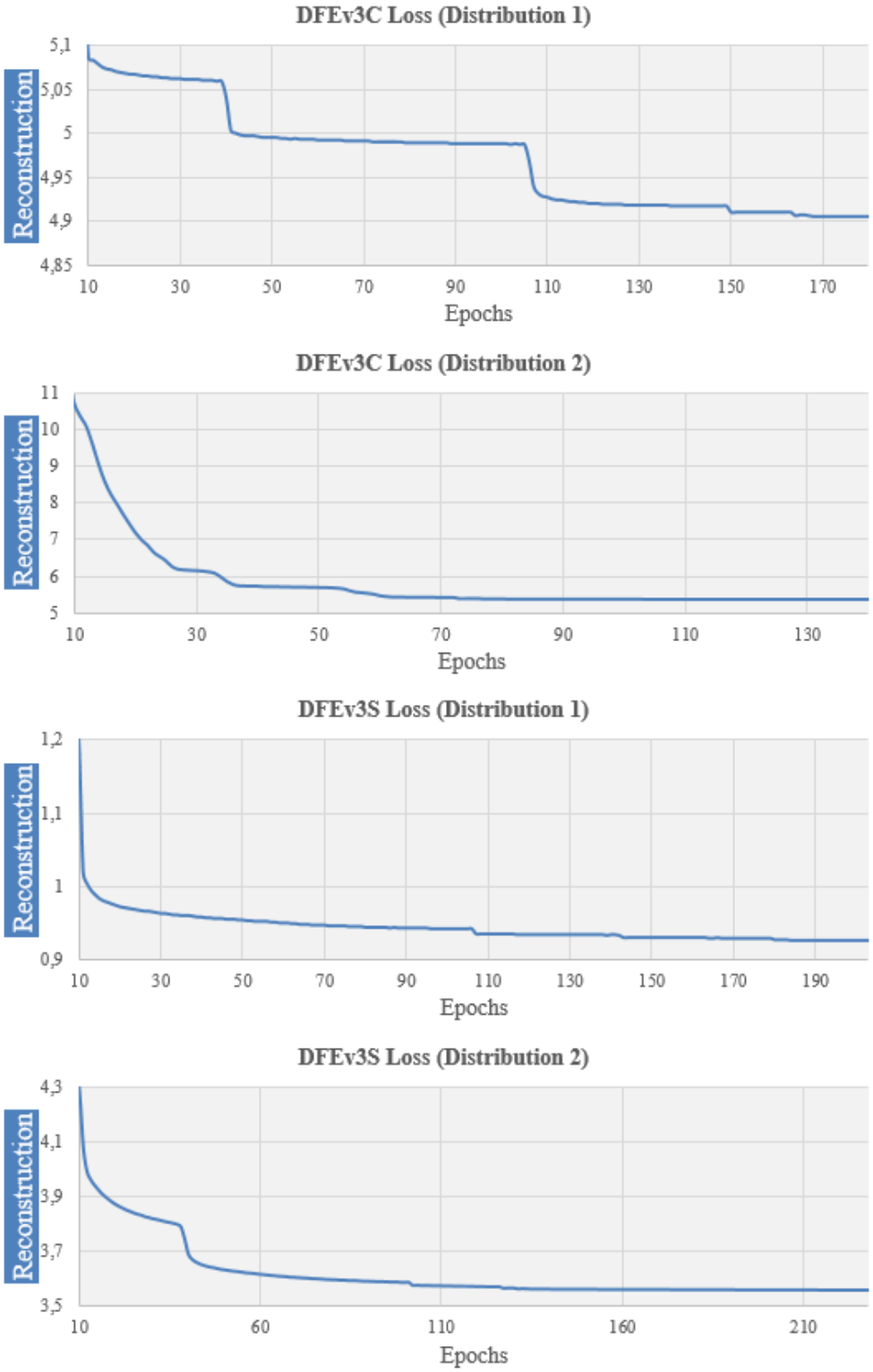

Figure 5.3. Loss change during DFEv3 training

5.3 Results

5.3.1 Classification Performances

We present the performance of the DFEs, by using extracted features with a classifier and showing classification performances. In Table 5.4 and Table 5.5, a comparison of classification results is shown for distribution 1 and distribution 2 (in both distributions data augmentation is included), respectively. In these tables we give CNN network described in 4.1 as the first column which is used as a baseline in comparison. For each subject the average performance of CNN network is given in yellow color. Performances better than this average shift towards green, worse than this average shift towards red.

When we examine results in Table 5.4, we see that for 4 subjects DFEv2C has the best results. For 2 subjects DFEv3C, for 1 subject DFEv2S has the best results. Also, DFEv2C has the highest average score of all subjects, and DFEv3C has the second highest score. Other DFE networks remain under the performance of CNN.

In Table 5.5, we see the results of same networks, but using distribution 2 instead. It can be clearly seen that in distribution 2, all networks achieves better performances except DFEv1C. However, DFEv2C still achieves the best performance.

Table 5.4 Classification performance of DFEs on distribution 1

Network		CNN	DFEv1C	DFEv1S	DFEv2C	DFEv2S	DFEv3C	DFEv3S
ID	**1**	74,66	72,28	64,53	76,48	70,56	75,36	73,28
	2	56,93	56,82	55,54	58,95	56,96	56,70	54,79
	3	52,84	52,88	52,44	51,97	55,28	53,30	51,78
	4	96,78	96,31	92,91	96,75	95,31	96,69	95,84
	5	85,47	78,03	73,56	83,36	78,38	85,14	80,28
	6	76,25	73,81	70,00	81,98	77,06	82,06	77,41
	7	73,13	71,94	62,03	75,08	70,28	73,91	69,88
	8	91,19	90,84	85,09	90,97	85,88	92,42	88,81
	9	84,78	82,94	79,81	87,48	82,13	86,55	83,28
Mean		76,89	75,09	70,66	78,11	74,65	78,01	75,04

Table 5.5 Classification performance of DFEs on distribution 2

Network		CNN	DFEv1C	DFEv1S	DFEv2C	DFEv2S	DFEv3C	DFEv3S
ID	**1**	74,66	70,72	72,50	77,13	74,41	76,33	73,03
	2	56,93	56,04	60,79	62,14	54,89	58,04	59,43
	3	52,84	52,22	54,09	57,08	54,72	53,22	52,72
	4	96,78	97,41	94,59	97,11	94,38	96,61	96,25
	5	85,47	79,81	75,28	85,34	81,19	86,66	80,88
	6	76,25	74,63	75,13	85,36	79,06	82,22	82,06
	7	73,13	67,50	73,25	75,72	71,59	73,34	70,28
	8	91,19	90,56	86,47	93,58	87,75	92,02	90,38
	9	84,78	83,03	82,13	87,30	81,66	86,13	85,16
	Mean	76,89	74,66	74,91	80,08	75,52	78,28	76,69

To sum up, it appears from the results that DFEv2C and DFEv3C increases classification performance compared to CNN network, whereas other DFE networks do not show success.

It can be inferred from the results that training DFEs with single electrode dataset does not increase performance. In addition, using clustering loss does not contribute to the performance, as well. On the other hand, it can also be inferred that using convolutional autoencoders in DFEs for obtaining embedding representation of EEG MI STFT images increases performance. Moreover, assisting autoencoder with a classification loss boosts performance more.

5.3.2 Analysis of Extracted Features

In the previous chapter, it is shown that DFEv2C and DFEv3C increases the classification performance compared to the CNN model which classifies input images directly. To get a better understanding of how these feature extractors improve classification performance, we visualize raw images from the set and features extracted by these DFEs. To achieve this visualization, we use t-SNE (t-

Distributed Stochastic Neighbor Embedding) method. The purpose of t-SNE is to represent high-dimensional data in a lower-dimensional space, typically two dimensions, while preserving the local and global structure of the data.

We made t-SNE analysis for each subject separately and we see similar results. Here we choose to present the analysis for subject IDs 4, 5 and 6. Because all the networks have their best performances on subject ID-4, DFEv3C has the best performance for subject ID-5 and DFEv2C has the best performance for subject ID-6 (distribution 2). Figure 5.4, Figure 5.5 and Figure 5.6 show the t-SNE representation of raw images, DFEv2C and DFEv3C extracted features for left/right hand MI images of subject IDs 4, 5 and 6 respectively. Each point on graphs corresponds to an input image for raw data and corresponds to an extracted feature vector in DFE features.

First remarkable inference on these graphs is that “left” and “right” labeled images are more separated for subject ID-4 regardless of it is raw data or extracted feature. This explains, why classification performance of subject ID-4 is the highest among all subjects.

Second significant implication from these figures is that DFEv2C features are more separated compared to raw data and DFEv3C features. Although no worthy separation exists in raw data for subject IDs 5 and 6, DFEv2C features show clear separation. 9 percent performance increase for subject ID-6 while using DFEv2C features compared to CNN can be explained with this separation.

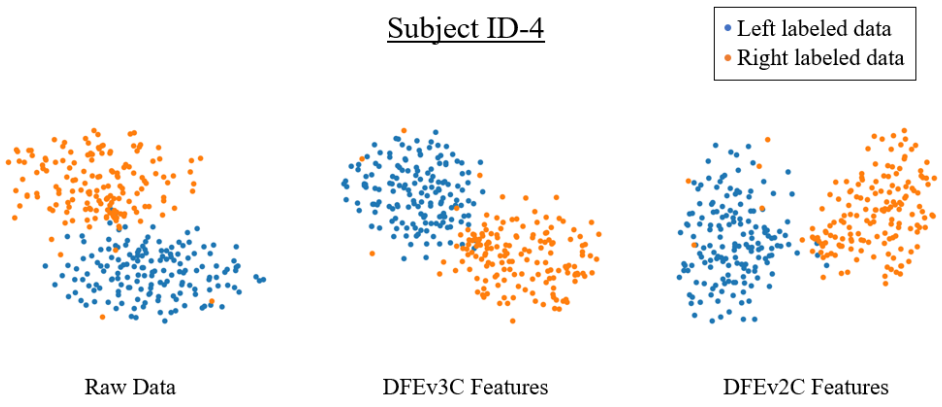

Figure 5.4. t-SNE representation for subject ID-4

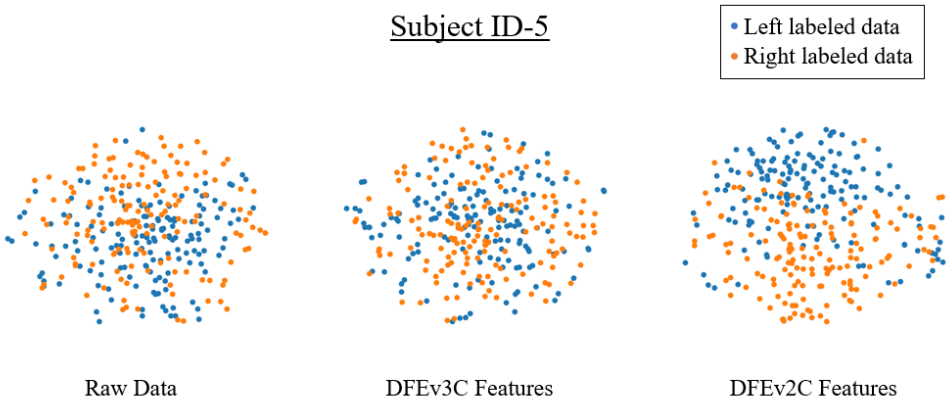

Figure 5.5. t-SNE representation for subject ID-5

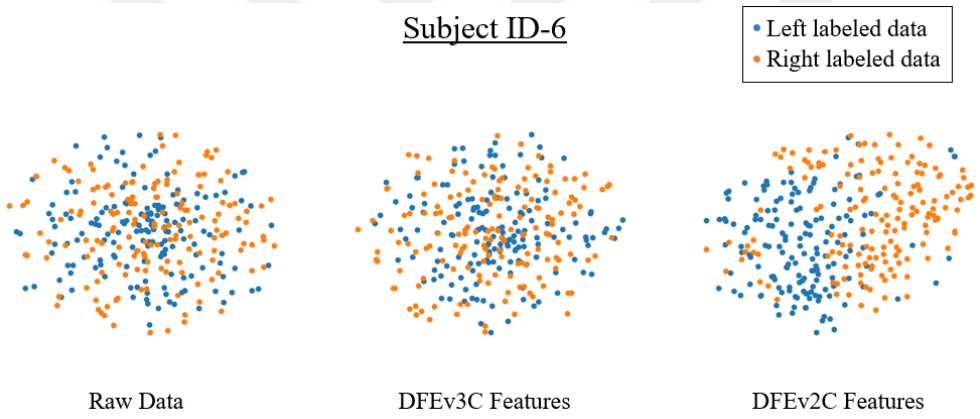

Figure 5.6. t-SNE representation for subject ID-6

Lastly, despite the classification performance boost of DFEv3C compared to CNN, we don't see any apparent improvement on t-SNE representation. DFEv3C uses a CNN feature classifier, unlike DFEv2C which uses a MLP classifier. CNN feature classifier resolves more complex patterns. Therefore, although features extracted by DFEv3C are not separated on t-SNE representation, the CNN classifier can achieve a good performance on classification.

5.3.3 Deeper Analysis of DFEv1 Results

While designing DFEv1 we expect that using clustering loss in an unsupervised way helps creating features that carries information of MI events. However, results show that it does not fulfill the expectation.

To understand the issue, first we checked the reconstructed image quality. In Figure 5.7 and Figure 5.8, two of the reconstructed images are shown with their original forms. As can be seen from figures, reconstructed images are close to original ones. In addition, when we check the loss values at final epochs, reconstruction loss values of DFEv1 are similar to DFEv2. For distribution 1, DFEv1 reconstruction loss value at final epoch is 3% better than DFEv2. So, underperformance of DFEv1 shouldn't cause from the reconstruction part.

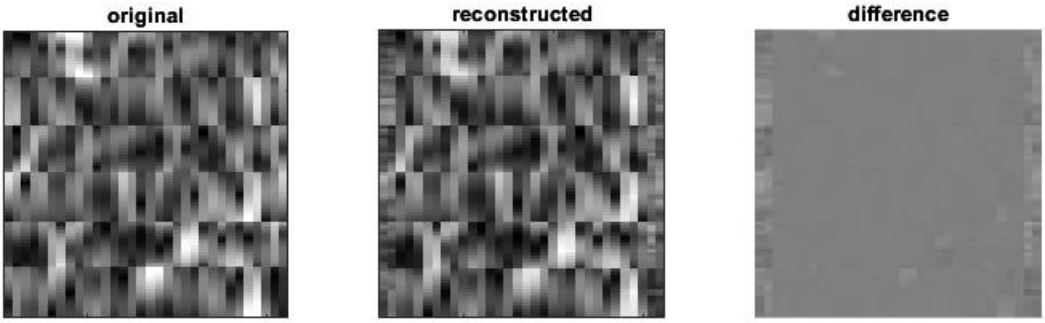

Figure 5.7. Reconstructed image example 1

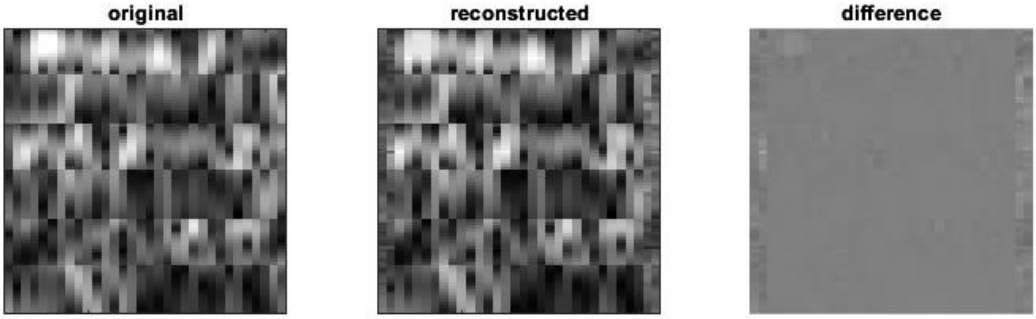

Figure 5.8. Reconstructed image example 2

Another analysis we do in order to comprehend the low success of DFEv1 is to check whether the clusters that are formed in an unsupervised way corresponds to actual labels or not. To achieve this we use t-SNE representations of embedded features and we give statistical information about distribution of labels among clusters.

Figure 5.9 shows t-SNE representation of embedding features obtained with DFEv1C. The left graph shows cluster assignments for these features, whereas the right graph shows actual labels of these data. As it is clearly inferred by comparing these two graphs, obtained clusters do not corresponds to actual labels. Figure 5.10 gives an alternative way to show the distribution of actual labels among clusters of DFEv1C. Ideally we expect that right labeled images gather in one cluster, while left labeled images gather in the other cluster (on the right). However, when we examine the bar charts, we see that both labels are distributed almost equally to both clusters (on the left). This means that unsupervised training learns too little information about the labels of images which finally becomes the cause of low classification success with features obtained with DFEv1 networks.

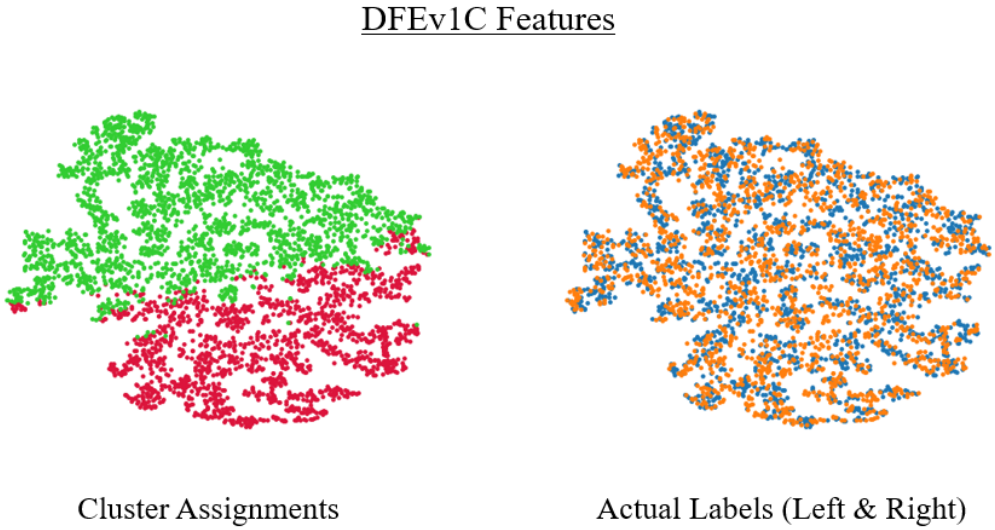

Figure 5.9. t-SNE representation of clusters vs actual labels

Figure 5.10. Distribution of actual labels among clusters and desired distribution

5.3.3.1 Effect of Whitening

In Chapter 3.3, we mentioned the importance of whitening. Because we foresee that images belonging to the same subject ID are more similar to each other compared to images belonging to same MI event. In this subsection, we take a deeper look to the features of DFEv1C network to understand the effect of whitening.

Figure 5.11 and Figure 5.12 shows the distribution of actual labels among clusters of DFEv1C, but this time we give the subject dependent distributions. For each one of the 9 subjects of set-7, we give a separate distribution. In these figures red and green parts of the bars belong to different clusters learned by the DFE in unsupervised manner.

Figure 5.11 shows the distribution when we do not apply whitening to data. This graph clearly shows that the clustering part of the network learns the subjects, but not the labels. As can be seen, most of the images from subjects 1, 7, 8 and 9 get in to cluster 2 (red bars) and most of the images from subjects 2, 3, 5 and 6 get in to cluster 1 (green bars) without depending on the left/right labels. In expected situation, we want all left labeled images to fall into one cluster and all right labeled images to fall into the other cluster without depending on the subject ID.

In order to overcome this foreseen problem, we applied whitening to data so that clustering part of the network will not focus on the subject IDs instead of labels. In Figure 5.12, we show the same distribution when the whitening is applied. Openly, application of whitening removes the dependency to subject IDs. Distribution of data for each subject quite balanced among clusters. However, although the network does not get stuck in learning subject IDs, learning labels is not achieved. Therefore, DFEv1 networks do not fulfill the expected performance.

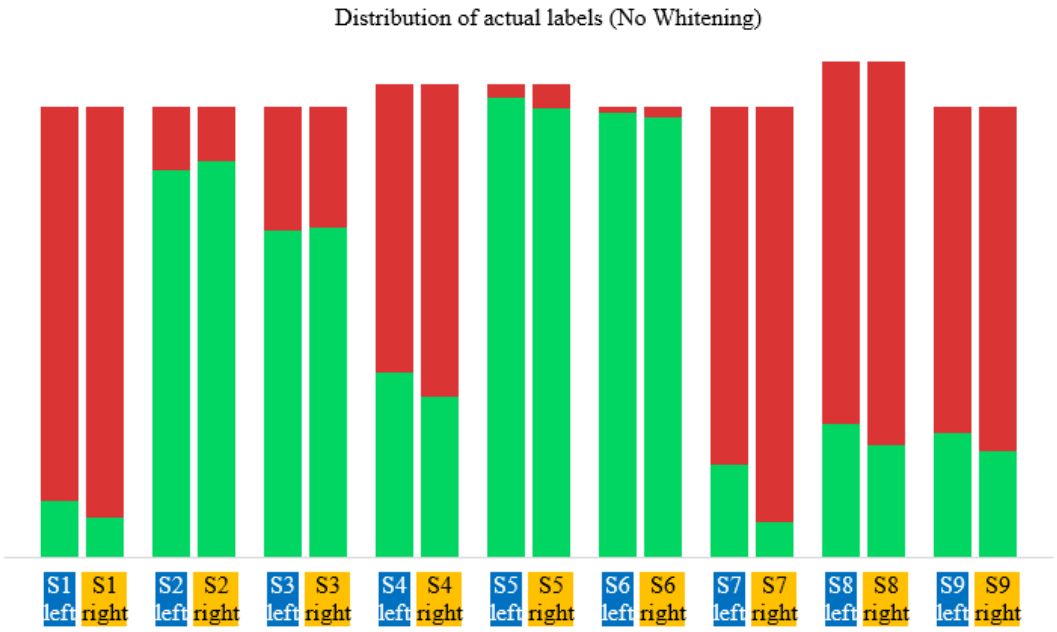

Figure 5.11. Subject dependent distribution of actual labels (No Whitening)

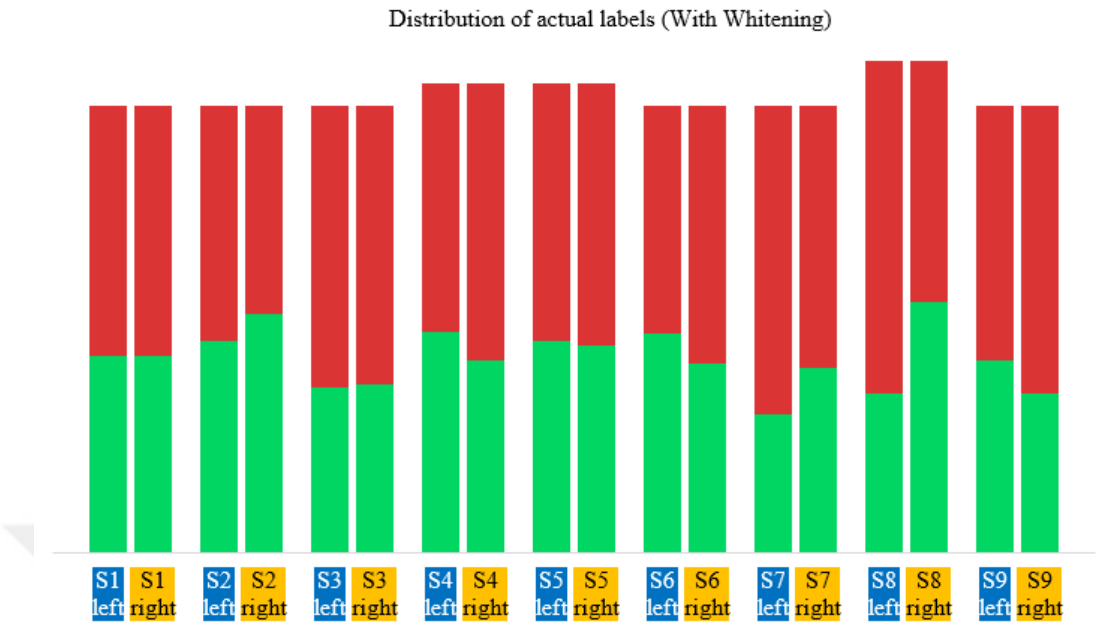

Figure 5.12. Subject dependent distribution of actual labels (With Whitening)

5.3.4 Effect of Augmentation

As given in Table 5.5, the average classification score for DFEv2C distribution 2 is 80.08. This score is obtained while augmentation is applied. Without augmentation average score is 79.32 as given in Table 5.6.

Table 5.6 Classification performance of DFEv2C (dist2) w/wo augmentation

		without Augmentation	with Augmentation
ID	**1**	77,41	77,13
	2	62,57	62,14
	3	57,28	57,08
	4	96,47	97,11
	5	82,38	85,34
	6	83,19	85,36
	7	74,97	75,72
	8	92,38	93,58
	9	87,25	87,30
	Mean	79,32	80,08

We mentioned in 3.4.3 that we applied joint augmentation of gaussian noise and circular shift, and we also mentioned in 5.1.3 that we chose noise variance 0.005 and circular shift count 5. To decide these variance and shift count parameter values, we made several experiments, summary of which is given in Table 5.7. Firstly, we fixed the noise variance to 0.01 and find that optimum circular shift count is 5. Then we fixed the circular shift count to 5 and find that optimum noise variance is 0.005. Using augmentation with these parameters brings us 0.76 increased classification performance for DFEv2C.

Table 5.7 Augmentation experiments

Circular Shift Count		**1**	**5**	**7**	**10**
Noise Variance	**0.000**	79,32	*	*	*
	0.010	*	79,90	79,63	79,73
	0.005	*	80,08	*	*
	0.002	*	78,61	*	*

After tuning augmentation parameters, we also use augmentation on distribution 1 of DFEv2C and two distributions of DFEv3C. Table 5.8 summarizes the effect of augmentation on these networks and distributions.

Table 5.8 Effect of augmentation on different networks/distributions

	without Augmentation	**with Augmentation**	**Performance Increase**
DFEv2C dist1	77,87	78,11	0,24
DFEv2C dist2	79,32	80,08	0,76
DFEv3C dist1	78,21	78,01	-0,20
DFEv3C dist2	78,14	78,28	0,14

5.4 Comparison to State of the Art Methods

In section 2.2.1.3, we indicated the importance of how using different data distributions leads to crucial changes in classification performance. Therefore, in order to compare our results with the popular methods in the literature, we first have to choose researches that use the same data distribution for classification training and

testing. We also referred that [44] notices the mentioned ambiguity about the data distributions of different studies over training/test data and creates a taxonomy of works according to their data distributions while presenting their results. In this chapter, we use their taxonomy to compare our results.

Our both distributions fall into “Data Distribution 2” category in [44], because our distributions use the same data for the training of feature classifier network, they only differ in DFE training. We give DFEv2C-distribution2 as our results, since the best results are obtained at this combination.

Table 5.9 Comparison of DFEv2C with state of the art methods

Network		**EEGNet [86]**	**TLCSD [87]**	**RSMM [88]**	**Bi-Spectrum [89]**	**CSP [90]**	**FBCSP [90]**	**anchored STFT +skip-net-GNAA [44]**	**DFEv2C +MLP Classifier**
ID	**1**	70,00	70,30	72,50	77,00	65,90	70,00	75,00	**77,13**
	2	61,60	50,60	56,40	**64,50**	61,50	60,40	61,60	62,14
	3	**62,00**	52,80	55,60	61,00	56,30	60,90	59,70	57,08
	4	95,60	93,80	97,20	96,50	96,30	**97,50**	96,90	97,11
	5	91,30	63,80	88,40	82,00	76,30	**92,80**	92,20	85,34
	6	77,50	74,10	78,70	84,50	75,00	80,70	**87,20**	85,36
	7	81,40	61,90	77,50	75,00	77,20	77,50	**81,90**	75,72
	8	86,90	83,10	91,90	91,00	92,80	92,50	93,40	**93,58**
	9	78,10	77,20	83,40	87,00	82,80	87,20	**87,80**	87,30
Mean		78,27	69,73	77,96	79,83	76,01	79,94	**81,74**	80,08

Table 5.9 gives comparison of our DFEv2C results with state-of-the art methods that use the same data distribution. As can be inferred from the table, our method outperforms other solutions for subjects ID-1 and ID-8. In addition to that in average our method has a better result than EEGNet [86], TLCSD [87], RSMM [88], Bi-Spectrum [89], CSP [90] and FBCSP [90], and achieves the second best result.

CHAPTER 6

CONCLUSION

In this thesis, we proposed combining all publicly available EEG MI datasets, and trained a deep neural network to be used as a feature extractor for these kind of signals. To achieve this, we prepared two different EEG image datasets, and we designed six different Deep Feature Extractors (DFEs), three of which use the first image set, and others use the second image set.

We adopted input format from [1] while creating combined 3-electrode dataset. We converted time series EEG signals to Short Time Fourier Transform (STFT) images in this set. Also, we concatenated images obtained from three electrode (c3, cz and c4) in one image. Contrarily, in single electrode image dataset, we prefered using images obtained from different electrodes separately dissimilar to [1].

There exist variations in public EEG MI datasets because of the different acquisition setups, different equipments, different softwares and different acquisition scenarios. To overcome this problem, we proposed four step signal processing before applying STFT to signals. First, we found useful to apply 3Hz-100Hz bandpass filter to signals, which removes unwanted noisy frequencies and still maintains the necessary information. Secondly, we offered resampling to equalize the sampling frequency of all signals to 250Hz. Then, we suggested applying amplitude normalization. Finally, we preferred extracting 2-second signals starting from 0.5 seconds after first cue is displayed. With the help of these four steps we achieved obtaining similar time series signals from all datasets before applying STFT.

After converting time series signals to time frequency images, we noticed that despite the signal processing in the raw time series signal level, STFT images still have more similarity depending on the dataset and test subjects, and have less similarity coming from motor imagery movement types. We overcomed this issue

by applying a special type of whitening. Instead of applying whitening to whole image at once, we prefered row based whitening not to disrupt the frequency information.

In order to increase data count, we applied a joint augmentation technique which involves adding gaussian noise and shifting image circularly along the time axis. Since preserving frequency and electrode information is crucial in our dataset, here we chose applying circular shift only on time axis and not in frequency and electrode axes.

In chapter 4, we gave details of 6 different DFE networks we designed and classifier networks that use features created by these DFEs. In addition to DFEs, we gave details of a CNN network to be used for performance comparison during experiments.

There are 6 different DFEs that we designed because we tried three different methods on our two different datasets. First method was inspired from [14] and it involves using a convolutional autoencoder (CAE) to obtain deep representations of input images, and applying clustering on these embedded representations. This method was completely unsupervised, whereas in the second method we tried a supervised solution. This solution contains the same CAE, but instead of using clustering techniques on embedded features, we tried classifying embedded features in a supervised manner. In the last method, we only preferred to use CAE to obtain embedded features. We named our DFEs based on the method they used as DFEv1, DFEv2 and DFEv3 respectively, and we attached letter “S” or “C” to the end of the name depending on the image set it uses.

To measure the quality of the extracted DFE features, we performed several experiments. Firstly, we used extracted features in classification tasks and compared the accuracy with the CNN network that tries classification directly from images. According to experimental results, among 6 DFEs, DFEv2C and DFEv3C outperformed CNN network, whereas other networks did not show sufficient performance. Between the successful two networks, DFEv2C was slightly better

than the other. Our results showed that classifying features extracted by DFEv2C produces 3,19% better performance compared to CNN, and the performance boost was 1,39% for DFEv3C.

Besides classification, we also visually verified that extracted features of these networks are more seperable compared to images. To make this vizualization we used t-SNE representation. We also got a deeper understanding of why DFEv1 networks did not perform well using these representations.

In our experiments, we corrected that using whitening prevented unsupervised network DFEv1C to learn subject-IDs which was an undesired situation. However, still whitening was not sufficient to make the network to learn motor imagery movements. We also measured the effect of augmentation quantitatively. We saw that applying joint augmentation of gaussian noise and circular shift increased the classification performance by 0,76%

Finally, we compared our DFEv2C network with state of the art methods. Our concern in this part was dissolving the ambiguity of the distribution of the data in other studies. When we compared our results with appropriate studies, we see that it shows better performance than all of them except [44] in which anchored-STFT is used for feature extraction, gradient norm adversarial augmentation is used for data augmentation and inputs are classified using Skip-Net algorithm. Anchored-STFT method used in this work is different from our STFT application and it outputs a detailed 3D STFT data. In addition augmentation method used is different from ours. In their method, data augmentation is not directly applied to STFT data. They train the network with the existing dataset. Then using the gradient of the cost function on this trained network, they add some small disturbance to obtain the augmented data. Lastly, they use a CNN-based classifier, which has two convolutional layers and a skip connection that combines outputs of the first and the second convolutional layers. Considering these differences their method has better classification accuracy.

To sum up, our method can be used to extract valuable features from EEG MI signals and compared to CNN network, the proposed approach results in performance

improvement. As a future work, number of electrodes used can be increased, since we only considered three of them. In addition to that, different type of input formats which contains more details can be tried to achieve better results. Moreover, we applied augmentation only on image level. Applying augmentation on raw time series signal level may boost the performance more. Lastly, we selected 2-second signal segments, choosing a larger time segment can produce better results.

REFERENCES

[1] Tabar, Y. R., & Halici, U. (2017). A novel deep learning approach for classification of EEG motor imagery signals. *Journal of Neural Engineering*, *14*(1), 016003. https://doi.org/10.1088/1741-2560/14/1/016003. 1.3 2.2.1 2.2.3 2.2.4 3.1.4 3.2 3.2.1 4.1 4.2.1 6

[2] Rak, R. J., Kołodziej, M., & Majkowski, A. (2012). Brain-computer interface as measurement and control system the review paper. *Metrology and Measurement Systems*, *19*(3), 427–444. https://doi.org/10.2478/v10178-012-0037-4. 1.1 2.2.2 3.2.1

[3] Abdulkader, S. N., Atia, A., & Mostafa, M.-S. M. (2015). Brain computer interfacing: Applications and challenges. *Egyptian Informatics Journal*, *16*(2), 213–230. https://doi.org/10.1016/j.eij.2015.06.002. 1.1

[4] Rasheed, S. (2021). A Review of the Role of Machine Learning Techniques towards Brain–Computer Interface Applications. In *Machine Learning and Knowledge Extraction* (Vol. 3, Issue 4, pp. 835–862). MDPI. https://doi.org/10.3390/make3040042. 1.1

[5] Caron, M., Bojanowski, P., Joulin, A., & Douze, M. (2018). *Deep Clustering for Unsupervised Learning of Visual Features*. http://arxiv.org/abs/1807.05520. 2.3 2.3.1 2.3.3

[6] Hershey, J. R., Chen, Z., Roux, J. le, & Watanabe, S. (2015). *Deep clustering: Discriminative embeddings for segmentation and separation*. http://arxiv.org/abs/1508.04306. 2.3.3

[7] McConville, R., Santos-Rodriguez, R., Piechocki, R. J., & Craddock, I. (2019). *N2D: (Not Too) Deep Clustering via Clustering the Local Manifold of an Autoencoded Embedding*. http://arxiv.org/abs/1908.05968. 2.3.1 2.3.3

[8] Nutakki, G. C., Abdollahi, B., Sun, W., & Nasraoui, O. (2019). *An Introduction to Deep Clustering* (pp. 73–89). https://doi.org/10.1007/978-3-319-97864-2_4. 2.3

[9] Faragó, K. B., Skaf, J., Forgács, S., Hevesi, B., & Lőrincz, A. (2022). Soldering Data Classification with a Deep Clustering Approach: Case Study of an Academic-Industrial Cooperation. *Applied Sciences (Switzerland)*, *12*(14). https://doi.org/10.3390/app12146927. 2.3.1 2.3.3

[10] Karim, M. R., Beyan, O., Zappa, A., Costa, I. G., Rebholz-Schuhmann, D., Cochez, M., & Decker, S. (2021). Deep learning-based clustering approaches for bioinformatics. *Briefings in Bioinformatics*, *22*(1), 393–415. https://doi.org/10.1093/bib/bbz170. 2.3

[11] Aljalbout, E., Golkov, V., Siddiqui, Y., Strobel, M., & Cremers, D. (2018). *Clustering with Deep Learning: Taxonomy and New Methods*. http://arxiv.org/abs/1801.07648. 2.3

[12] Li, L., & Kameoka, H. (2018). Deep Clustering with Gated Convolutional Networks. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, *2018-April*, 16–20. https://doi.org/10.1109/ICASSP.2018.8461746. 2.3.3

[13] Guérin, J., & Boots, B. (2018). *Improving Image Clustering With Multiple Pretrained CNN Feature Extractors*. http://arxiv.org/abs/1807.07760. 2.3.1 2.3.3

[14] Guo, X., Liu, X., Zhu, E., & Yin, J. (2017). Deep Clustering with Convolutional Autoencoders. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 10635 LNCS* (pp. 373–382). Springer Verlag. https://doi.org/10.1007/978-3-319-70096-0_39. 2.3 2.3.1 2.3.3 4.2 6

[15] Mrabah, N., Khan, N. M., Ksantini, R., & Lachiri, Z. (2020). Deep clustering with a Dynamic Autoencoder: From reconstruction towards centroids construction. *Neural Networks : The Official Journal of the International Neural Network Society*, *130*, 206–228. https://doi.org/10.1016/j.neunet.2020.07.005. 2.3 2.3.1 2.3.3

[16] Lu, S., & Li, R. (2021). *DAC: Deep Autoencoder-based Clustering, a General Deep Learning Framework of Representation Learning*. http://arxiv.org/abs/2102.07472. 2.3.1 2.3.3

[17] Chang, S. (2022). *Deep clustering with fusion autoencoder*. http://arxiv.org/abs/2201.04727. 2.3.1 2.3.3

[18] Chazan, S. E., Gannot, S., & Goldberger, J. (2019). Deep Clustering Based On A Mixture Of Autoencoders. *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, 1–6. https://doi.org/10.1109/MLSP.2019.8918720. 2.3.1 2.3.3

[19] Dizaji, K. G., Herandi, A., Deng, C., Cai, W., & Huang, H. (2017). *Deep Clustering via Joint Convolutional Autoencoder Embedding and Relative Entropy Minimization*. http://arxiv.org/abs/1704.06327. 2.3 2.3.1 2.3.3

[20] Tavakoli, N., Siami-Namini, S., Adl Khanghah, M., Mirza Soltani, F., & Siami Namin, A. (2020). An autoencoder-based deep learning approach for clustering time series data. *SN Applied Sciences*, *2*(5). https://doi.org/10.1007/s42452-020-2584-8. 2.3.1

[21] Dong, S., Xu, H., Zhu, X., Guo, X., Liu, X., & Wang, X. (2020). Multi-View Deep Clustering based on AutoEncoder. *Journal of Physics: Conference Series*, *1684*(1), 012059. https://doi.org/10.1088/1742-6596/1684/1/012059. 2.3.1 2.3.3

[22] Opochinsky, Y., Chazan, S. E., Gannot, S., & Goldberger, J. (2020). K-Autoencoders Deep Clustering. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4037–4041. https://doi.org/10.1109/ICASSP40776.2020.9053109. 2.3.1

[23] Hadipour, H., Liu, C., Davis, R., Cardona, S. T., & Hu, P. (2022). Deep clustering of small molecules at large-scale via variational autoencoder embedding and K-means. *BMC Bioinformatics*, *23*. https://doi.org/10.1186/s12859-022-04667-1. 2.3.1

[24] Toma, R. N., Piltan, F., & Kim, J.-M. (2021). A Deep Autoencoder-Based Convolution Neural Network Framework for Bearing Fault Classification in Induction Motors. *Sensors (Basel, Switzerland)*, *21*(24). https://doi.org/10.3390/s21248453. 2.3.1 2.3.3

[25] Lim, K. L., Jiang, X., & Yi, C. (2020). Deep Clustering with Variational Autoencoder. *IEEE Signal Processing Letters*, *27*, 231–235. https://doi.org/10.1109/LSP.2020.2965328. 2.3.1

[26] Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., & Joulin, A. (2020). *Unsupervised Learning of Visual Features by Contrasting Cluster Assignments*. http://arxiv.org/abs/2006.09882. 2.3.1

[27] Asano, Y. M., Rupprecht, C., & Vedaldi, A. (2019). *Self-labelling via simultaneous clustering and representation learning*. http://arxiv.org/abs/1911.05371. 2.3.3

[28] Tabar, Y. R. (2017). *MOTOR IMAGERY EEG SIGNAL CLASSIFICATION USING DEEP LEARNING FOR BRAIN COMPUTER INTERFACES A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF MIDDLE EAST TECHNICAL UNIVERSITY*. https://catalog.library.metu.edu.tr/search~S15?/XYousef+Rezaei+Tabar&searchscope=15&SORT=D/XYousef+Rezaei+Tabar&searchscope=15&SORT=D&SUBKEY=Yousef+Rezaei+Tabar/1%2C11%2C11%2CB/frameset&FF=XYousef+Rezaei+Tabar&searchscope=15&SORT=D&1%2C1%2C. 3.2

[29] Altaheri, H., Muhammad, G., Alsulaiman, M., Amin, S. U., Altuwaijri, G. A., Abdul, W., Bencherif, M. A., & Faisal, M. (2021). Deep learning techniques for classification of electroencephalogram (EEG) motor imagery (MI) signals: a review. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-021-06352-5. 1.1

[30] Al-Saegh, A., Dawwd, S. A., & Abdul-Jabbar, J. M. (2021). Deep learning for motor imagery EEG-based classification: A review. *Biomedical Signal Processing and Control*, *63*, 102172. https://doi.org/10.1016/j.bspc.2020.102172. 1.1 1.3

[31] Zhang, C., Kim, Y. K., & Eskandarian, A. (2021). EEG-inception: An accurate and robust end-to-end neural network for EEG-based motor imagery classification. *Journal of Neural Engineering*, *18*(4). https://doi.org/10.1088/1741-2552/abed81. 2.2.1 2.2.2 2.2.3

[32] Zhao, H., Zheng, Q., Ma, K., Li, H., & Zheng, Y. (2021). Deep Representation-Based Domain Adaptation for Nonstationary EEG Classification. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(2), 535–545. https://doi.org/10.1109/TNNLS.2020.3010780. 2.2.1 2.2.3

[33] Roy, S., Chowdhury, A., McCreadie, K., & Prasad, G. (2020). Deep Learning Based Inter-subject Continuous Decoding of Motor Imagery for Practical Brain-Computer Interfaces. *Frontiers in Neuroscience*, *14*. https://doi.org/10.3389/fnins.2020.00918. 2.2.1 2.2.3

[34] Zhang, K., Xu, G., Chen, L., Tian, P., Han, C., Zhang, S., & Duan, N. (2020). Instance Transfer Subject-Dependent Strategy for Motor Imagery Signal Classification Using Deep Convolutional Neural Networks. *Computational and Mathematical Methods in Medicine*, *2020*, 1683013. https://doi.org/10.1155/2020/1683013. 2.2.1 2.2.3

[35] Zhang, K., Xu, G., Han, Z., Ma, K., Zheng, X., Chen, L., Duan, N., & Zhang, S. (2020). Data Augmentation for Motor Imagery Signal Classification Based on a Hybrid Neural Network. *Sensors (Basel, Switzerland)*, *20*(16), 1–20. https://doi.org/10.3390/s20164485. 2.2.1 2.2.3 2.2.4

[36] Tayeb, Z., Fedjaev, J., Ghaboosi, N., Richter, C., Everding, L., Qu, X., Wu, Y., Cheng, G., & Conradt, J. (2019). Validating deep neural networks for online decoding of motor imagery movements from eeg signals. *Sensors (Switzerland)*, *19*(1). https://doi.org/10.3390/s19010210. 2.2.1 2.2.3

[37] Rong, Y., Wu, X., & Zhang, Y. (2020). Classification of motor imagery electroencephalography signals using continuous small convolutional neural network. *International Journal of Imaging Systems and Technology*, *30*(3), 653–659. https://doi.org/10.1002/ima.22405. 2.2.1 2.2.3

[38] Dai, M., Zheng, D., Na, R., Wang, S., & Zhang, S. (2019). EEG Classification of Motor Imagery Using a Novel Deep Learning Framework. *Sensors (Basel, Switzerland)*, *19*(3), 551. https://doi.org/10.3390/s19030551. 2.2.1 2.2.3 2.2.4

[39] Xu, G., Shen, X., Chen, S., Zong, Y., Zhang, C., Yue, H., Liu, M., Chen, F., & Che, W. (2019). A Deep Transfer Convolutional Neural Network Framework for EEG Signal Classification. *IEEE Access*, *7*, 112767–112776. https://doi.org/10.1109/ACCESS.2019.2930958. 2.2.1 2.2.3

[40] Ha, K.-W., & Jeong, J.-W. (2019). Motor Imagery EEG Classification Using Capsule Networks. *Sensors (Basel, Switzerland)*, *19*(13), 2854. https://doi.org/10.3390/s19132854. 2.2.1 2.2.3

[41] Milanes Hermosilla, D., Trujillo Codorniu, R., Lopez Baracaldo, R., Sagaro Zamora, R., Delisle Rodriguez, D., Llosas Albuerne, Y., & Alvarez, J. R. N. (2021). Shallow Convolutional Network Excel for Classifying Motor Imagery EEG in BCI Applications. *IEEE Access*, *9*, 98275–98286. https://doi.org/10.1109/ACCESS.2021.3091399. 2.2.1 2.2.3

[42] Milanés-Hermosilla, D., Trujillo-Codorniú, R., Lamar-Carbonell, S., Sagaró-Zamora, R., Tamayo-Pacheco, J. J., Villarejo-Mayor, J. J., & Delisle-Rodriguez, D. (2023). Robust Motor Imagery Tasks Classification Approach Using Bayesian Neural Network. *Sensors (Basel, Switzerland)*, *23*(2). https://doi.org/10.3390/s23020703. 2.2.1 2.2.3

[43] Ma, W., Wang, C., Sun, X., Lin, X., & Wang, Y. (2023). A double-branch graph convolutional network based on individual differences weakening for motor imagery EEG classification. *Biomedical Signal Processing and Control*, *84*, 104684. https://doi.org/10.1016/j.bspc.2023.104684. 2.2.1 2.2.3

[44] Ali, O., Saif-Ur-Rehman, M., Dyck, S., Glasmachers, T., Iossifidis, I., & Klaes, C. (2022). Enhancing the decoding accuracy of EEG signals by the introduction of anchored-STFT and adversarial data augmentation method. *Scientific Reports*, *12*(1), 4245. https://doi.org/10.1038/s41598-022-07992-w. 2.2.1 2.2.1.3 2.2.3 5.4 6

[45] Cho, H., Ahn, M., Ahn, S., Kwon, M., & Jun, S. C. (2017). EEG datasets for motor imagery brain–computer interface. *GigaScience*, *6*(7). https://doi.org/10.1093/gigascience/gix034. 1.1 2.1 2.1.3

[46] Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P. C., Mark, R., & Stanley, H. E. (2009, September 9). *EEG Motor Movement/Imagery Dataset*. EEG Motor Movement/Imagery Dataset. https://doi.org/10.13026/C28G6P. 1.1 2.1

[47] Luciw, M. D., Jarocka, E., & Edin, B. B. (2014). Multi-channel EEG recordings during 3,936 grasp and lift trials with varying weight and friction. *Scientific Data*, *1*(1), 140047. https://doi.org/10.1038/sdata.2014.47. 1.1 2.1

[48] Kaya, M., Binli, M. K., Ozbay, E., Yanar, H., & Mishchenko, Y. (2018). A large electroencephalographic motor imagery dataset for electroencephalographic brain computer interfaces. *Scientific Data*, *5*(1), 180211. https://doi.org/10.1038/sdata.2018.211. 1.1 2.1

[49] Blankertz, B. (2008). *BCI Competition IV - Dataset 1*. https://www.bbci.de/competition/iv/desc_1.html. 1.1 2.1

[50] Brunner, C., Leeb, R., Müller-Putz, G. R., Schlögl, A., & Pfurtscheller, G. (2008). *BCI Competition 2008-Graz data set A Experimental paradigm*. https://www.bbci.de/competition/iv/#dataset2a. 1.1 1.3 2.1 2.2.1

[51] Leeb, R., Brunner, C., Müller-Putz, G. R., Schlögl, A., & Pfurtscheller, G. (2008). *BCI Competition 2008-Graz data set B Experimental paradigm*. https://www.bbci.de/competition/iv/#dataset2b. 1.1 1.3 2.1 2.2.1

[52] Schirrmeister, R. T., Springenberg, J. T., Fiederer, L. D. J., Glasstetter, M., Eggensperger, K., Tangermann, M., Hutter, F., Burgard, W., & Ball, T. (2017). Deep learning with convolutional neural networks for EEG decoding and visualization. *Human Brain Mapping*, *38*(11), 5391–5420. https://doi.org/10.1002/hbm.23730. 1.1 2.1

[53] *Project BCI - EEG motor activity data set*. (n.d.). Retrieved April 30, 2023, from https://sites.google.com/site/projectbci/. 1.1 2.1

[54] Bhatt, R. (2012, July 17). *Planning Relax Data Set*. https://archive.ics.uci.edu/ml/datasets/Planning+Relax#. 1.1 2.1

[55] Zhou, Q. (2020, November 29). *EEG dataset of 7-day Motor Imagery BCI* . IEEE Dataport. https://doi.org/10.21227/f1c7-7x89. 1.1 2.1

[56] Steyrl, D., Scherer, R., Förstner, O., & Müller-Putz, G. R. (2014). Motor Imagery Brain-Computer Interfaces: Random Forests vs Regularized LDA-Non-linear Beats Linear. *6th International Brain-Computer Interface Conference*. https://doi.org/10.3217/978-3-85125-378-8-61. 1.1 2.1 2.1.3

[57] Mahato, S., Goyal, N., Ram, D., & Paul, S. (2020). Detection of Depression and Scaling of Severity Using Six Channel EEG Data. *Journal of Medical Systems*, *44*(7), 118. https://doi.org/10.1007/s10916-020-01573-y. 2.1.4

[58] Wu, H., Niu, Y., Li, F., Li, Y., Fu, B., Shi, G., & Dong, M. (2019). A Parallel Multiscale Filter Bank Convolutional Neural Networks for Motor Imagery EEG Classification. *Frontiers in Neuroscience*, *13*. https://doi.org/10.3389/fnins.2019.01275. 2.2.1 2.2.3

[59] Lee, H. K., & Choi, Y.-S. (2019). Application of Continuous Wavelet Transform and Convolutional Neural Network in Decoding Motor Imagery Brain-Computer Interface. *Entropy*, *21*(12), 1199. https://doi.org/10.3390/e21121199. 2.2.1 2.2.3

[60] Cheng, L., Li, D., Yu, G., Zhang, Z., Li, X., & Yu, S. (2020). A Motor Imagery EEG Feature Extraction Method Based on Energy Principal Component Analysis and Deep Belief Networks. *IEEE Access*, *8*, 21453–21472. https://doi.org/10.1109/ACCESS.2020.2969054. 2.2.1 2.2.3 2.2.4

[61] Li, M.-A., Han, J.-F., & Duan, L.-J. (2020). A Novel MI-EEG Imaging With the Location Information of Electrodes. *IEEE Access*, *8*, 3197–3211. https://doi.org/10.1109/ACCESS.2019.2962740. 2.2.1

[62] Xue, J., Ren, F., Sun, X., Yin, M., Wu, J., Ma, C., & Gao, Z. (2020). A Multifrequency Brain Network-Based Deep Learning Framework for Motor Imagery Decoding. *Neural Plasticity*, *2020*, 1–11. https://doi.org/10.1155/2020/8863223. 2.2.1 2.2.3

[63] KORHAN, N., ABİLZADE, L., ÖLMEZ, T., & ÖLMEZ, Z. D. (2021). Classification of left and right hand motor imagery EEG signals by using deep neural networks. *International Journal of Applied Mathematics Electronics and Computers*, *9*(4), 85–90. https://doi.org/10.18100/ijamec.995022. 2.2.1 2.2.1.3 2.2.3

[64] Liu, T., & Yang, D. (2021). A Densely Connected Multi-Branch 3D Convolutional Neural Network for Motor Imagery EEG Decoding. *Brain Sciences*, *11*(2), 197. https://doi.org/10.3390/brainsci11020197. 2.2.1 2.2.2

[65] Deng, X., Zhang, B., Yu, N., Liu, K., & Sun, K. (2021). Advanced TSGL-EEGNet for Motor Imagery EEG-Based Brain-Computer Interfaces. *IEEE Access*, *9*, 25118–25130. https://doi.org/10.1109/ACCESS.2021.3056088. 2.2.1 2.2.3

[66] Xu, M., Yao, J., Zhang, Z., Li, R., Yang, B., Li, C., Li, J., & Zhang, J. (2020). Learning EEG topographical representation for classification via convolutional neural network. *Pattern Recognition*, *105*, 107390. https://doi.org/10.1016/j.patcog.2020.107390. 2.2.1

[67] Liao, J. J., Luo, J. J., Yang, T., So, R. Q. Y., & Chua, M. C. H. (2020). Effects of local and global spatial patterns in EEG motor-imagery classification using convolutional neural network. *Brain-Computer Interfaces*, *7*(3–4), 47–56. https://doi.org/10.1080/2326263X.2020.1801112. 2.2.1

[68] Amin, S. U., Alsulaiman, M., Muhammad, G., Bencherif, M. A., & Hossain, M. S. (2019). Multilevel Weighted Feature Fusion Using Convolutional Neural Networks for EEG Motor Imagery Classification. *IEEE Access*, *7*, 18940–18950. https://doi.org/10.1109/ACCESS.2019.2895688. 2.2.1 2.2.2 2.2.3

[69] Kumar, S., Sharma, R., & Sharma, A. (2021). OPTICAL+: a frequency-based deep learning scheme for recognizing brain wave signals. *PeerJ Computer Science*, *7*, e375. https://doi.org/10.7717/peerj-cs.375. 2.2.1 2.2.3 2.2.4

[70] Zhao, X., Zhao, J., Liu, C., & Cai, W. (2020). Deep Neural Network with Joint Distribution Matching for Cross-Subject Motor Imagery Brain-Computer Interfaces. *BioMed Research International*, *2020*, 1–15. https://doi.org/10.1155/2020/7285057. 2.2.1 2.2.3

[71] Fan, C.-C., Yang, H., Hou, Z.-G., Ni, Z.-L., Chen, S., & Fang, Z. (2021). Bilinear neural network with 3-D attention for brain decoding of motor imagery movements from the human EEG. *Cognitive Neurodynamics*, *15*(1), 181–189. https://doi.org/10.1007/s11571-020-09649-8. 2.2.1 2.2.3

[72] Lun, X., Yu, Z., Chen, T., Wang, F., & Hou, Y. (2020). A Simplified CNN Classification Method for MI-EEG via the Electrode Pairs Signals. *Frontiers in Human Neuroscience*, *14*. https://doi.org/10.3389/fnhum.2020.00338. 2.2.1 2.2.3

[73] Roots, K., Muhammad, Y., & Muhammad, N. (2020). Fusion Convolutional Neural Network for Cross-Subject EEG Motor Imagery Classification. *Computers*, *9*(3), 72. https://doi.org/10.3390/computers9030072. 2.2.1 2.2.3

[74] Hou, Y., Zhou, L., Jia, S., & Lun, X. (2020). A novel approach of decoding EEG four-class motor imagery tasks via scout ESI and CNN. *Journal of Neural Engineering*, *17*(1), 016048. https://doi.org/10.1088/1741-2552/ab4af6. 2.2.1 2.2.3

[75] Li, D., Wang, J., Xu, J., & Fang, X. (2019). Densely Feature Fusion Based on Convolutional Neural Networks for Motor Imagery EEG Classification. *IEEE Access*, *7*, 132720–132730. https://doi.org/10.1109/ACCESS.2019.2941867. 2.2.1 2.2.3

[76] Peng, X., Feng, J., Xiao, S., Lu, J., Yi, Z., & Yan, S. (2017). *Deep Sparse Subspace Clustering*. http://arxiv.org/abs/1709.08374. 2.3.3

[77] Tian, F., Gao, B., Cui, Q., Chen, E., & Research, M. (2014). *Learning Deep Representations for Graph Clustering Tie-Yan Liu*. www.aaai.org. 2.3

[78] Zhou, S., Xu, H., Zheng, Z., Chen, J., li, Z., Bu, J., Wu, J., Wang, X., Zhu, W., & Ester, M. (2022). *A Comprehensive Survey on Deep Clustering: Taxonomy, Challenges, and Future Directions*. http://arxiv.org/abs/2206.07579. 2.3

[79] Ji, P., Zhang, T., Li, H., Salzmann, M., & Reid, I. (2017). *Deep Subspace Clustering Networks*. http://arxiv.org/abs/1709.02508. 2.3

[80] Li, Y., Hu, P., Liu, Z., Peng, D., Zhou, J. T., & Peng, X. (2021). Contrastive Clustering. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(10), 8547–8555. https://doi.org/10.1609/aaai.v35i10.17037. 2.3 2.3.1

[81] Dang, Z., Deng, C., Yang, X., & Huang, H. (2021). *Doubly Contrastive Deep Clustering*. http://arxiv.org/abs/2103.05484. 2.3.1

[82] Yang, J., Parikh, D., & Batra, D. (2016). *Joint Unsupervised Learning of Deep Representations and Image Clusters*. https://doi.org/10.48550. 2.3

[83] Mukherjee, S., Asnani, H., Lin, E., & Kannan, S. (2018). *ClusterGAN : Latent Space Clustering in Generative Adversarial Networks*. http://arxiv.org/abs/1809.03627. 2.3.1

[84] Yang, B., Fu, X., Sidiropoulos, N. D., & Hong, M. (2016). *Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering*. http://arxiv.org/abs/1610.04794. 2.3.2

[85] Huang, P., Huang, Y., Wang, W., & Wang, L. (2014). Deep Embedding Network for Clustering. *2014 22nd International Conference on Pattern Recognition*, 1532–1537. https://doi.org/10.1109/ICPR.2014.272. 2.3.2

[86] Lawhern, V. J., Solon, A. J., Waytowich, N. R., Gordon, S. M., Hung, C. P., & Lance, B. J. (2018). EEGNet: a compact convolutional neural network for EEG-based brain–computer interfaces. *Journal of Neural Engineering*, *15*(5), 056013. https://doi.org/10.1088/1741-2552/aace8c 5.4

[87] Raza, H., Cecotti, H., Li, Y., & Prasad, G. (2016). Adaptive learning with covariate shift-detection for motor imagery-based brain–computer interface. *Soft Computing*, *20*(8), 3085–3096. https://doi.org/10.1007/s00500-015-1937-5. 5.4

[88] Zheng, Q., Zhu, F., & Heng, P.-A. (2018). Robust Support Matrix Machine for Single Trial EEG Classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *26*(3), 551–562. https://doi.org/10.1109/TNSRE.2018.2794534. 5.4

[89] Shahid, S., & Prasad, G. (2011). Bispectrum-based feature extraction technique for devising a practical brain–computer interface. *Journal of Neural Engineering*, *8*(2), 025014. https://doi.org/10.1088/1741-2560/8/2/025014. 5.4

[90] Ang, K. K., Chin, Z. Y., Wang, C., Guan, C., & Zhang, H. (2012). Filter Bank Common Spatial Pattern Algorithm on BCI Competition IV Datasets 2a and 2b. *Frontiers in Neuroscience*, *6*(MAR). https://doi.org/10.3389/fnins.2012.00039. 5.4