Styrk Hundseid Kamsvåg
Oskar Størmer

Exploring EEG self-supervised learning through channel grouping

Master's thesis in MTDT
Supervisor: Donn Morrison
Co-supervisor: Odd Sandbekkhaug
June 2023

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Styrk Hundseid Kamsvåg
Oskar Størmer

Exploring EEG self-supervised learning through channel grouping

Master's thesis in MTDT
Supervisor: Donn Morrison
Co-supervisor: Odd Sandbekkhaug
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

EEG classification holds a critical role in neuroscience. The identified lack of transferability and limited utilization of transformer-based models inspired this study to explore novel approaches that can enhance EEG classification. This master's thesis explores the application of self-supervised learning (SSL) and transformer-based encoders to EEG classification. It presents a flexible pre-training framework that supports pre-training on fusion datasets with varying numbers of channels. This thesis proposes grouping each EEG recording into subgroups of n channels, and pre-training by encoding these groups individually, rather than encoding the entire recordings. The proposed pre-training scheme effectively handles data fusion with varying numbers of channels, improving the transferability of learning between datasets.

The main contributions of this thesis are the introduction of a self-supervised learning framework called DECCaTNet[1], which utilizes contrastive learning on groups of EEG channels using a hybrid CNN-transformer encoder, and a preprocessing framework capable of processing large datasets from different sources.

A thorough experimental evaluation is performed by examining the effect of channel grouping and SSL. The findings demonstrate the effectiveness of pre-training an encoder with n channels for EEG data, with an optimal group size of $n = 4$ in the proposed configuration. Results demonstrate an 84.26% classification accuracy on the TUH Abnormal test set. Pre-training the encoder on a larger dataset is found to be beneficial, however, pre-training for an excessive number of epochs leads to degradation of results. The transformer-based encoder performs well within the SSL architecture, although direct comparisons with other encoders are not included.

Overall, this thesis presents a novel approach to EEG signal classification by introducing channel grouping, combined with self-supervised learning and transformer-based encoders. The proposed DECCaTNet model, channel grouping techniques and preprocessing framework contribute to the field of EEG research and offer potential for advancements in multichannel time series analysis.

[1]*Our implementation can be found at* `https://github.com/ostormer/deccatnet`*. For help regarding the implementation, feel free to contact the authors via GitHub.*

Sammendrag

EEG-klassifisering spiller en kritisk rolle innen nevrovitenskap. Den oppdagede begrensningen med overførbarhet og den begrensede bruken av transformer-baserte modeller inspirerte denne studien til å utforske nye tilnærminger som kan forbedre EEG-klassifisering. Denne masteroppgaven utforsker bruken av selvovervåket læring (self-supervised learning (SSL)) og transformer-baserte modeller til EEG-klassifisering. Den presenterer et fleksibelt rammeverk som muliggjør forhåndstrening på kombinerte datasett med varierende antall kanaler. Oppgaven foreslår å gruppere hver EEG-måling i undergrupper av n kanaler og forhåndstrene ved å enkode disse gruppene individuelt, i stedet for hele målingen. Det foreslåtte rammeverket håndterer effektivt kombinasjoner av datasett med varierende antall kanaler, og forbedrer overførbarheten av læring mellom datasett.

Hovedbidragene i denne oppgaven er introduksjonen av et selvovervåket rammeverk kalt DECCaTNet[2], som bruker kontrastiv læring på grupper av EEG-kanaler ved hjelp av en hybrid CNN-transformer-modell, og et preprosesseringsrammeverk som er i stand til å behandle store datasett fra forskjellige kilder.

En grundig eksperimentell evaluering blir utført for å undersøke effekten av kanalgruppering og selvovervåket læring (SSL). Funnene viser at det er effektivt å forhåndstrene modellen med n kanaler for EEG-data, med en gunstig gruppestørrelse på $n = 4$ i den foreslåtte konfigurasjonen. Resultatene viser en klassifiseringsnøyaktighet på 84,26% på TUH Abnormal testsett. Det viser seg å være gunstig å forhåndstrene modellen på et større datasett, men forhåndstrening over et overdrevent antall epoker fører til svakere resultater. Transformer-baserte modeller presterer godt innenfor SSL-arkitekturen, men direkte sammenligninger med andre modeller er ikke inkludert.

Oppsummert presenterer denne oppgaven en ny tilnærming til EEG-signalklassifisering ved å introdusere kanalgruppering, kombinert med selvovervåket læring og transformer-baserte modeller. Den foreslåtte DECCaTNet-modellen, kanalgrupperingsteknikker og forbehandlingsrammen bidrar til videreutvikling av EEG-forskning og tilbyr potensial for fremskritt innen analyse av flerkanal-tidsserier.

[2]*Implementeringen vår kan finnes på* `https://github.com/ostormer/deccatnet`*. For hjelp med implementasjonen kan forfatterene kontaktes via GitHub.*

Contents

Abstract . . . iii
Sammendrag . . . v
Contents . . . vii
Figures . . . xi
Tables . . . xiii
Acronyms . . . xv
1 Introduction . . . 1
2 Background and related work . . . 5
2.1 Electroencephalography . . . 5
2.1.1 Characteristics of time series . . . 6
2.1.2 Characteristics of EEG . . . 6
2.2 Deep learning architectures . . . 7
2.2.1 Convolutional neural networks . . . 8
2.2.2 Attention and transformers . . . 8
2.3 Deep learning architectures in EEG . . . 9
2.3.1 Transformers . . . 9
2.3.2 Hybrid CNN-transformer models . . . 12
2.4 Self-supervision . . . 13
2.4.1 Contrastive learning . . . 14
2.5 SSL for EEG classification . . . 14
2.5.1 Pretext tasks . . . 14
2.5.2 Contrastive learning in EEG . . . 15
2.5.3 Predictive pretext tasks in EEG . . . 16
2.6 Augmentations in EEG . . . 18
2.7 Data . . . 22
2.7.1 Data aggregation . . . 22
2.7.2 Model input selection . . . 24
2.7.3 Preprocessing of EEG signals . . . 25
2.7.4 Artifact and noise removal . . . 27
2.7.5 Class distribution and imbalance handling . . . 28
3 Method . . . 31
3.1 Framework overview . . . 31
3.2 Data . . . 35
3.2.1 Datasets . . . 35

3.2.2 Loading and preprocessing 37
3.3 Pre-training 41
3.3.1 Pytorch datasets and loaders 43
3.3.2 Initial Dataset 43
3.3.3 PathDataset 44
3.3.4 Augmentation 46
3.3.5 Pretext-task 48
3.3.6 DECCaTNet encoder 50
3.4 Fine-tuning 54
3.5 Experimental cycle 56
4 Experimental setup and results 61
4.1 Experimental setup 61
4.1.1 Computation resources 62
4.1.2 Fine tuning parameters 62
4.1.3 Pre-training parameters 63
4.1.4 Optimal number of channels 64
4.1.5 Frozen encoder weights and random initialized encoder . . 65
4.1.6 Transfer learning 65
4.1.7 Adding additional noise 66
4.1.8 Percentage of labeled data 66
4.1.9 Optimize performance on TUAB 67
4.2 Results 67
4.2.1 Initial hyperparameter search 67
4.2.2 Optimal number of channels 68
4.2.3 Frozen encoder weights and random initialized encoder . . 69
4.2.4 Transfer learning 71
4.2.5 Additional noise and harder pretext task 71
4.2.6 Percentage of labeled data 72
4.2.7 Optimize performance on TUAB 73
5 Discussion 75
5.1 Grouping channels 75
5.2 Impact of SSL 78
5.2.1 Frozen encoder weights 78
5.2.2 Transfer learning 79
5.2.3 Percentage of labeled data 79
5.2.4 Optimized performance on TUH Abnormal EEG Corpus (TUAB) 81
5.3 Architecture 81
5.3.1 Pretext task 81
5.3.2 Augmentations 82
5.3.3 Encoder architecture 83
5.3.4 Fine-tuning classifier 84
5.4 Further work 85
5.4.1 Architecture 85

5.4.2 Upscaling 87
5.4.3 Transferability to other problems 88
6 Conclusion 89
Bibliography 93
A Experimental results 107
A.1 Hyperparameter search 107
A.2 Optimal number of channels 109
A.3 Frozen Pre-trained encoders 110
A.4 Number of pre-training epochs 110
A.5 Transfer learning 111
A.6 Harder pretext task 111
A.7 Percentage of labels available during fine-tuning 112

Figures

2.1 Transformer architecture 10
2.2 Self-attention module 11
2.3 Contrastive learning for EEG signals 16
2.4 Augmentations proposed by nuerologists 19
2.5 Augmentations proposed by nuerologists 20
2.6 Weak Augmentations 20
2.7 Strong Augmentations 21
2.8 Common EEG Filters 26

3.1 Overview of proposed framework 33
3.2 Size of datasets 36
3.3 Overview of our preprocessing pipeline 38
3.4 Overview of splitting EEG recordings into channel groups 40
3.5 Pre-training framework 42
3.6 File structure after prerpocessing 45
3.7 Overview of DECCaTNet encoder 51
3.8 Detailed overview of Dual EEG Contrastive Convolution and Transformer Network encoder 52
3.9 Fine-tuning architecture 55
3.10 Experimental cycle 57

4.1 Metrics from channel selection for $n = \{2, 5\}$ 69
4.2 Metrics from channel selection for $n = \{4, 5, 6, 7\}$ 70
4.3 Freeze encoder and pretraining test results 70
4.4 Metrics from pre-training for 100 epochs. 71
4.5 Comparison of metrics from 100 pre-training epochs. 72
4.6 Metrics for experiment with changed augmentations 73

Tables

3.1 Datasets used for pre-training and fine-tuning 35
3.2 Preprocessing parameters 37
3.3 Channel grouping functions 41
3.4 Average time used on pre-training operations 46
3.5 Chosen augmentations 48
3.6 Parameters for contrastive learning (CL)-algorithm 50
3.7 Parameters for Dual EEG Contrastive Convolution and Transformer Network (DECCaTNet)-encoder 54
3.8 Downstream classifier parameters 56
3.9 Parameters for RayTune experimental setup 59

4.1 Experimental setup hyperparameter search for fine-tuning 63
4.2 Experimental setup hyperparameter search for pre-training 63
4.3 Experimental setup hyperparameter search for pre-training 2 64
4.4 Experimental setup optimal number of channels 64
4.5 Experimental setup optimal number of channels 65
4.6 Experimental setup freeze encoder and initalization 65
4.7 Adjusted augmentations to create harder pretext task. 66
4.8 Experimental setup percentage of labelled data 67
4.9 Setup final hyperparameter search 67
4.10 Final result and comparable results 74

5.1 Example number of groups for different grouping functions 77
5.2 Model accuracy on TUAB with percentage of labels compared to relevant papers 80

A.1 Initial fine-tune net hyperparameter search results 107
A.2 First pre-training hyperparameter search results 108
A.3 Second pre-training hyperparameter search results 109
A.4 Results of first *n* channels hyperparameter search 109
A.5 Results of second *n* channels hyperparameter search 110
A.6 Results of frozen encoder weights and random initialized encoder . 110
A.7 Results of number of pre-training epochs 110
A.8 Results of frozen encoder without transfer learning 111

A.9 Results of additive noise. 111
A.10 Results of harder pretext task. 111
A.11 Results of percentage of labels available for fine-tuning. 112

Acronyms

AR average reference

BCI brain-computing interfaces

BCICIV BCI Competition IV

BCICIV1 BCICIV Dataset 1

CL contrastive learning

CNN convolutional neural network

CPC Contrastive Predictive Coding

CV computer vision

DECCaTNet Dual EEG Contrastive Convolution and Transformer Network

DL deep learning

DL-EEG deep learning in electroencephalography

EEG electroencephalography

ELU exponential linear unit

FFT fast Fourier transform

LSTM long short-term memory

ML machine learning

NLP natural language processing

ReLU Rectified Linear Unit

REST reference electrode standardization technique

RNN recurrent neural network

RP Relative Positioning

SEED SJTU Emotion EEG Dataset

SNR signal-to-noise ratio

SOTA state of the art

SSL self-supervised learning

TS Temporal Shuffling

TUAB TUH Abnormal EEG Corpus

TUEG TUH EEG Corpus

TUH Temple University Hospital

Chapter 1

Introduction

Aiding electroencephalography with deep learning Electroencephalography (EEG) is a non-invasive technique that measures neuronal activity in the brain by capturing small differences in electrical potentials in the scalp [1]. This technique plays a crucial role in both the clinical and academic domains in studying various brain processes. However, the inherently noisy nature of EEG signals has presented challenges for accurate analysis and interpretation. Traditionally, EEG analysis has relied on labor-intensive preprocessing pipelines and manual interpretation by domain experts. The advent of deep learning techniques, specifically in the field of EEG analysis (DL-EEG), has shown promising potential in automating and enhancing these processes, thus optimizing the use of expert resources.

Recent advances have demonstrated that deep learning models can achieve performance comparable to or superior to classical statistical methods, while also reducing the dependence on manual analysis [2]. However, supervised deep learning approaches often require large annotated datasets, which can be challenging to acquire in the biomedical domain due to the expertise required for accurate annotation. Consequently, the availability of large-scale labeled EEG datasets, such as the Temple University Hospital EEG Corpus [3], has spurred interest in exploring alternative learning paradigms that can leverage unannotated data.

State of the field Significant progress has been made in the field of DL-EEG. Researchers have improved supervised models by incorporating deeper architectures with more trainable parameters, designing sophisticated convolutional neural networks (CNNs), recurrent neural networks (RNNs), and introducing self-attention mechanisms [4]. Although these advances have shown promise in computer vision (CV) and natural language processing (NLP), they often require extensive labeled datasets for training.

To mitigate the burden of annotation and to improve model generalizability, self-supervised learning (SSL), a machine learning technique that uses unlabeled data, has gained attention. SSL involves learning a latent representation of data from unannotated datasets, followed by fine-tuning using a subset of labeled data for the downstream task. In EEG processing, SSL has garnered interest, with several studies demonstrating superior performance compared to state of the art (SOTA) approaches in various EEG classification tasks. Two prominent categories of SSL architectures in EEG are contrastive models and predictive models. Contrastive models maximize agreement between two augmented versions of an EEG signal, while predictive models reconstruct the original signal from its augmented representation. In EEG, contrastive learning has shown the most promising results so far, with common encoder architectures including CNNs, RNNs, and transformers.

Current challenges and limitations While notable progress has been made in using SSL and transformer-based architectures for EEG analysis, limited research has explored the combination of these concepts. To the extent of our knowledge, only one study has employed a transformer encoder for SSL and subsequent transfer learning [5]. One of the primary objectives of SSL is to learn a robust and generalizable latent representation that can facilitate transfer learning across datasets. However, EEG recordings exhibit variability in terms of montage, the number of channels, and the recording environment, which may require different preprocessing steps. Mohsenvand *et al.* [6] demonstrated improved classification accuracy by applying transfer learning between EEG datasets from different domains, highlighting the potential to leverage SSL in such scenarios. In particular, while the effectiveness of transformer-based architectures has been observed in other machine learning domains, their specific impact and performance in EEG analysis remain relatively unexplored. Thus, there is a need for further investigation into the interplay between SSL model architecture, transfer learning, and choice of augmentations, considering their implications on EEG analysis.

Transfer learning and number of channels Variability in the number of channels between EEG data sets causes difficulties in transfer learning. One common method is to perform pre-training with a limited number of channels shared by all data sets used for transfer learning. However, by following this method, vast amounts of valuable data may be lost. A promising method was proposed by Mohsenvand *et al.* [6], where an encoder was pre-trained on single EEG channels before several encoders were combined to match the number of channels while fine-tuning. The transfer learning challenge was solved; however, as only single-channel encoders were explored, inter-channel dependencies are left to be learned during fine-tuning. The lack of a SSL framework that enables transfer learning between datasets with different numbers of channels while learning inter-channel dependencies is evident.

Contributions

Listed below are the main contributions of this master thesis. They have been designed as part of this research project, which was aimed at answering our research questions. The research questions are listed below the contributions.

C1 A self-supervised learning model, called DECCaTNet, using contrastive learning to pre-train on groups of channels. The encoder of DECCaTNet is a hybrid CNN-transformer model. The model distinguishes itself by splitting each EEG signal into groups of *n* channels and encoding the channel groups separately. This supports dataset fusion and transfer learning more flexibly than previous methods.

C2 A preprocessing framework, built on MNE and Braindecode, capable of preprocessing large datasets from different sources, using serialization and limited parallelization. The preprocessing framework applies a standardized pipeline of rescaling, windowing, resampling, rereferencing, and filtering to all data, before splitting the samples into channel groups of size *n*. The framework can be easily extended to accommodate new datasets.

Research questions

RQ1 *How does pre-training an n-channel encoder perform, and how is it best implemented for EEG data?*

RQ2 *What is the optimal number of channels in each group when using SSL with grouped channels on EEG-data?*

RQ3 *How does a transformer-based encoder perform in an SSL architecture for classifying multichannel EEG data?*

These research questions were formed on the basis of a literature review performed as part of our project thesis. As our main contribution, we propose a novel model architecture that splits the multi-channel EEG input into groups of size *n*, encodes each group independently, and combines the encodings in a classifier. The encoder is pre-trained on data which can be from multiple sources, as long as it can be split into groups of size *n*. Furthermore, to answer our research questions, we needed to implement a preprocessing framework that enables channel grouping and works for data from different sources. This model architecture is intended to open up a new world of possibilities within EEG-research, as it is the first of its kind.

Thesis structure This thesis is organized as follows:

Chapter 1, Introduction, briefly introduces EEG and SSL concepts and explains the objectives of the review.

Chapter 2, Background and related work, describes the background and theory of common concepts and related problems, and explains how they have been addressed in related studies. The characteristics of EEG, deep learning (DL) architectures, SSL methods, and data processing methods are described.

In Chapter 3, Method, our proposed model and preprocessing pipeline are introduced, with explanations of the choices made during their implementation.

Chapter 4, Experimental setup and results, describes the experimental plan and the experimental results. Both architecture optimization, hyperparameter search, and final results are described.

Chapter 5, Discussion, goes into more depth about some of the implementation choices and discusses the results of our model. Both early results, with comparisons to different variations we tested, and final results, which are compared to other studies are discussed. Then, we propose how the architecture can be further developed.

Finally, in Chapter 6, Conclusion, our findings are summarized.

Chapter 2

Background and related work

Introduction In order to cover one subtopic at a time and minimize jumping between topics, we have combined the background theory and related work chapters into one in this thesis. We believe that this makes for an easier reading experience, as for each topic, relevant theory and common challenges are introduced before describing how those challenges have been tackled in related work. This chapter introduces the basics of EEG and how DL is applied to it, before going into depth about SSL. Lastly, it describes the data augmentations used on EEG for SSL and how EEG data is preprocessed for DL.

This chapter is a continuation of our project thesis *Transformers and Self-Supervised Learning in EEG Classification: a Literature Review* [7], which was written as a preparation for this master thesis.

2.1 Electroencephalography

Electroencephalography (EEG) is a noninvasive method to detect small differences in electric potential in the scalp and to read the activity of brain neurons [1]. It is widely used to research many different aspects of the brain in academic and clinical settings. EEG measures the combined electric potential of neurons in a larger area close to the electrode, compared to invasive methods, which can measure neuron activity at a more precise and in-depth level and even measure action potentials in single neurons. Since the EEG signal is noisy, it is possible to get a better picture of the more widespread activity of the brain by combining readings from several electrodes.

It is clear that EEG is valuable for disease diagnosis, treatment, and neurological research in both clinical and academic contexts. EEG recordings are noisy; therefore, traditional analytical methods have relied on preprocessing pipelines with results examined by human experts. Expert labor is expensive, but with the aid of sophisticated signal processing and classification methods, it can be used more effectively.

2.1.1 Characteristics of time series

Numerous methods for learning time series representations are currently in use, and many of them are influenced by well-performing techniques from the CV and NLP fields. Both of the aforementioned domains contain some assumptions that may not apply to time series representation learning. The images in CV exhibit strong spatial correlations. For instance, cropping an image would probably result in minimal information loss, as missing parts are often given by its surrounding pieces. Contrary to images, a time series cropped into a subsequence may have a different distribution and semantics than the original time series from which it was derived. This is because the properties of time series may change over time.

In addition, for a model to be able to infer both long- and short-term trends, learning time-series representations requires knowledge of various temporal patterns. The existence of some patterns in both fine- and coarse-grained representations is known as scale invariance. Although some approaches focus on the inherent temporal differences of the data, others create frameworks that encourage learning various temporal dependencies [8–10].

Some time series are multivariate, which means that they were obtained simultaneously from multiple related time series. Spatial dependency refers to the information that is hidden in similarity or dissimilarity between pairs of signals. For the purpose of producing effective multivariate time series representations, it is crucial to comprehend domain-specific spatial patterns.

2.1.2 Characteristics of EEG

EEG signals have a high temporal resolution due to the common selection of 128 to 1024 Hz as the sampling frequency range [11]. Given that studying frequency bands in the range of 1 Hz to 50 Hz is sufficient for emotion recognition [12], keeping the original sampling frequencies could lead to excessive temporal resolution [13]. Each EEG electrode in a typical scalp EEG configuration collects signals from nearby regions, resulting in coarse spatial resolution (a few centimeters). Therefore, EEG signals have a low spatial resolution and a high temporal resolution [14]. With traditional techniques, the frequency domain has become the main topic of interest in EEG data because various frequencies have been associated with various measures of brain activity [15].

Learning effective representations is made more difficult by the added complexities associated with EEG time series. The signal-to-noise ratio (SNR) of EEG is poor [16, 17]. Multiple layers of skin and bone cover the brain activity that can be measured with EEG. As a result, when the electric potential reaches the scalp-mounted sensors, it is contaminated with noisy data. Uncontrollable and ambient physiological noise is often added in addition, often resulting in more apparent artifacts. Roy *et al.* [2] found that 63% of the DL techniques used for EEG classification did not systematically remove EEG artifacts despite the numerous guaranteed problems. A common intuition in AI is that spending time on pre-processing is vital for good results. However, results may improve by applying models directly to raw EEG signals [2].

EEG is a non-stationary signal, as the temporal dependencies fluctuate over time, even within a single subject [18, 19]. As a result, temporally constrained data may cause models to overfit on what appears to be a stationary signal. However, overfitting could result in poor generalization of data from the same subject recorded at a different time because temporal dependencies are likely to change over time or within the next recording session. This is a problem that researchers frequently encounter since EEG data are frequently constrained in terms of both the number of subjects and the recording time. Understanding intra-subject relationships is important for brain-computing interfaces (BCI) research, but it is also important to consider when performing other EEG-realated classification tasks.

Inter-subject variability is also important to consider when working with EEG-data. Physiological variations in subjects lead to inter-subject variability. As a result, different abnormal brain activity, especially in terms of magnitude, appears different for each subject. In DL, deep networks are often assumed to work as subject-invariant feature extractors. However, the generalizability of models that classify between subjects can be impacted by the magnitude variation [20]. To learn subject-invariant representations given the described properties of EEG signals, adversarial learning has been proposed [21]. Adversarial learning penalizes the model for learning representations which enables correctly prediction the subject's identity, encouraging subject-invariant representations. The age-related slowing of EEG signals is another characteristic that some research takes into account [22]. Due to this phenomenon, the EEG signals of younger subjects should differ from those of older subjects when it comes to temporal characteristics. In addition, the subject's age is often not regarded as an expert label, opening new opportunities for SSL.

Each EEG signal consists of several signals, referred to as channels. Each channel shows the difference in voltage between an electrode and the common reference, usually a common reference electrode. The common reference can also be digitally calculated, such as the average of all electrode voltages. The number of sensors (electrodes) used in the corresponding recording session determines the number of channels. Transfer learning between various EEG-related tasks is complicated due to the inconsistent number of channels in different data sets. As a result, some transfer learning networks do not make use of all available channels to handle different downstream tasks.

2.2 Deep learning architectures

Recent research has demonstrated that models based on DL can perform on par with or better than traditional statistical SOTA methods in many tasks while avoiding expensive expert manpower [2]. In the past ten years, interest in the subfield of using DL to improve EEG processing (deep learning in electroencephalography (DL-EEG)) has increased. Recent releases of several large, publicly accessible EEG datasets, such as the Temple University Hospital (TUH) EEG Corpus [3] and the

National Sleep Research Resource [23], are likely key contributors to the increased interest of the DL research community.

In recent years, deep learning has made significant strides in processing complex and information-rich data, including images, audio signals, and text [24]. As a result, researchers have explored the application of DL techniques to improve the processing of EEG data. Various DL models for EEG classification have been developed using different neural network architectures such as convolutional neural networks (CNN), recurrent neural networks (RNNs), and fully connected networks. Recently, transformers and their derivatives, which rely on self-attention layers, have demonstrated SOTA performance in NLP and CV tasks [4]. As a result, these models have been applied to EEG data analysis with promising results. In general, deep learning techniques have shown potential in the analysis of EEG data, and researchers have proposed various models and architectures to achieve high classification accuracy. The use of transformers and self-attention mechanisms has become increasingly popular and has shown promising results in various domains, including EEG analysis.

2.2.1 Convolutional neural networks

CNNs recognize learned patterns in input data by passing a fixed-size filter with fixed weights over it, thereby identifying the location of the pattern in the input [25]. CNNs can produce multiple channels by applying convolutional filters in parallel and deeper networks by stacking multiple layers. To address the problem of vanishing gradients in deep networks, modern CNN-based approaches incorporate residual shortcut connections [26]. In particular, ResNet [27] achieved SOTA performance in computer vision using this idea.

Other advances in CNNs include the combination of filters of different sizes in the same layer, as demonstrated by the Inception network [28, 29], to better recognize objects independent of their size. CNNs can also incorporate attention, which learns which part of the input to pay attention to [30]. Although CNNs were initially developed for computer vision, many of the concepts and methods can be adapted for time-series data. Since time-series data have one channel dimension and one temporal dimension, adjusting a CNN for time-series data involves treating the temporal dimension as a spatial one, and the time-series channels as the color channels of an image. This adjustment allows two-dimensional convolutional layers to be replaced by equivalent one-dimensional ones.

2.2.2 Attention and transformers

Self-attention is an attention mechanism that allows different positions in an input sequence to be related to each other, resulting in a sequence representation based on internal relationships. Self-attention has been successfully used in conjunction with RNNs [31, 32] or CNNs [30].

More recently, transformers [4], a model that relies mainly on self-attention and completely avoids RNNs and CNNs, have gained widespread attention in both

NLP and CV domains. As of our literature review in October 2022 [7], transformer-based models were state-of-the-art in language modeling [33] and image classification [34] among other tasks. In addition, the large NLP model GPT-3 [35], recently made famous by ChatGPT, is also based on the transformer architecture and the attention mechanism, which enables ChatGPT to generate high-quality text that adapts to the input and context provided by the user.

Transformers have a significant advantage over CNNs in that they can more easily learn long-range dependencies in the data, as the number of operations required to relate two positions is fixed independent of the distance between them. The success of transformers in other domains and their ability to model long-range dependencies have motivated researchers to explore their application in time series and signal processing tasks. In recent years, several transformer variants have been developed that have the potential to significantly improve the state-of-the-art performance of some tasks, such as time series forecasting and classification [36]. The transformer architecture will be explained in-depth in subsection 2.3.1,

2.3 Deep learning architectures in EEG

In the field of deep learning for EEG processing, selecting an appropriate neural network architecture is a critical step. The selection process may depend on various factors, including the specific task at hand, properties of the dataset, and intended use of the model. The popularity of different model architectures in the DL-EEG field has evolved rapidly over the past 15 years [2]. In their study, Roy *et al.* [2] analyzed DL-EEG articles published between 2010 and 2018 and observed that earlier articles commonly employed fully connected networks and relied heavily on pre-processing. However, as the DL field progressed, CNNs for CV demonstrated impressive results, and DL-EEG articles soon followed this trend.

During the era when RNN-based models, such as long short-term memory (LSTM) networks, were considered the SOTA for NLP and speech recognition [37], many researchers developed RNN-based models for EEG classification [38, 39]. From 2015 to 2019, DL approaches to EEG classification were mainly divided between RNN-based sequence-focused models and CNN-based spatial-temporal proximity-focused models, and their combinations.

2.3.1 Transformers

Recently, with the success of transformer models in CV, NLP, and speech recognition domains [4, 40], it is natural for time series classification and DL-EEG to follow this trend. Although transformers typically excel when they can utilize self-supervision, they have recently also been successfully applied to supervised learning problems in time series classification [9, 41, 42]. This success is believed to be due to the multi-head attention layers' ability to learn long-range dependencies.

The articles reviewed in this study that used supervised transformers all combined CNN and transformer architectures, as discussed in detail in section 2.3.

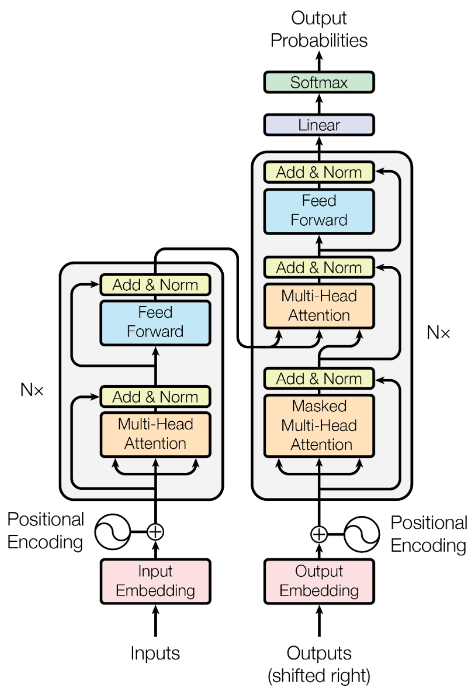


Figure 2.1: The transformer architecture. The encoder is on the left and the decoder is on the right. *Source: [4]*

However, two studies also experimented with transformer models that did not include convolutional layers and were based solely on the transformer architecture to compare with their hybrid variants [9, 42]. In both studies, the hybrid models outperformed the pure transformer models, but only by a small margin.

Transformers are most effective when they are able to perform self-supervised pre-training. Self-supervision has been used to achieve state-of-the-art performance in NLP and CV, and researchers are exploring ways to leverage it for EEG as well. The transformer model was originally designed as an encoder-decoder architecture designed for self-supervision using a reconstructive pretext task, with the encoder being the component of interest for downstream tasks.

The transformer encoder [4] consists of attention modules stacked on top of each other to create a deep learning model that avoids convolutional or recurrent layers. The input is first embedded, and then positional encoding is applied before passing it through the attention modules. The transformer model is illustrated in Figure 2.1, with the encoder on the left. For EEG and time series data, there are several approaches to positional encoding. [42] and [9] explore three categories: relative positional encoding, channel correlation positional encoding, and learned positional encoding. Relative positional encoding involves embedding a sine or cosine function of the channel number and time step to each position of the EEG recording. Channel correlation positional encoding calculates cosine similarities between a central electrode and all other electrodes for every sample and replaces the channel number in the sine and cosine formulas used for the relative positional encoding technique. With learned positional encodings, a trainable matrix with the same dimensions as the input samples is embedded in the input, initialized with random values, and trained by backpropagation. The same learned matrix is then embedded in all inputs.

Multi-head attention The core component of the transformer encoder is the stack of attention modules, represented by the grey box on the left in Figure 2.1. The inputs are passed through the first attention module, with or without positional embedding, and subsequently through all attention modules in a similar manner to regular deep neural network layers. Each attention module consists of two sub-layers, namely a multi-head self-attention mechanism, followed by a fully connected network. A residual connection is used around each sub-layer, and these shortcut connections are added to the sub-layer output and normalized. As mentioned in section 2.2, residual connections address the vanishing gradient problem and improve the performance of many deep networks [27, 28].

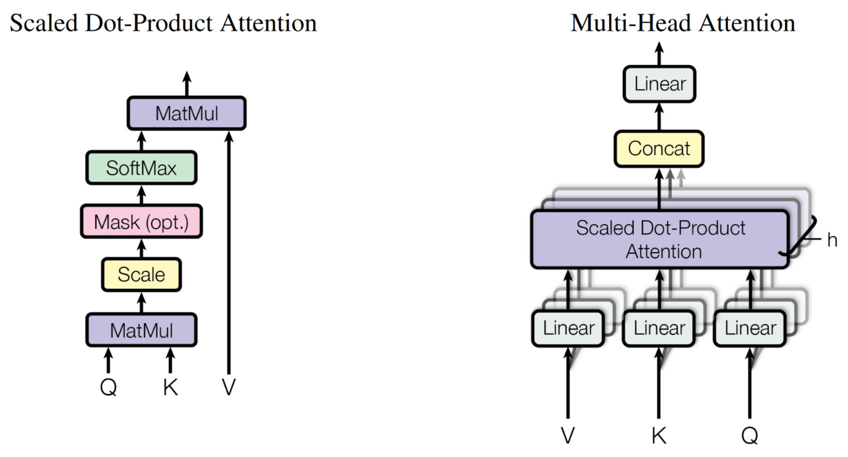


Figure 2.2: Scaled Dot-Product Attention and Multi-Head Attention, both used in the transformer architecture. *Source: [4]*

The multi-head attention sub-layers, as shown in Figure 2.2, are composed of several self-attention heads applied to an input vector. The scaled dot-product self-attention operation is performed on an input vector by computing the dot-product of all possible input position pairs, scaling the values by dividing each value by $\sqrt{d_k}$, where d_k is the key dimension of the input, and then applying a softmax function to derive the weights. The original transformer was initially developed for NLP, where the input is a vector of word embeddings representing a one-dimensional string of words, and each word is compared with each other word in the sequence using self-attention. However, for EEG data, the input is a time series, usually multivariate, and therefore must be handled differently. Attention can be applied across all channels at set time points, across all time points for each channel, or a combination of both. This will be further discussed later in this section.

The attention weights are then passed to the next sub-layer, which is the position-wise feed-forward network. The network has two fully connected layers with a Rectified Linear Unit (ReLU) activation function between them, as shown in Equation 2.1. The layers have identical weights for each position but distinct weights between layers. This is equivalent to having two convolution layers with kernel size 1 but multiple filters. During training, only the weights in this feed-forward network are updated via backpropagation.

$$\text{FFN(x)} = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.1}$$

Vaswani *et al.* found that instead of applying the scaled dot-product attention as one operation to the entire input with size d_{model}, it is advantageous to linearly project the input into h different, learned, linear projections with size d_{model}/h. The attention function can be executed in parallel on each of these projections. Although h different projections need to be learned, this approach is more computationally efficient because the complexity of the self-attention module increases with $O(d^2)$ and $h(d_{model}/h)^2 < (d_{model})^2$.

2.3.2 Hybrid CNN-transformer models

As the original transformer architecture was designed primarily for NLP, it is not suitable for processing high sample rate time series with multichannel input without modifications. Therefore, many researchers have introduced modules using convolution and/or pooling layers before the transformer module to effectively convert the input signal into a sequence of “subsample embeddings” [5, 9, 41, 42]. Each element in this sequence represents an embedding of the multichannel EEG signal in a given time slice and is fed into the stack of attention modules with positional encoding.

Wu *et al.* [41], create a model where convolution layers are integrated into the scaled dot-product attention module itself, going beyond simply adding CNNs to the transformer. The authors employ squeeze-and-excitation convolution submodules [43], which enhance the representational power across different channels by learning channel features with two convolution layers. The first convolution layer reduces the number of channels by a factor of $\frac{1}{2}$, hopefully learning relevant features, while the second layer restores the original number of channels.

Conformer Another architecture combining CNNs and transformers that is gaining attention in the field of EEG is the Conformer [44]. The Conformer architecture was first introduced in 2021 by Google researchers as a novel neural network architecture that combines the strengths of convolutional neural networks (CNNs) and transformers. It is designed to handle sequential data with variable lengths and can capture both local and global dependencies in the data.

Recently, the conformer has been applied to EEG data, specifically in the context of emotion recognition and motor imagery tasks [45]. Similar to the hybrid CNN-transformer models, the conformer uses subsampling to reduce the sample rate of the input signal, which is then fed into the transformer modules. One advantage of the conformer over traditional CNNs is its ability to capture long-range dependencies in the data, making it suitable for analyzing EEG signals that may exhibit complex patterns over longer time periods.

Overall, both the hybrid CNN-transformer models and the Conformer architecture show promising results in the field of EEG data analysis, demonstrating the power of combining different neural network architectures to tackle complex problems.

Spatial and temporal modeling When using transformers on multivariate time series data, two common approaches are applying multi-head self-attention spatially (channel-wise) or temporally (time-wise). Some researchers have used a combination of both, by training a fully connected network on the combined outputs of a spatial model and a temporal model [42]. Wu *et al.* combine spatial and temporal learning by having separate channel attention and time-slice attention modules [41]. Each of these modules has their initial convolution filters shaped either along the channel axis or the time axis, before the data is embedded and passed through a lightly modified transformer architecture. The outputs of both modules are combined and passed through a final module, in a similar fashion to [42].

2.4 Self-supervision

SSL aims to solve the problem of supervised learning models requiring large amounts of labeled data. Labeling data is a resource-intensive and expensive process that sometimes requires the precious time of educated experts, which are in short supply in many fields. Although there are fully unsupervised approaches, such as reinforcement learning, unsupervised learning cannot be applied to all problems.

Self-supervised learning aims to use unlabeled datasets to learn a good latent representation of the data. Training a classifier from this latent space is often much simpler than from the original input and can achieve good results when fine-tuned on only a small portion of the data that has been labeled [46].

Manually reviewing and labeling EEG recordings is time-consuming and requires many work hours from neurologists around the world. Ensuring that data sets follow privacy regulations, are compatible with other data, and are of high quality further increases the demand for resources. SSL is proposed as a way to reduce the need for labels in several domains, including health data [47]. Using SSL, our goal is to learn meaningful representations of the EEG recordings simply by using the data itself. This, in turn, can be useful for downstream tasks, in our case time series classification. SSL often includes a pre-training task, for example, reconstructing a masked image [48]. By augmenting an input (for example, masking), spatial and temporal features can be learned from the data itself, but the type of augmentation must be adapted to the input data [49]. This happens because different data types (pictures, time series, or language) have different spatial and temporal dependencies.

2.4.1 Contrastive learning

Contrastive learning (CL) has emerged as a popular methodology within SSL and has achieved state-of-the-art results in both image and time series classification, comparable to supervised approaches [10, 50]. The primary advantage of CL is its discriminative nature, achieved by creating positive pairs using appropriate data augmentations during training (such as shifting, masking, or adding noise), forcing similar samples closer and dissimilar pairs further apart in the latent space. This results in high-quality, generalized representations that are useful for downstream tasks like classification. A more detailed explanation of contrastive learning will come in subsection 2.5.2, explained using EEG data as an example.

2.5 SSL for EEG classification

SSL has shown remarkable success in natural language processing and computer vision with the use of transformers as feature extractors. Recently, SSL has also gained attention in the field of EEG classification. In this section, we will focus on SSL and transformers in EEG classification.

We have mainly reviewed research papers related to the TUH EEG dataset, but we have also considered other datasets since some studies have demonstrated the transferability of SSL methods between different datasets and tasks [6, 8, 51]. SSL for EEG classification differs in two main aspects: the pretext task is different, and the augmentations applied to the EEG data differ from other domains.

2.5.1 Pretext tasks

The pretext task is the core concept of SSL, as it provides an unsupervised method for training deep neural networks. Pretext tasks are domain-specific tasks that neural networks can solve. Unlike supervised learning, which requires labeled datasets, SSL algorithms leverage the data to obtain "free" labels. Pretext tasks come in many forms, including contrastive learning and reconstructing masked data, as mentioned earlier. In contrastive learning, the model learns to distinguish between similar and dissimilar pairs of samples, while in data reconstruction, the network attempts to predict the missing parts of a data sample. These "free" labels serve as proxies for the true labels and enable the network to learn meaningful representations of the data.

EEG-specific pretext tasks The choice of pretext task is critical in determining which properties of the data are learned and can significantly impact the downstream classification performance of SSL. According to Wagh *et al.*, EEG requires domain-specific pretext tasks, and they present three EEG-adapted pretext tasks: hemispheric symmetry, behavioral state, and age contrast (where age is not considered an expert label). All these tasks use the unique properties of EEG as “free” labels.

Hemispheric symmetry utilizes the fact that healthy brains exhibit similar brain activity across both the left and right hemispheres. Thus, the pretext task penalizes the model for learning different representations of each hemisphere. The behavioral state pretext task is based on the assumption that spectral power ratios in the central brain can be used to estimate the behavioral state of the patient. This estimate becomes the target value and forces the model to learn representations sensitive to behavioral states. The age contrast task is based on the importance of age in modulating brain activity [52]. Age-related slowing in EEG should lead to different representations for younger and older patients. To achieve this, the authors use contrastive triplet learning [53], which forces similar age groups to be closer in the latent space and different age groups to be further apart.

2.5.2 Contrastive learning in EEG

Contrastive learning is a popular technique for self-supervised learning in EEG. Several studies, [6, 54, 55], have explored the use of contrastive learning in EEG. In particular, Jiang *et al.* used a similar contrastive approach to that described in the first paragraph of subsection 2.4.1, but with the addition of cosine similarity as the distance metric between vector space representations in the loss function. This approach aims to maximize the agreement between two representations, as described in [6]. This method is based on simCLR [56], which is a vision-based contrastive framework. The loss and similairty functions used are presented in Equation 2.3 and Equation 2.2, respecitvely. The contrastive loss calculates the loss for a positive pair, (e_i, e_j), with respect to all other encodings in the batch, which serve as negative pairs to e_i. It is important to note that this EEG inspired approach does not rely on a domain-specific pretext-task but depends on domain-specific augmentations, which will be discussed in section 2.6.

$$Cosine(x,y) = \frac{x \cdot y}{|x||y|} \tag{2.2}$$

$$\mathcal{L}(e_i, e_j) = -\log \frac{\exp(Cosine(e_i, e_j)/\tau)}{\sum_{k=1}^{2BS} \exp(Cosine(e_i, e_k)/\tau)} \tag{2.3}$$

In the studies by Mohsenvand *et al.* [6] and Jiang *et al.* [54], negative pairs are obtained by considering all other pairs in the given mini-batch as dissimilar. A visualization of how latent space representations are handled by contrastive loss

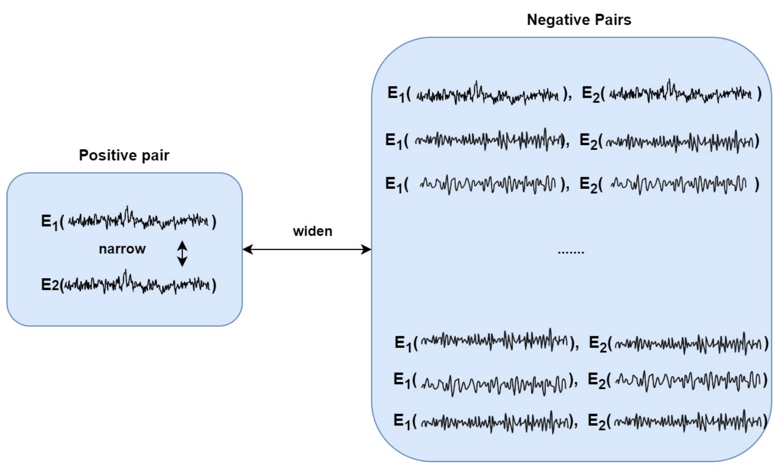


Figure 2.3: Visual representation of contrastive learning for EEG-signals. $E_1()$ and $E_2()$ represent an encoding of a positive pair after applying aug_1 and aug_2.

can be seen in Figure 2.3. The resulting latent space is encouraged to clearly distinguish EEG-signals containing disimilar semantic information. However, this approach may cause samples with similar characteristics to be pushed further apart in the vector space, leading to dissimilar representations even when they belong to the same class. In EEG, this could result in two samples from the same subject with seizures being forced to have dissimilar representations. To address this issue, it is possible to increase the number of positive pairs in the mini-batch by doing class aware contrastive learning [57, 58], which can improve the performance of downstream tasks.

To promote the learning of several EEG properties at once (see subsection 2.1.2), it is possible to combine multiple contrastive learning pretext tasks. For example, by combining contextual and temporal contrastive loss, representations that capture temporal features while being discriminative can be created [58]. Temporal contrastive loss is similar to contrastive loss, but it uses the representations of predicted future time steps to maximize and minimize similarity between latent space representations, instead of evaluating the current time step representations. Ultimately, both normal and temporal contrastive losses are combined in the loss function, which encourages the model to consider both temporal and discriminative features.

2.5.3 Predictive pretext tasks in EEG

While predictive pretext tasks are not as clearly defined as contrastive learning, they involve generating latent space representations by predicting some property or target, reconstructing missing values, or forecasting future values [59, 60]. As many downstream tasks involve the prediction of a label or future values, generating representations through similar problems can improve the transferability to downstream tasks [8, 61].

Temporal dependency based pretext tasks For example, in sleep stage classification, a predictive pretext task inspired by the downstream task could involve predicting whether EEG signals are close in time or not, as signals from the same sleep stage are likely to be close in time [62]. This mimics the downstream task of classifying different sleep stages.

The contrastive learning approach can also be used in predictive pretext tasks in order to learn discriminative features. This could be done by selecting positive and negative pairs and predicting their classification instead of altering similarity directly with a contrastive loss [8, 62]. Banville *et al.* proposes several ways of creating pairs, where each approach focuses on learning temporal EEG properties. One approach to incorporating temporal dependencies in predictive pretext tasks is to focus on the time-based distance between samples when creating pairs. This helps the model learn that temporal dependencies change over time. Another approach is to shuffle the order of some samples and predict whether shuffling has been applied or not. Finally, another proposed method does not utilize pairs but groups of samples, where the model must predict which sample follows a given sequence. This encourages the model to learn representations that can identify similar temporal dependencies from a larger group instead of pairwise identification.

Frequency based pretext tasks In order to model EEG signals, it is important to understand their temporal and spectral properties (see subsection 2.1.2). Ko and Suk proposed two predictive pretext tasks which were designed to specifically focus on learning representations dependent on temporal and spectral properties. To avoid the challenge of learning both representations simultaneously, each property is learned separately. Spectral properties are related to different frequency ranges [63], and the pretext task is designed to learn representations of these. The proposed task is called stop band prediction, where fake EEG samples are generated by removing signals in specific frequency ranges. The model is then trained to identify the removed frequency bands in comparison to the original EEG sample, thereby forcing it to learn the unique patterns associated with each frequency band.

Being able to detect both short-term and long-term patterns is crucial for various downstream EEG tasks [64, 65]. In many models, local temporal embeddings are commonly applied, which inherently capture short-term patterns. However, to learn long-term temporal patterns, a temporal trend identification pretext task can be employed [59]. This involves adding transformations to the data samples to create raw, stationary, trend-stationary, or cyclostationary versions of the data. The pretext task is then to identify which transformation each sample has undergone. To distinguish between stationary and cyclostationary samples, temporal features from a global standpoint are required. Therefore, these two pretext tasks can enhance the latent space representations by incorporating both temporal and spectral features.

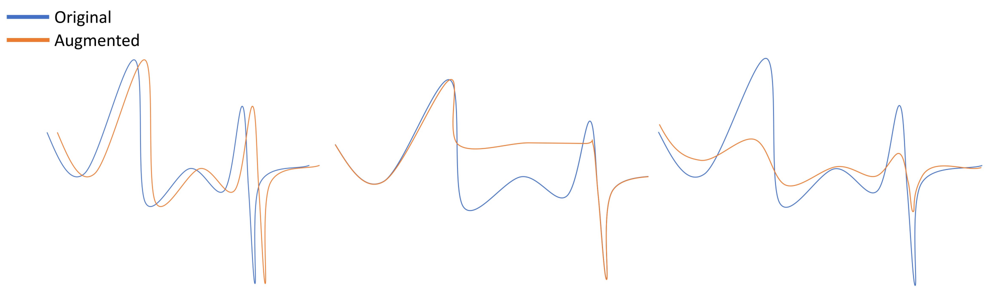
Reconstruction based pretext tasks Reconstruction-based pretext tasks are commonly used in computer vision and natural language processing, but they have also been applied in multivariate time series analysis [61] and EEG representation learning [66]. To make the reconstruction task more challenging, an augmentation technique (section 2.6) is often used. Although reconstruction is typically combined with other pretext tasks, it can also be used as a standalone pretext task in EEG [66]. The main objective is to predict the original signal given a noisy, or otherwise augmented, signal, and the mean absolute error is used as a loss function. This approach leads to the development of robust EEG representations, as domain-specific properties must be learned to perform accurate reconstructions from noisy data [66].

2.6 Augmentations in EEG

A well-designed pretext task is not sufficient alone to learn useful latent space representations for the downstream task. Good augmentations are needed to create positive pairs or predictive targets, as contrast or reconstruction using exact copies are too easy tasks. The described tasks would not enable the model to learn useful EEG semantics for downstream tasks and as a result would be useless for SSL. To harness the power of generally applicable pretext tasks, combining them with good augmentations, preferably domain-adapted ones, is vital [56].

In this context, an augmentation is an operation that changes one or several properties of an EEG signal. By altering the properties of signals, the model uses the pretext task to learn underlying structures of the signal. In order to learn meaningful representations of EEG signals, the augmentations should not alter the semantic information of the signals, but rather alter the numerical values when transforming signals. In this way, augmentations encourage the model to shift its focus from value-dependent representations to semantic-dependent representations. Carefully designed augmentations can therefore increase the generalization capabilities and encourages overlooking distinct values. With EEG's challenges within inter-subject variability in mind, applying the correct augmentations can be a powerful tool. In this section, we will introduce several augmentations techniques utilized in EEG classification today.

Augmentations do not necessarily have the domain transferability of pretext tasks. For instance, color distortion and rotation used in computer vision are not directly applicable to time series data, and equivalent augmentations should be discovered. Commonly, time series adapted augmentations are utilized in the EEG domain as well [10, 58]. However, similarly to time series, the EEG domain has its own set of challenges, such as inter-subject variability and non-stationary signals, which in some cases requires augmentations adapted to EEG data [6, 62].



(a) Time Shift **(b)** Masking **(c)** Scaling

Figure 2.4: Augmentations proposed by nuerologists

Common EEG augmentations We have discovered several EEG-related augmentations used in relevant papers. These are presented in Figures 2.4 to 2.7. Even though several augmentations are commonly used, there is no golden rule or standard framework within SSL for EEG, as each dataset and pretext task requires its own unique solution to achieve the best possible results. In this section we look at augmentations used in different studies, and how they are applied to SSL

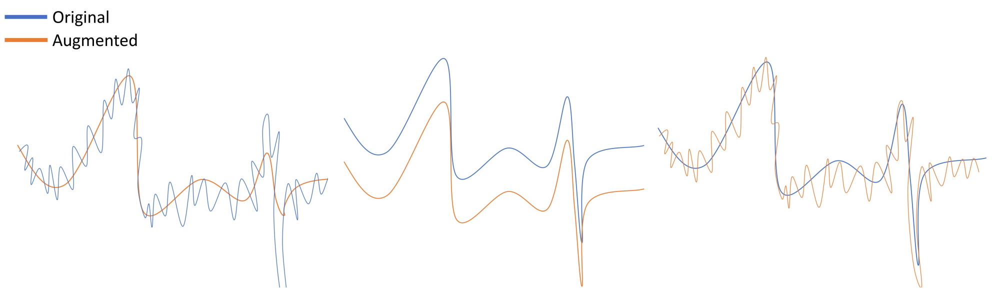
Neurologist-proposed augmentations Domain-specific augmentations were proposed by Mohsenvand *et al.* To ensure that data augmentation techniques used on EEG signals were appropriate, EEG specialists were consulted and asked to propose augmentations that would preserve the interpretability of the EEG signals [6]. The suggested augmentations include scaling, time shift, DC shift, masking, jitter, and band-stop filtering. If specialists are able to accurately classify an augmented signal, it suggests that the relevant semantic information for classification has been preserved. These augmentations are presented in Figures 2.4 and 2.5.

Time shift (Figure 2.4a) involves shifting the EEG signal along the time axis by a certain number of samples or time units. This can help simulate temporal variations and enable the model to recognize the same patterns at different time points.

Masking (Figure 2.4b) randomly sets a portion of the EEG signal to zero, simulating missing data or artifacts in the recording. This technique can help the model to better handle noisy or incomplete data. Additionally, learning to reconstruct missing data forces the model to learn underlying semantic structures.

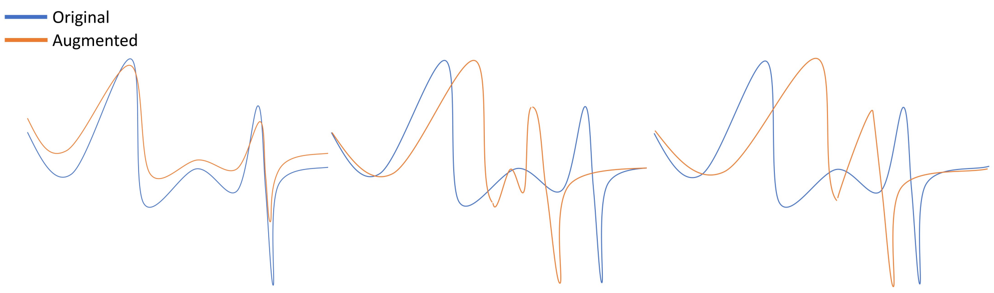
Scaling (Figure 2.4c) involves multiplying the amplitude of the EEG signal by a constant factor to adjust its magnitude that may vary between different recordings. This can enable the model to better capture relevant features at different scales. With the properties of EEG, described in subsection 2.1.2, in mind, such as inter-subject variability and age-related slowing, scaling may lead to better generalization.

Another augmentation, proposed by neurologists and capable of better han-



(a) Band-Stop Filter **(b)** DC Shift **(c)** Jitter

Figure 2.5: Augmentations proposed by nuerologists



(a) Average Filter **(b)** Time Warping **(c)** Cutout&Resize

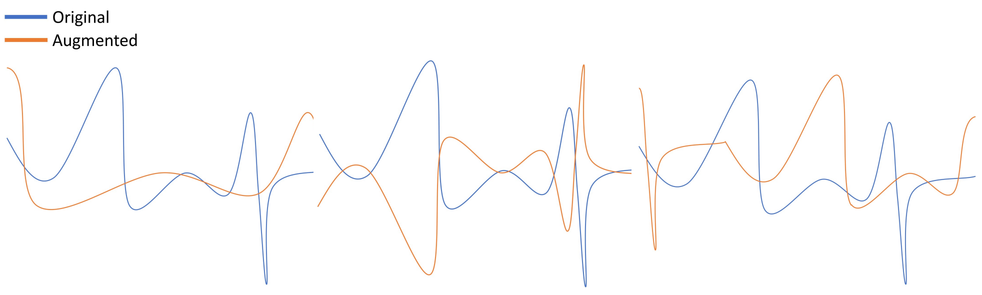
Figure 2.6: Weak Augmentations

dling inter-subject variability, is DC shift (Figure 2.5b). DC shift adjusts the baseline level of the entire EEG signal by adding a constant value. This technique can help the model handle different intensity levels in the data.

Band-stop filtering (Figure 2.5a) removes a specific frequency range from the EEG signal. Learning to create robust latent-space representations without all aviable frequencies enables the model to better handle data that contains noise or artifacts. AS previously stated in subsection 2.1.2, EEG-signals have a high noise-to-signal ratio. Robust representations capable of handling noisy data are therefore beneficial for the downstream task.

To further prevent noise from dominating latent space representations, neurolgists suggest introducing jitter (Figure 2.5c) as an augmentation. Jitter adds random noise to the EEG signal, which forces the model to focus on temporal and spatial dependencies rather than value-specific properties.

Weak vs. strong augmentations According to Eldele *et al.* a weak augmentation (Figure 2.6) applies limited changes to the distinct shape of the signal. Examples are the augmentations time shift and the DC shift presented previously. Strong augmentations (Figure 2.7), on the other hand, alter the shape of the signal but keep some temporal semantics intact, such as permutation (Figure 2.7c) or horizontal flip (Figure 2.7b). When performing permutations, the signal is split into



(a) Crop&Resize **(b)** Horizontal Flip **(c)** Permutation

Figure 2.7: Strong Augmentations

M chunks and then randomly reorganized. The original signal is not easily recognizable; however, the semantics of each segment is kept intact. For horizontal flip, each signal is simply flipped around it's x-axis, keeping useful frequency- and value-knowledge intact while altering the shape. Crop&rezise (Figure 2.7a) relies on the assumption that if the target event can be derived from the entire signal, it should also be obtainable from a subpart of the original signal.

In particular, the weak augmentations presented in Figure 2.6 can be considered as both strong and weak augmentations depending on the parameters chosen for each augmentation. For example, cutout&resize (Figure 2.6c), where a segment is removed and substituded by resizing the remaining signal, is highly dependent on the segment size. This is also the case for some domain-specific augmentations presented by neurologists, such as jitter, which depends on the amplitude of the added noise. As a result, several studies have conducted preliminary studies to determine the augmentation parameters [30, 54, 58]. Models were pre-trained on limited data with different fixed augmentation parameters before results on the downstream task were obtained. By doing this, the most promising augmentation parameters could be used for pre-training on the entire dataset.

Combining augmentations Mohsenvand *et al.* were advised by neurologists to randomize the strength of each augmentation within given ranges for each augmentation. After this, two augmentations were randomly selected from the already proposed augmentations and applied to the EEG signals to create a positive pair. The combination of augmentations or the application of the same augmentations to each input was not discussed with specialists; however, the combination of several augmentation techniques can lead to increased performance [54, 57, 67].

In contrast to Mohsenvand *et al.* [6], Eldele *et al.* [58] conducted an ablation study in order to select a fixed set of augmentations to apply for each input. Interestingly, performance was partially increased by combing two and two augmentations leading to stronger augmentations. The increase in performance varied with the different possible combinations, but all the augmentations performed better when combined with jitter. Notably, the standard deviation (*i.e.* the strength) of the jitter is considerably smaller than when it is applied alone. The final augmentation set was permutation+jitter (strong augmentation) and scale+jitter (weak augmentation). Jiang *et al.* did not experiment with augmentation combinations, however, the final augmentation set that led to the best performance was permutation and crop&resize. This indicates that permutation, which was not proposed by neurologists, is a useful augmentation for contrastive pretext tasks.

There is no golden standard for augmentations, and each challenge requires a unique approach. Testing all possible combinations of augmentations on all data requires a vast amount of computing power. As a result, it is common to report results for each composition on limited data or a smaller subtask. A common solution is to compare the augmentations individually and then select the best performers as an augmentation base for randomized selection [67]

2.7 Data

In order to achieve satisfactory performance in supervised or self-supervised deep learning tasks, it is essential to have a dataset of sufficient size and quality. However, when working with EEG data or medical data in general, we are often faced with similar challenges. Collecting such data can be resource-intensive and expensive; obtaining crucial data related to rare conditions or events can be particularly difficult; and ensuring the privacy and rights of the subjects while sharing the data is a crucial concern. Furthermore, proper data acquisition, examination, and labeling of data require trained personnel, whose availability is often limited.

Additionally, in order to increase data quality, it is often normal to do certain preprocessing steps and to remove noise and artifacts. Relevant theory and previous approaches to EEG dataset handling and preprocessing are outlined in this section.

2.7.1 Data aggregation

Dataset differences and challenges There exists a variety of large-scale openly accessible EEG datasets that are deidentified, facilitating the application of deep learning and transfer learning techniques. However, these datasets originate from diverse settings, including clinical epilepsy, sleep staging, and emotion recognition. Consequently, EEG recordings have different characteristics as a result of these diverse settings. Despite the inherent dissimilarities among these datasets and their respective patterns, evidence suggests that pre-training on a combination of diverse datasets enhances the performance of subsequent tasks targeted at specific problems. Mohsenvand et al. Mohsenvand *et al.* demonstrate that combined pretraining on sleep data, abnormality detection, and affective recognition yields improved performance across all three domains compared to pre-training solely on data collected within the relevant setting.

One challenge encountered during the implementation of transfer learning lies in the disparate recording setups employed across different datasets. Factors such as the sample rate, number of channels, and EEG montage vary between datasets or even within the same dataset. The variation in the number of channels is commonly addressed by excluding channels that are not shared among all utilized datasets [8, 51], thereby enforcing a consistent input shape for all data sources in the model. Another approach, as employed in Mohsenvand et al. [6], involves the design of a feature extractor for a single EEG channel, enabling the learning of a latent representation from a chosen channel in a self-supervised manner. Consequently, the single-channel feature extractor can learn from relevant datasets, regardless of the number of channels or labeling scheme.

TUH EEG Corpus The largest openly available dataset in the field of epilepsy research is the TUH EEG Corpus (TUEG) [3]. As of May 2023, this corpus comprises 26,846 clinical EEG recordings collected between 2002 and 2017 at Temple University Hospital (TUH) in Philadelphia, Pennsylvania, USA. The dataset is divided into numerous subsets, allowing for targeted training on specific problems through sampled and labeled classes. However, it should be noted that the TUH EEG Corpus, which constitutes the majority of the TUH dataset, lacks EEG signal labeling, only including patient gender, age, and date of recording as potential prediction targets. These are not considered expert labels. The recordings were collected for clinical purposes and were not intended for machine learning at the time of recording. As the corpus is very large but lacks labels to identify diseases, TUEG presents an excellent opportunity for self-supervised pretraining, as SSL does not rely on labeled data.

The data in the TUH dataset was collected between 2002 and 2017, and as EEG recording technology has evolved over time, so has the data [68]. Different machines were used to collect the data, resulting in variations in data formats even within the corpus. For example, an examination of 500 arbitrarily selected recordings from the TUEG subset revealed eight different channel counts ranging from 29 to 36 and five different sample rates ranging from 250 to 1000 Hz. This demonstrates the necessity of preprocessing steps such as resampling and resizing for this dataset. Furthermore, the length of recordings exhibits significant variability, ranging from a few seconds to more than an hour. TUEG is comprised of a combination of recordings using the average (AR) and the linked ear (LE) references.

Specific subsets within the TUEG, such as the TUAB [69], have been manually labeled and sampled from TUEG to provide more accessible data for research on specific problems. The TUAB subset, in particular, is commonly used for EEG ML Tasks, as it is a balanced dataset with equal parts normal and abnormal EEG data. TUAB focuses on dataset quality by undersampling the much larger corpus, selecting files with high data quality. EEG records containing abnormalities are sampled disproportionately frequently in this data set, better balancing it for ML purposes.

2.7.2 Model input selection

Channel selection The vast majority of research projects that we have examined use the same channel selection for pretraining and fine-tuning, usually selecting a subset of channels common to all data [5], or a smaller subset believed to contain sufficient information [5]. Having the input to the self-supervised pretraining on the same format as the input to the downstream task is helpful, as it allows feeding the encoder the raw data in the same way both while pre-training and while fine-tuning. However, when working with data with different numbers of channels, keeping only common channels may lead to the elimination of large amounts of usable EEG data.

One notable exception to this convention is SeqCLR [6] by Mohsenvand *et al.*, where only single channels are encoded by the encoder during pretraining. This has the distinct advantage that every channel can be used for pre-training, letting the encoder learn from a larger body of data.

Window size There have been many different approaches to window size selection in the recent literature. The size of the classification windows is often closely related to the classification problem; Sleep staging often has larger window sizes than emotion recognition [6, 58]. For example, much of the previous literature on the SEED dataset [13, 70], an emotion recognition dataset, uses 1-second windows [13, 70, 71].

In the epilepsy domain, which we use for testing during the development of DECCaT-Net, the literature shows many different choices of window sizes. Mohsenvand *et al.* use 1 minute windows for the abnormal/normal problem, stating that most of the prior work considers the first minute of the recordings as the quality of signals drops with time [6]. Roy *et al.* demonstrated that using the first 11 minutes of the recordings can improve the classification results [72]. The Epileptic Seizure Recognition dataset [73] consists of 23.6 second long recordings, and as such this window size has also been used in recent literature [58].

Datasets for epilepsy or sleep staging, both clinical domains, commonly contain long recordings that often last a whole night, in the case of sleep staging. This allows for larger window sizes than those used in BCI or motor function datasets. This distinction is due to the time scale in which relevant events happen. For BCI, it is important that the desired machine input is read immediately after it is detectable in the brain. Simply put, we do not want to wait for input lag when we control the computer directly using our brain. Due to this, studies in the BCI domain often use a window length of 1-4 seconds [45], with models working on shorter windows being preferred. On the other hand, the aim of sleep staging is to detect long-lasting sleep stages from a whole night's worth of EEG data, thus if a sleep stage change only has a precision of ±30 s it is not critical for the task. Because of this, some sleep staging studies use a window size of 30 s [5, 6].

2.7.3 Preprocessing of EEG signals

The choice of preprocessing methods for the signals themselves is an important part of every time series prediction or classification task. Roy *et al.* observed that the most common preprocessing steps are downsampling, bandpass filtering, and windowing [2]; however, many different preprocessing schemes are used. For example, Hefron *et al.* note that a substantial amount of preprocessing was necessary for evaluating cognitive workload using deep learning techniques [74]. This involved trimming the EEG trials, downsampling the data to a lower frequency and number of channels, identifying and interpolating bad channels, calculating the average reference, removing line noise, and high-pass filtering the data at 1 Hz. In contrast, Stober *et al.* used only a single preprocessing step by discarding the bad channels of each recording[75].

These contrasting examples show a challenge in choosing preprocessing steps for a deep ML task on EEG data. Extensive research has been dedicated to preprocessing pipelines; however, to the best of our knowledge, there is a lack of sufficient research investigating the impact of complex preprocessing on the performance of modern deep learning (DL) architectures. Most of the models reviewed for this thesis incorporate some form of preprocessing to adapt the data to their respective models, including downsampling, trimming, and channel removal, among others [75]. Additionally, more intricate preprocessing techniques are commonly employed, such as rereferencing to a common average, low-pass and high-pass filtering, and band-pass filtering.

Some studies incorporate even more sophisticated preprocessing approaches, such as line noise removal [76], interpolation of faulty data points, and transformation to the frequency domain, often through the use of the fast Fourier transform (FFT) and other similar techniques. Due to the low signal-to-noise ratio (SNR) of EEG signals, noise removal presents challenges and may result in significant information loss; however, the potential increase in SNR justifies its consideration. However, a consensus has yet to be established on the optimal extent of preprocessing for modern DL architectures. Transforming data from the time domain to the frequency domain using the FFT algorithm can offer insights into the behavior of time series that are not readily apparent by examining the data solely in the time domain [77].

Signal filters The most commonly used frequency filters for EEG preprocessing are low-pass, high-pass, band-pass, and band-stop filters, all shown in Figure 2.8. In the case of preprocessing for ML purposes, the EEG signals are saved digitally, and as a result we use digital filters. Digital filters work by transforming the signal from the time domain to the frequency domain and filtering out certain frequency bands. Low-pass (Figure 2.8a) and high-pass (Figure 2.8b) filters remove frequencies that are, respectively, higher and lower than a specified threshold. Band-pass filters (Figure 2.8c) only keep frequencies between a lower and a higher threshold, while band-stop filters (Figure 2.8d) reject frequencies between a lower and

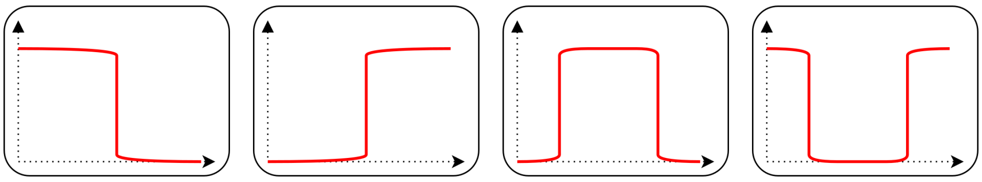

(a) Low-pass filter **(b)** High-pass filter **(c)** Band-pass filter **(d)** Band-stop filter

Figure 2.8: The most common filters used for EEG preprocessing, shown in the frequency domain by which frequencies they let pass.

a higher threshold.

In EEG, high-pass filters are used to correct baseline drift [78], low-pass filter are commonly used to remove random noise [79, 80], and bandpass filters are a way of combining the two [6]. Band-stop filters have seen use as augmentations for SSL [6, 59], or to filter out unwanted noise from *i.e.* ambient electrical voltage from alternating current, which has a specific frequency of usually 50 or 60 Hz, in the environment [81].

Downsampling Lowering the sampling rate, called downsampling, comes with upsides and downsides. The downsampled signal uses less space on disk and in memory, and leads to faster computation times. Faster computation times allow the model to be trained for more epochs, be deeper, or be run on a lower-end system.

Downsampling inherently leads to loss of information. However, not all the lost information is useful for the ML task, as EEG data have a notoriously low signal-to-noise ratio (SNR). The small differences between neighboring time points in EEG signals with high sampling rates are believed to be largely affected by signal noise, although how much remains an open question. Common practice shows that when downsampling to around 200 Hz [5, 6, 60, 82] or 100 Hz [8, 51, 59, 80] the performance gains compensate for the information loss, leading to the best classification results.

EEG rereferencing Each EEG channel represents the voltage difference between two electrodes, one of which is usually the common reference. Rereferencing is a fundamental preprocessing step aimed at reducing common artifacts and improving the interpretability of EEG signals. It involves transforming the recorded EEG signal by subtracting a reference signal to change the common reference. There are several different reference schemes that can influence the subsequent analysis and interpretation of EEG data [83]. Neurologists often use rereferencing to obtain a new view of the data when performing analysis [84].

References can be physical or digital. Physical references are often obtained by selecting an electrode location that is believed to have low EEG activity, such as the nose, earlobes, or neck ring. Digital references are computed from multiple EEG electrodes, such as the average reference (AR) or reference electrode standardization technique (REST) [85]. Rereferencing often changes the common reference from a physical to a digital reference.

The average reference, where the EEG signal is rereferenced by taking the average of all electrode channels as the reference, is widely used in the DL-EEG literature [38, 68]. This method aims to eliminate the contribution of common-mode signals across the scalp, but assumes that all electrodes contribute equally to the reference, which may not always be the case.

The REST can be used to obtain an "infinity reference", which aims to standardize the reference to a point on at infinity [85]. EEG reference comparison studies using classical EEG analysis methods have shown that REST performs better than AR, which again outperforms physical references.

As an EEG channel represents the voltage channel between a given electrode and the common reference, any two channels may be subtracted from each other to produce a new, valid channel. This technique can be used to produce new valid channels when more EEG channels are desired.

Mohsenvand *et al.* [6] proposed learning representations of channels one by one, followed by combining the appropriate number of single channel encoders when fine-tuning for downstream tasks. Proposing a framework not relying on the number of channels in a dataset enables the authors to benefit from a EEG-specific data augmentation technique based on rereferencing. Rereferencing enables us to obtain $N \times (N-1) + N = N^2$ new channels from an EEG signal that has N channels by creating all possible electrode pairs. Prior to learning spatial dependencies during fine-tuning on a downstream task, more data could lead to better representations of temporal dependencies during pre-training.

2.7.4 Artifact and noise removal

Sources of random noise and artifacts Noise in EEG can be divided into two categories, internal and external noise. Internal noise originates from brain activity from parts of the brain that are more distant from the electrode [83]. This noise has a lower amplitude than the signals originating from the brain just below the electrode, as the electrical voltage needs to propagate through more tissue to reach the electrode. The removal of internal noise is challenging, as it is present in all raw EEG data.

External noise can originate from the experiment environment or from the subjects themselves, such as muscular activity. Muscle activity, such as ocular movement, blinks, and other facial movements, generate electrical signals that have a much higher amplitude than brain activity, up to 100 µV for blinks [83]. This causes artifacts in the data, which can be detected due to the way they differ from brain activity and random noise, mainly by their amplitude. Small patient movements and sweating, which affects conductivity, are also a source of external noise. External noise from the environment can, for example, be from ambient electrical current in the room with from other electrical equipment in proximity of the EEG sensors, and line noise from the EEG equipment itself.

Related noise removal practices One of the main goals of preprocessing is removing noise. In this section, we divide noise into two categories, random noise and artifacts, as the approaches to combat them are different. Random noise, such as line noise and internal brain noise, is difficult to remove, but some research into this has been done [76]. Random noise often has a high frequency, and as such band-pass or low-pass filters are often used to reduce random noise [6, 79, 80].

Artifacts are often very different from normal data and can have a large negative impact if they are not detected and removed. Ocular artifacts, which arise from eye movements, are particularly prevalent. Extensive research has been conducted on the handling and removal of artifacts in EEG data [86], while certain studies opt to discard samples with values that surpass a predefined threshold indicative of an artifact [6, 38, 87]. Mohsenvand *et al.* are also able to drop bad channels during pretraining, as their model is not dependent on using the same number of channels from each EEG recording when pretraining their encoder [6].

Despite the common practice of artifact removal, a comprehensive review of 154 articles published in the field of DL-EEG led Roy *et al.* to assert that explicit artifact removal can potentially be circumvented when employing deep learning techniques, without adversely affecting task performance [6]. However, common practice in recent SOTA models shows that discarding some artifacts, such as spikes due to muscular movement or flats due to dead electrodes, is generally viewed as helpful [6, 8, 80].

2.7.5 Class distribution and imbalance handling

Class imbalance is a common issue in medical data, which poses challenges for classifiers striving to achieve optimal performance [88]. This imbalance arises due to the abnormal occurrence nature of diseases, making it difficult to gather sufficient data for rare events. Various strategies exist to address class imbalance, including oversampling, undersampling, augmentation, and imbalanced loss functions.

In SSL, a well-balanced class distribution facilitates the learning of a robust latent representation by the encoder. However, in the case of pretraining with unlabeled data, it becomes impractical to measure the class imbalance, as the class of each data point is unknown. Nevertheless, data set balance assumes particular importance in supervised learning, and when fine-tuning a pretrained model.

Over- and Undersampling Undersampling the majority classes is a viable approach to balance data sets, especially when a sufficient amount of data is available. Although this approach significantly reduces the size of the training dataset and often leads to a decrease in overall accuracy, it tends to improve the accuracy of minority class classifications [89]. This is particularly relevant in health-related data, such as EEG or ECG, where data indicating a disease or significant findings are typically in the minority compared to normal data. Correct identification of diseases is crucial and in the majority of cases, a false positive result is less harmful than a false negative.

Oversampling is another technique to handle class imbalance, involving the replication of data samples from the minority class, creating multiple identical copies. This increases the number of samples from the minority class and has been proven to be effective with classical machine learning methods [90]. However, with deep neural networks, oversampling can lead to overfitting and compromise the model's generalizability.

Chapter 3

Method

Introduction In this chapter, the implementation details of the proposed self-supervised model for EEG classification are presented. First, the general framework, system architecture, and the tools used to develop it are presented in section 3.1. Then, in sections 3.2 to 3.4, all aspects of the implementation are elaborated with detailed explanations and justifications. Finally, in section 3.5, Experimental cycle, the implementation details of our hyperparameter search and experiment setup are described.

3.1 Framework overview

We present Dual EEG Contrastive Convolution and Transformer Network (DECCaTNet), a model and contrastive learning framework for multichannel time series with a focus on transfer learning capabilities, able to effectively learn from large quantities of data. We have implemented and tested it on EEG data, but it is also applicable to multichannel time series from other domains.

Research questions

RQ1 *How does pre-training an n-channel encoder perform, and how is it best implemented for EEG data?*

RQ2 *What is the optimal number of channels in each group when using SSL with grouped channels on EEG-data?*

RQ3 *How does a transformer-based encoder perform in an SSL architecture for classifying multichannel EEG data?*

In order to answer our research questions, an SSL framework for pre-training DECCaTNet on EEG data using the tools presented in section 3.1 was implemented. A general overview of our framework is presented in Figure 3.1. The framework relies on three main components: data loading and preprocessing, pre-training, and fine-tuning, all of which are described in the following sections of this chapter.

First, all data sets are preprocessed with some dataset-specific and some general parameters. Then, all recordings used for pre-training are split into channel groups of n channels per group and saved as new recordings. Moving on, DECCaTNet is pre-trained on recordings with groups of n channels. The pre-trained encoder of DECCaTNet can then be saved and used for fine-tuning on a fine-tuning dataset that keeps its original number of channels. Finally, we obtain results on the downstream task.

Our framework follows that proposed by Mohsenvand *et al.*, which is again a modification of the SimCLR contrastive learning framework [56]. However, the unique properties of our proposed framework is splitting the recordings into groups of ($n \geq 1$) channels, which affects the implementation of other components. Similarly to all other parameters, the number of channels in each group, (n), can be simply altered in a configuration file, which means that the whole framework must dynamically adapt to the chosen n.

Justification of SSL In our project thesis [7] we found that SSL-based models consistently outperformed non-SSL models for EEG classification tasks [49, 54, 58, 91]. However, results from other domains suggest that SSL encoders are dependent on massive datasets to obtain good results. To establish SSL's value in EEG applications, it is crucial to demonstrate good performance compared to SOTA models and showcase transferability across datasets.

The surveyed literature confirms that SSL-based approaches achieve comparable or superior results on all surveyed datasets, even with limited annotated data. Furthermore, they outperform supervised models in cross-dataset performance [59, 62, 80, 82], highlighting the usefulness of SSL in the EEG domain. The impressive SOTA performance and transferability of SSL-based approaches warrant further exploration in EEG classification.

Tools

Python[1]	Python is a powerful and versatile programming language that is widely used for data analysis, machine learning, and scientific computing. Python is the backbone for our implementation, all other packages and tools listed are built upon Python.
Numpy[2]	Numerical Python library for scientific programming. We use Numpy to implement our own preprocessing functions and mathematical programming before sending the data to Pytorch Tensors.

[1]https://docs.python.org/3/
[2]https://numpy.org/doc/
[3]https://pytorch.org/docs/

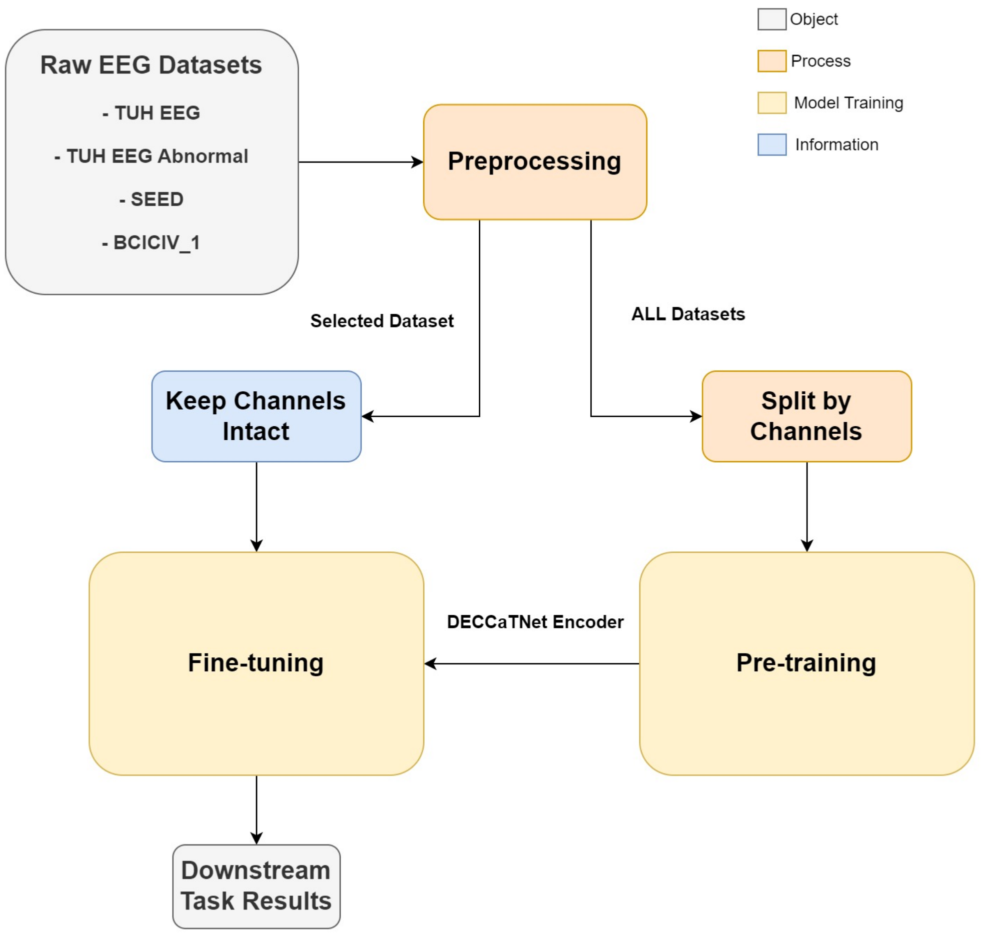


Figure 3.1: Overview of proposed framework

Pytorch[3] A flexible and efficient open-source machine learning library, with a dynamic computational graph to build and train neural networks. Our SSL-framework is built using PyTorch. Pytorch was chosen due to its easily customizable training loops and dataset classes, which we prefer when making an SSL model, as well as for its parallelization possibilities.

Mne[4] Package to explore and analyze neurophysiological data. It provides a comprehensive set of tools to process and visualize EEG, MEG, and other electrophysiological data. In order for our framework and preprocessing steps to function as desired, Mne has undergone some small changes presented in section 3.2.

Braindecode[5] Braindecode is an extension of Mne that enables users to work with deep learning models on EEG-data more efficiently. Including dataset fetchers, preprocessing, and visualization tools. For our use case, Braindecode has essentially been used as a wrapper for Mne-functions, loading and creating datasets. However, as our framework has some unique features, we ended up creating our own datasets and rewriting parts of the Braindecode library; see section 3.2.

YAML[6] YAML is a human-readable data serialization format that is used to store and exchange data between systems. It provides a simple syntax that is easy to read and write, making it a popular choice for configuration files and other structured data. We use it in order to create configuration files for running experiments and obtaining results. With YAML, our runs can be easily customized through text.

Ray[7] Ray is a fast and simple distributed computing framework for Python that makes it easy to scale and parallelize Python applications. Through Ray we utilized `Ray.Tune` which is a tuning framework allowing us to perform a hyperparameter search for all parameters in our YAML configuration files. It enables us to use parallelization when performing experiments such as the number of channels and, in general, speeding up the hyperparameter selection process. Will be further explained in experimental setup section 3.5

GitHub[8] GitHub is a widely used platform for Git version control and collaborative software development. We use GitHub as a code repos-

[4]https://mne.tools/stable/
[5]https://braindecode.org/
[6]https://yaml.org/spec/1.2.2/
[7]https://docs.ray.io/en/latest/
[8]https://docs.github.com/

itory in order to manage code and collaborate while designing the experiment. We also created forks of both Braindecode and MNE on GitHub to better keep track of our changes to the libraries.

3.2 Data

3.2.1 Datasets

To demonstrate transfer learning capabilities, we opted to sample datasets from different sources. To obtain many windows of training data from an easily manageable number of datasets, those with mostly long recordings and a high (more than 30) number of channels were chosen. As we want to compare the performance of pretraining with different channel group sizes $n > 1$, datasets with fewer than n channels could not be used. This excludes some popular sleep stage classification datasets from our list of options, such as SleepEDF [92], which uses only two channels. In the end, we ended up combining four different datasets, from three different sources, shown in Table 3.1 and Figure 3.2.

	Pre-training	Fine-tuning
TUH EEG Corpus (TUEG)	✓	
TUH Abnormal EEG Corpus (TUAB)	✓	✓
SJTU Emotion EEG Dataset (SEED)	✓	
BCI Competition IV (BCICIV) 1	✓	

Table 3.1: Datasets used for pre-training and fine-tuning

TUH EEG Corpus As described in section 2.7.1, TUEG is well suited for self-supervised pretraining. The data being unlabeled is not an issue. However, the disparate sample lengths and various EEG recording setups pose a challenge. TUEG has been sampled with size as a more important factor than data quality and contains a large number of recordings with bad channels or artifacts.

As the dataset is 1643 GB in size, the training cost of using the entire dataset would be too large. To demonstrate transfer learning capabilities, we do not wish TUEG to completely overshadow the other datasets, and only use a subset of 137 GB for our pretraining. From each recording, we drop non-EEG channels and keep the whole length of the recording.

TUH Abnormal EEG Corpus We use TUH Abnormal EEG Corpus (TUAB) [69] for both pre-training and fine-tuning. TUAB is a subset of TUEG that has been hand-picked to have high-quality recordings of at least 15 minutes using the average reference and only slight variations from the 10-20 international standard EEG montage. Signals are classified as normal or abnormal and sampled to create a balanced dataset. It has been divided into a training set and an evaluation set,

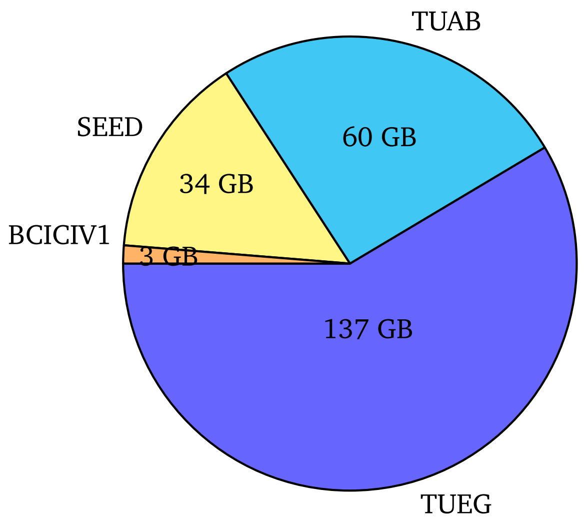


Figure 3.2: Size of the datasets used by DECCaTNet for pretraining. The combined size is 234 GB. 137 GB is only 8% of the whole TUEG, but this subset was chosen to not drown out the other datasets and to reduce training time.

the evaluation set being 10% of the size of the training set. The entire TUAB is 60 GB in size, and we use all EEG channels and the entire length of the recordings.

SJTU Emotion EEG Dataset SJTU Emotion EEG Dataset (SEED) [13, 70] is an emotion recognition dataset collected for emotion recognition experiments at Shanghai Jiao Tong University. EEGs were recorded while the subjects watched film clips chosen to elicit specific emotions. The data is labeled by what emotion the video was supposed to elicit. However, since we chose to use the dataset for only pre-training, we do not use the labels. The dataset comes with preprocessed segmented files in addition to raw data. For our experiment we chose to use the raw data as the samples are longer and it lets us use the same general preprocessing pipeline we have designed for all datasets. The raw files we use in our experiment are in total 34 GB in size, and again we select all EEG channels and the entire length of the recordings.

BCI Competition IV Dataset 1 This dataset was one of the datasets used for BCI Competition IV (BCICIV) which was held in 2008. The dataset is created for motor imagery experiments and was recorded by the Berlin BCI group [93]. We have chosen this dataset because it is a bci dataset consisting of relatively long continuous recordings, from which we can extract sufficiently sized windows for pre-training. However, it being a BCI dataset, where recordings tend to be much shorter than in clinical datasets, it is only 3.3 GB in size. We use all EEG channels and the entire length of each recording.

3.2.2 Loading and preprocessing

Dataset loading A challenge we face when transfer learning is that different datasets have different EEG recording environments, and for optimal results, each dataset could need its own custom-tailored preprocessing pipeline. However, we want to demonstrate large-scale transfer learning capability, so dataset-specific preprocessing steps were kept to a minimum. For the TUH datasets, Braindecode comes with premade loader functions, allowing us to get a Braindecode `BaseConcatDataset` containing MNE `Raw` objects from each EEG recording file.

Unfortunately, the Braindecode loader functions had not been updated for the updates to TUEG v2.0.0 and TUAB v3.0.0, released in December 2023. In these updates, the directory hierarchy and file naming scheme changed significantly, breaking the old loading functions. This change was what initially led us to create our own fork of Braindecode to add our own changes to the library. We submitted our changes to the Braindecode developer community, who in turn used them when updating the official library.

For the SJTU Emotion EEG Dataset (SEED) and BCICIV Dataset 1 (BCICIV1) datasets, we created our own loader functions in the style of Braindecode's TUH dataset loader functions. In these functions, we loaded the data files to `Raw` MNE objects, manually set the correct channel names and original frequency, and discarded non-EEG channels. As BCICIV1 is stored using datatype `int16` the data needed to be converted to floats.

When working with datasets this large, it may be impossible to keep the data loaded in memory. Instead, the process needs to be serialized by storing it on disk, and the recordings are only loaded into memory when they need to be processed. Braindecode implements serialization in their loading, saving, and preprocessing functions, though while using them we ran into multiple bugs and challenges.

Parameter	Value
Window length	30 s
Peak-to-peak high threshold	8000 µV
Peak-to-peak low threshold	1 µV
Sample rate	200 Hz
Bandpass low	0.3 Hz
Bandpass high	80 Hz

Table 3.2: Preprocessing parameters

Windowing Using the `BaseConcatDatasets` obtained from the previous dataset loading step, we apply the exact same preprocessing pipeline, outlined in Figure 3.3, to each of the datasets from this point forward. All selected values of the numerical parameters are shown in Table 3.2.

First, all data are rescaled to microvolts (µV), and non-EEG channels that have manually been selected are dropped. For the fine-tuning dataset we only keep

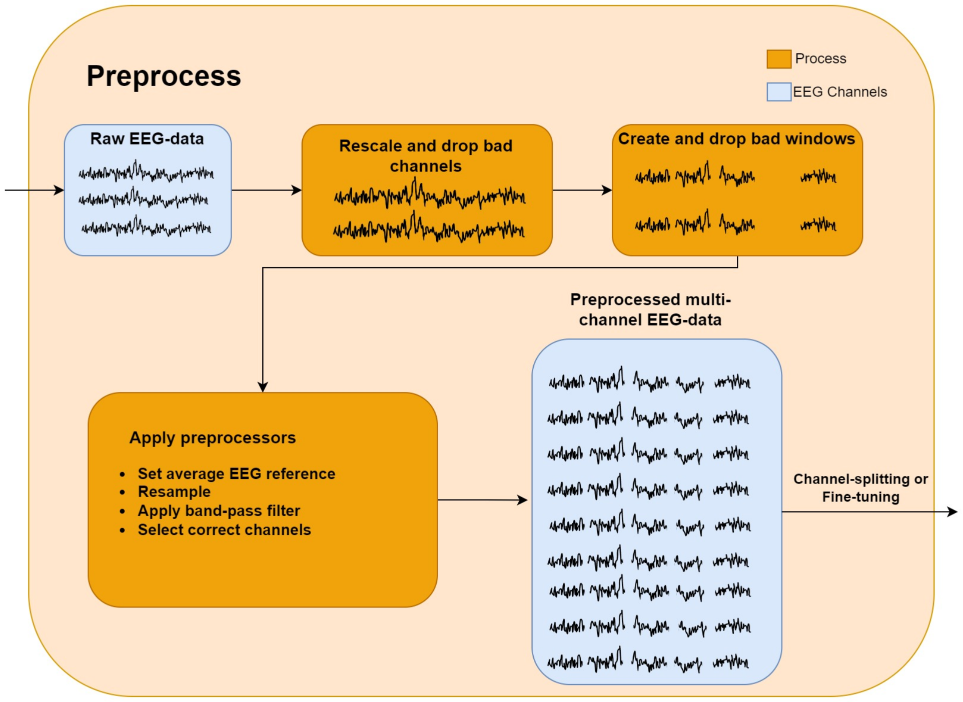

Preprocess
Process
EEG Channels
Raw EEG-data
Rescale and drop bad channels
Create and drop bad windows
Preprocessed multi-channel EEG-data
Apply preprocessors
Set average EEG reference
Resample
Apply band-pass filter
Select correct channels
Channel-splitting or Fine-tuning

Figure 3.3: Overview of our preprocessing pipeline

common channels for all recordings and ensure the channels are saved in the same order for all recordings. Then we split all recordings in each dataset into 60 s long windows using Braindecode's `create_fixed_length_windows()` function. To speed up the rest of the preprocessing pipeline, we wish to downsample the data to 200 Hz as early as possible, but when windowing MNE objects that have been downsampled, signal quality decreases, as downsampling effectively jitters trigger timings[9].

The `create_fixed_length_windows()` function does not save a new file for each window, but creates an MNE `Epochs` object that stores window start and stop time points for each recording. The function also checks each window, marking windows that pass certain thresholds as "bad". Bad windows are not returned when iterating through the object and are thus effectively dropped from the dataset. The function checks for bad windows by calculating the maximum peak-to-peak distance in each window, and marks it as bad if it is above the upper limit of 8000 µV, indicating an ocular or other muscular artifact, or below the lower limit of 1 µV, indicating a dead channel. The function is implemented with CPU parallelization, running on multiple cores.

[9]mne.io.Raw — MNE 1.4.0 documentation, resample; `https://mne.tools/stable/generated/mne.io.Raw.html#mne.io.Raw.resample`

Windowed data issues with large datasets in Braindecode After the dataset is windowed, it is sent through the rest of the signal preprocessing pipeline. As previously mentioned, when preprocessing datasets this large, the data need to be serialized, as all data does not fit in memory at the same time. We use Braindecode `BaseConcatDataset` object that can keep pointers to the actual data files, while only recording description data is kept loaded in memory.

When loading windowed files without preloading the data, which is not an option for large datasets, Braindecode `WindowsDataset` helper objects keep an open file pointer to a JSON file with the data description for each recording. For our large dataset, we ran into multiple issues due to this design choice in Braindecode. First, we ran into the open file limit of the operating system when loading the entire windowed dataset due to the open description files. We solved this problem by loading one `BaseConcatDataset` at a time from each batch of 500 files for further preprocessing. Keeping more `Epochs` helper objects open at a time caused the `OSError` from having too many open files.

Another issue caused by the open file identifiers was that operations on the `Epochs` objects were no longer serializable, as the open file identifiers cannot be saved to disk. This stopped us from parallelizing the rest of the preprocessing pipeline, as objects could not be sent between threads. We chose not to tackle the open file identifier issue because it would require extensive low-level changes to the Braindecode package, we found suitable workarounds, and the runtime without parallelization was still feasible.

Further signal preprocessing steps The rest of our preprocessing pipeline is heavily inspired by the preprocessing steps used by Mohsenvand *et al.* [6] in SeqCLR, as it is the CL architecture that has inspired DECCaTNet the most. We did not have the resources to empirically test preprocessing parameters and have therefore tried to match SeqCLR to have good grounds for comparison. We needed to implement our own pipeline, but the numerical values of the preprocessing parameters were chosen to mirror those used for SeqCLR.

First, we resampled all the data to 200Hz. This was done using the MNE resample method which uses an adapted verion of `scipy.signal.resampls`'s resampling technique[10]. We chose to resample all signals to 200 Hz, as this is the frequency used for SeqCLR. The frequency content of the EEG signals lies mainly in the 0-40 Hz band, so a minimum sampling rate of 100 Hz is recommended [94]. The maximal frequency allowed in a signal given a sampling rate, called the Nyquist frequency, is half the sampling rate. So, for 200 Hz, the Nyquist frequency is 100 Hz, which should allow us to pick up abnormalities in the EEG data with frequencies up to 100 Hz.

We then rereference all recordings to use the average reference. Of the possible EEG referencing techniques, we chose to use the average reference, as it is already used by TUAB and parts of TUEG, and can be calculated for the other datasets

[10]mne-python/filter.py at maint/1.4 · mne-tools/mne-python · GitHub
`https://github.com/mne-tools/mne-python/blob/maint/1.4/mne/filter.py#L1903`

regardless of which reference they originally used. The rereferencing function we used is implemented in MNE.

Finally, we apply a bandpass filter to the signals, passing the 0.3 to 80 Hz band. These frequencies are the same thresholds used by Mohsenvand *et al.* for SeqCLR. The high-pass limit of 0.3 Hz helps mitigate baseline drift, while the low-pass limit of 80 Hz helps filter out some high-frequency noise.

After all the signal preprocessing operations are applied, a copy of the dataset is saved. The fine-tuning dataset does not need any more fine-tuning after this step, while the pre-training dataset still needs to be split by channel groups.

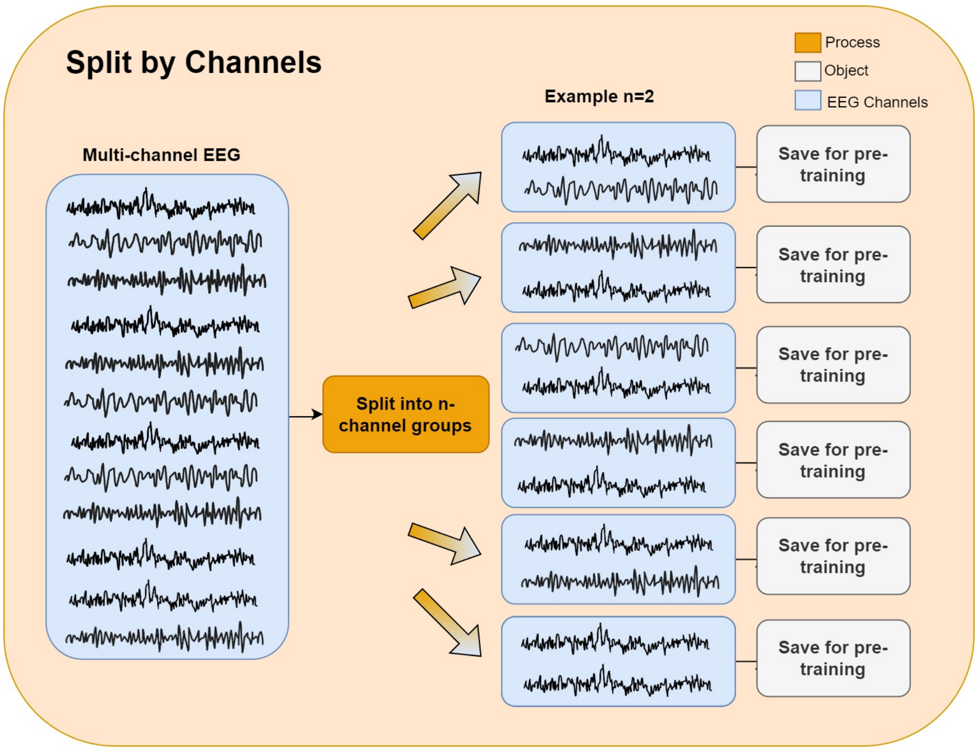


Figure 3.4: Overview of splitting EEG recordings into channel groups

Channel-grouping Our encoder in DECCaTNet takes groups of channels of size n, so the final step of our preprocessing pipeline is to split each recording into separate recordings according to their channel groups. The process is illustrated in Figure 3.4 for $n = 2$. We implemented multiple different channel grouping functions, each transforming a list of channels into several lists of channels of length n. Each of the channel splitting functions is explained in Table 3.3. Due to time and resource constraints, we were only able to perform large-scale tests using `make_adjacent_groups`, which produces the smallest number of groups and which we will discuss in more detail in this section.

The `make_adjacent_groups` function (Algorithm 1) takes a list of channel

Function name	**Number of groups**	**Description**
`make_adjacent_ groups`	$\lceil \frac{N}{n} \rceil$	Makes the fewest possible groups of sequential channels from the list of channels, while including each channel at least once.
`make_overlapping_ adjacent_groups`	$N-n+1$	Makes groups of n sequential channels starting from each channel in the list of channels
`make_all_ combinations`	$C(N,n)$	Makes all possible combinations of n groups.
`make_all_` textttpermutations	$P(N,n)$	Makes all possible permutations of n groups. Different orderings of the same n channels are all included.

Table 3.3: Implemented channel grouping functions. N is number of channels in the EEG recording, n is the number of channels per group, $C()$ is the number of combinations function, and $P()$ is the number of permutations function.

names and group size as input and makes the smallest possible number of groups. If the total number of channels N is not divisible by group size n, there will be some overlap between groups. We ensure this overlap is minimal and evenly spread by calculating the minimal possible number of groups while including all channels $\lceil \frac{N}{n} \rceil$, and having group starts evenly distributed between the first and last possible group index. See Algorithm 1 for the pseudocode of the function.

As each EEG recording can have a different list of channels, Algorithm 1 needs to be used for each recording. After creating the channel groups, a copy of the recording is saved to a new file for each group with only the given channels included. These copies are the final output of the preprocessing pipeline and what we use for pre-training the model.

3.3 Pre-training

Pre-training framework Our pre-training implementation is inspired by several papers using SSL on EEG data [6, 54, 55], but, simply put, is based on the contrastive learning framework SimCLR [56]. An overview of the data flow in the implementation is presented in Figure 3.5.

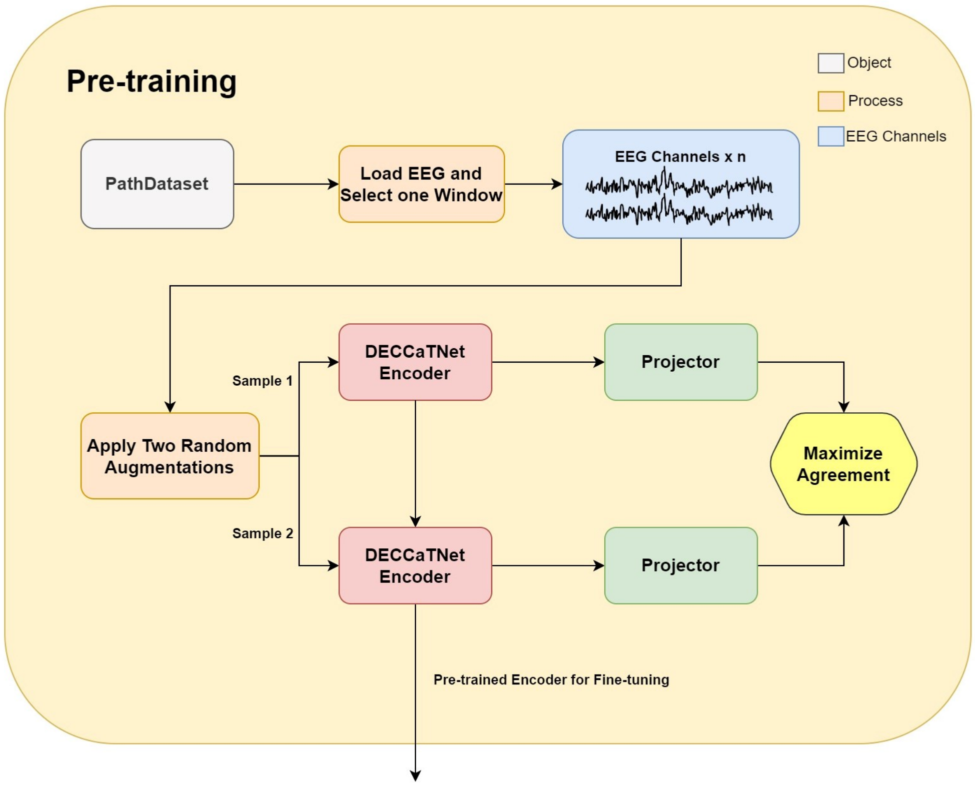


Figure 3.5: Pre-training overview, first part is loading data and the second part is applying contrastive learning to EEG data.

Algorithm 1 make_adjacent_groups(C, s_g)

```
Input: List<str> $C$ , int $s_g$                    ▷ $C$: Channel names, $s_g$: Group size
Output: List<List<str>> $G_c$                       ▷ $g_c$: Channel groups
 1: $N \leftarrow length(C)$
 2: $n_g \leftarrow \lceil \frac{N}{n} \rceil$                           ▷ $n_g$: Number of groups
 3: List $S \leftarrow$ arrange$(0, N - n, n_g)$           ▷ $S$: First index of each group
 4: List $G \leftarrow$ new empty list                         ▷ $G$: Groups
 5: for number $s_g \in S$ do
 6:     $s_g \leftarrow$ round$(s_g)$
 7:     List $g \leftarrow C[S : S + s_g]$                        ▷ Slice of list $C$
 8:     Append $g$ to $G_c$
 9: end for
10: return $G_c$
```

As we faced several challenges when it comes to data preprocessing, which are detailed in subsection 3.2.2, we have created our own custom dataset class called `PathDataset` (subsection 3.3.3), which extends Pytorch's Dataset class [95]. After loading, two randomly selected augmentations are applied to create a positive pair. Contrastive learning is then used to force representations of similar pairs closer and those of negative pairs further apart in the latent space (maximizing and minimizing agreement between representations). The pre-trained model is then saved, and DECCaTNet can be loaded to be used in fine-tuning. In addition, DECCaTNet and the projector can be loaded to continue pre-training at a later stage.

3.3.1 Pytorch datasets and loaders

Pytorch provides two essential data primitives: `torch.utils.data.DataLoader` and `torch.utils.data.Dataset`. These primitives allow researchers to have dataset code that is independent of the model training code, enabling readability and modularity. Additionally, Braindecode [52] offers specialized datasets for some of the most popular EEG datasets, which are extensions of Pytorch datasets. Extending Pytorch datasets enables the use of its data loader, enabling seamless access to data for training loops.

Previous implementations working with DL on EEG-data have utilized Braindecode, Pytorch, or similar libraries for data handling during training [5, 45, 54, 58, 96, 97]. However, since we are the first to split each recording into separate channel groups, more customization was needed.

3.3.2 Initial Dataset

As preprocessing is a tedious process that takes a long time, we wish to preprocess only once and load a preprocessed dataset each time we perform an experiment.

As described in section 3.2, several measures had to be considered during pre-processing. However, for loading datasets for pre-training our initial thought was to utilize the proposed method from Braindecode [98]. With Braindeocde, the pointers and metadata of the dataset are kept in memory, but EEG-signals are only loaded when necessary, *e.g.* for training.

Due to the Braindecode windowed dataset bug, described in subsection 3.2.2, the entire dataset could not be kept in memory simultaneously, even with the actual EEG data not being loaded. Initializing the dataset using Braindecode's built-in functions became unfeasible, requiring either splitting the dataset into parts, which we would have to keep track of during training, or writing a custom dataset class.

During pre-training, we only need the data itself, and not target variables or other features. As a result, in addition to solving the aforementioned problem, a custom dataset class can save resources by only loading the data itself.

3.3.3 PathDataset

An aim of the custom dataset class, called `PathDataset`, is that it should be computationally cheap to initialize and that the pointers to the windows take up minimal memory. Furthermore, the loading of data from the class as training progresses should not slow down the training loop. First, the PyTorch dataset class is extended to ensure compatibility with other PyTorch support functions.

As all groups of n channels are saved as separate files, the `PathDataset` needs to contain pointers to each window in each file, keeping a list of the location of each window. Each pointer is in the format $(\text{path}_i, \text{window}_j)$, where i is the offset of the saved recording (see Figure 3.6) and j is the index of the window in the recording. Other than the pointers, no additional metadata about the files is needed for training.

Attributes and methods

A custom dataset class must implement multiple methods, amongst them `__init__` and `__getitem__`. Initialization, done in the method `__init__`, along with the dataset file structure, are described first, before describing `__getitem__`.

File structure A general overview of the file structure saved after grouping channels and loaded by `PathDataset` is presented in Figure 3.6. For each channel group size n, a separate folder is created. Then for each value of n, each original recording is assigned an ID offset from increasing multiples of 100 $(0, 100, 200, \cdots)$. Then, as the recordings are split, each new channel group is saved in ascending order from the ID offset. Each subfolder will contain a `.fif` file, which is the EEG-signal and descriptive `.json` files saved by Braindecode, but not used by the model during pre-training.

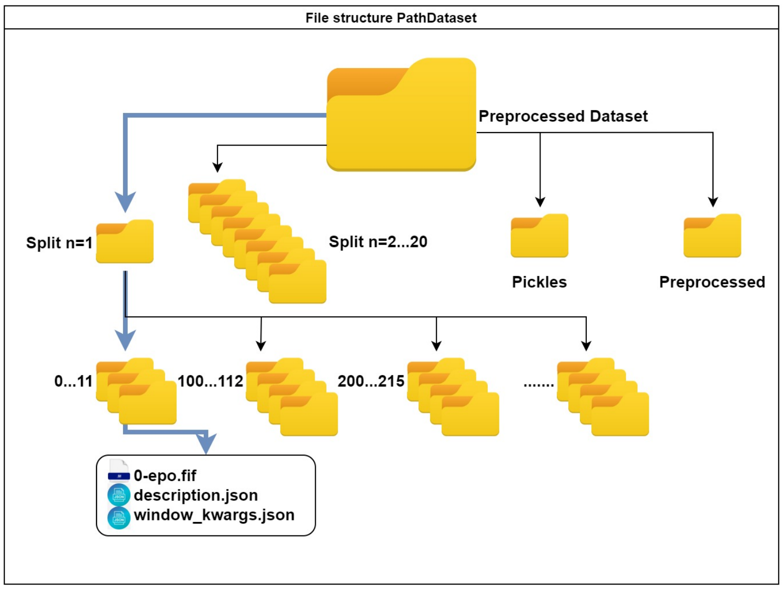


Figure 3.6: File structure after prerpocessing

`self.ids` To initialize the PathDataset for a given n, the correct pickled list of tuples $(\text{path}_i, \text{window}_j)$ from the Pickles folder and set the root path to the corresponding split folder. The list of pointer tuples is saved as an attribute called `self.ids`. All `.fif` file paths are derivable from the list `self.ids`.

`__getitem__` In our self-supervised framework, there is no need for labels or metadata, just the subchannel signals themselves. We capitalize on this by ignoring Braindecode metadata and only loading the signal we are looking for with MNE [99] in our `__getitem__` function. An overview of the function is presented in Algorithm 2. For each call to `__getitem__`, one signal is loaded and then the correct window is returned as x.

The method then applies augmentations, described later in subsection 3.3.4, and finally a positive pair (x_1, x_2) is returned.

Algorithm 2 PathDataset __getitem__(idx)

```
Input: $idx$
Output: $(x_1, x_2)$ positive pair
1: $path_i, window_j \leftarrow self.ids[idx]$                                ▷ subsection 3.3.3
2: $file\_path \leftarrow os.path.join(self.root\_path, path_i, path_i + "-epo.fif")$
3: $window\_dataset \leftarrow mne.load\_signals(file\_path)$
4: $x \leftarrow window\_dataset.get\_data(item = window_j)$
5: $x_1, x_2 \leftarrow apply\_augmentation(x)$                                 ▷ Algorithm 3
6: return $x_1, x_2$
```

ConcatPathDataset During the pre-training loop, several datasets are used at once. To enable this, we extend PyTorch's ConcatDataset [95]. ConcatPathDataset simply contains a list of all PathDatasets used for pre-training. We had to extend because we needed a custom split function in order to create test, validate and train splits from the PathDatasets. Iterating over and feeding data to the training loop is done via a PyTorch DataLoader [95].

Data loading cost We were afraid that our dataset implementation would slow down training as several signals are loaded for each batch. Therefore, we timed each operation in our training loop. The results are presented in Table 3.4. The timings were measured with $batch\_size = 128$ on two NVIDIA Tesla V100 PCIe 32 GB GPUs.

Operation	**Average Time (ms)**
Load and augment data	0.0595
Encode with DECCaTNet	0.1605
Calculate contrastive loss	0.1093
Calculate delta loss	0.0072
Perform backward propagation	0.4172
Clear CUDA cache	0.5803
Total time	1.3609

Table 3.4: Average time used on pre-training operations

From Table 3.4 it appears that the time used for loading and augmenting data is sufficiently small compared to the average total time for each epoch. Therefore, we conclude that our PathDataset meets our requirements of easy initialization and no noticeable time usage.

3.3.4 Augmentation

As stated in section 2.6, Previous work, Augmentations, each dataset, architecture, and task is unique and would optimally require different augmentation approaches. However, since the selection of augmentations for EEG-data is still an open problem [100], we chose to follow the augmentation work of Mohsenvand *et al.* [6]. The main reason behind this is that they conducted a comprehensive ablation study with the help of neurologists, to determine the best augmentations. Furthermore, their research resembles ours when it comes to datasets, architecture, and approach.

Random parameters We decided to randomize the parameters each time an augmentation is called to make the contrastive pretext task harder for our model. By doing this, we ensure that our model does not learn the different parameters but is faced with a new cross-view prediction task each time [6, 91]. Following Mohsenvand *et al.* [6], we define a transformation range for each augmentation with sensible min/max values. As there has been success with both fixed augmentation parameters [58] and randomized selection [91], we chose to randomize in order to have as much variability in our pretext task as possible.

Augmentation method Our augmentation method is presented in Algorithm 3. First, two augmentations are randomly selected from a set of predefined augmentations [6]. It is important to note that the selected augmentations are always different [54]. Then, the parameters for each augmentation are randomly selected before the augmentations are applied to the input sample, creating a positive pair. Since combining different augmentations was not performed by Mohsenvand *et al.* [6], we decided to perform an experiment on whether adding jitter would increase our performance. Therefore, the jitter can be configured from the configuration file.

Algorithm 3 apply_augmentation(x)

```
Input: $x$
Output: $(x_1, x_2)$ positive pair
1: $aug_1, aug_2 \leftarrow$ two random augmentations
2: $aug_{params} \leftarrow$ randomized parameters
3: $x_1, x_2 \leftarrow aug_1(x, aug_{params}), aug_2(x, aug_{params})$
4: if AddJitter then
5:     $x_1, x_2 \leftarrow$Jitter$(x_1, x_2)$
6: end if
7: return $x_1, x_2$
```

Chosen augmentations Our chosen augmentations and transformation ranges are listed in Table 3.5. All augmentations except permutation are obtained from Mohsenvand *et al.* [6]. The domain-specific augmentations proposed by neurologists are chosen because we argue that if specialists are able to classify an augmented signal, the semantic information relevant for classification is kept. Furthermore, further exploration shows that similar augmentations are applied in other research, without mentioning that these are domain-guided [54, 59, 67, 91]. In addition, most transformation ranges are derived from Mohsenvand *et al.* [6].

However, as our window size is larger, we have adjusted transformation ranges for augmentations that alter the time domain, such as time shift and masking, accordingly. The transformation range for the permutation augmentation is derived from Eldele *et al.* [58] and Jiang *et al.* [54].

[11] Gaussian Noise https://en.wikipedia.org/wiki/Gaussian_noise

Augmentation	**Parameter description**	**min**	**max**	**Visualization**
Permutation	Number of permutations	5	10	Figure 2.7c
Masking	Masked timepoints in a row	150	500	Figure 2.4b
Band-stop filter	5 Hz width, Hz filter	20	82.5	Figure 2.8d
Additive noise (jitter)	Standard deviation GS[11]	0	0.2	Figure 2.5c
DC-shift	Shift in µV	-10	10	Figure 2.5b
Amplitude scale	Scaling factor	0.5	2	Figure 2.4c
Time-shift	Size of shift in timepoints	-50	50	Figure 2.4a
Additional noise	Standard deviation for GS[11]	0	0.05	Figure 2.5c

Table 3.5: Chosen augmentations

3.3.5 Pretext-task

The need for domain-specific pretext tasks is reduced by adapting domain-guided augmentations. This is indicated by Mohsenvand *et al.* [6] outperforming Wagh *et al.* [51], who take advantage of domain-guided pretext tasks. The current SOTA on the abnormal / normal classification on TUAB also uses a CL-based approach [54]. However, as their supervised counterpart performs well without pre-training, one could argue that their architectural approach is as important as their pretext task. However, in our project thesis we observed two things, generally speaking. Most papers based on CL report increased performance compared to an unsupervised counterpart, and CL methods [6, 54, 58] often outperformed predictive methods [8, 51, 82, 101]. This indicates that CL is a valuable pretext task for the EEG domain.

Contrastive Pretext Task The proposed contrastive learning algorithm is presented in Algorithm 4. Our contrastive leraning approach is comparable to that of SeqCLR, and other models [6, 54, 58]. Using the custom dataset class `PathDataset`, our augmentations are applied by the `__get_item__()` function of the dataset.We also utilize the temperature-scaled contrastive loss function presented in Equation 2.3, Previous work, togheter with a cosine similarity metric (Equation 2.2).

The lower part of Figure 3.5 provides a simplified visualization of the contrastive learning mechanism, showing it maximimzing the agreement between the encodings of positive pairs. Algortihm 4 provides a more detailed explanation of the CL algortihm. First, all data are divided into batches of size BS. Then, for one batch at a time, we obtain an encoding of each positive pair in the batch. After obtaining all encodings, the cosine similarity between all samples in the batch is calculated by the Equation 2.2. The cosine similarities are used to calculate the final contrastive loss by normalizing the loss of each positive pair. We only have to calculate the loss for each positive pair, as each loss is calculated with regard to all samples in the batch, resulting in the inclusion of all negative pairs for each loss calculation. This rewards maximizing the agreement of positive pairs while mini-

Algorithm 4 Contrastive learning algorithm

```
Input: ConcatPathDataset $X$, Batch size $BS$, Encoder $E$, and Contrastive loss
    $\mathcal{L}(i, j)$, see Equation 2.3
Output: Pre-trained model
 1: for batch in $X$ do
 2:     for sample $x_{i_1}, x_{i_2}$ in batch, $i \in \{1, 2, ..., BS\}$ do          ▷ Positive pair
 3:         $e_i, e_{i+BS} \leftarrow E(x_{i1}), E(x_{i2})$     ▷ enocde positive pairs (subsection 3.3.6)
 4:     end for
 5:     for $i, j \in \{1, 2, ..., 2BS\}$ do
 6:         Calculate $Cosine(e_i, e_j)$ using Equation 2.2
 7:     end for
 8:     Final loss $\mathcal{L} \leftarrow \frac{1}{BS}\sum_{i=1}^{BS} \mathcal{L}(i, i+BS)$ ▷ Contrastive loss for all positive pairs
 9:     Update $f$ weights with $\mathcal{L}$
10: end for
11: return $f$
```

mizing it for negative pairs. Finally, the weights of the encoder and projector are updated according to the final loss by backpropagation. After several epochs, the final DECCaTNet encoder can be saved and used for fine-tuning on a downstream task.

CL Parameters There are several parameters associated with CL. The relevant parameters for this thesis are presented in Table 3.6. In the **Value range** column, the values given as a range in parentheses are those explored during the hyperparameter tuning in section 3.5. The ranges were established with inspiration from related literature [6, 54, 58].

Parameter	Description	Value range
Learning rate	Step size for weight updates. Influences the convergence speed and stability.	(1e-4, 1e-1)
Temperature	Controls the sharpness of the probability distribution generated by contrastive loss. Scales the similiarity scores, affecting the smoothnes between positive and negative pairs.	(0.01, 0.2)
Weight decay	Increase regularization strength and generalization performance. Adds additional term to the loss function which penalizes large weights.	(1e-5, 1e-2)
Epochs	How many training iterations to be performed before DECCaTNet is fine-tuned.	(10, 200)
Batch size	Number of samples per batch. Affects stability of gradients and computational effiency. Often limited by computational resources	(64, 1024)
Train split	Percentage of data used for training. During pre-training all data is used in order to train. Set value to constant 1	1
Shuffle	Wether the order of which samples are loaded should be random or not. Ensures that the model doesnt learn any patterns or biases present in the order of samples. Set to True as the model should be encoruaged to learn semantics of several datasets at once, instead of one and one.	True

Table 3.6: Parameters for CL-algorithm

3.3.6 DECCaTNet encoder

DECCaTNet is short for Dual EEG Contrastive Convolutional and Transformer Network, which refers to the entire network from the preprocessing to the downstream task. In this section, we will focus on the encoder part of our DECCaTNet network. The encoder is the backbone of the network, as this is the part that learns how to encode EEG signals. When contrastive loss is calculated by Algorithm 4, the encoder weights are updated.

During both pre-training and fine-tuning, the encoder functions as a feature extractor. While pre-training, the goal is to enhance the semantic EEG features learned by the encoder. On the other hand, while fine-tuning, the obtained features are used as features for a neural network fitted for the donwstream task. The general idea of the encoder is presented in Figure 3.7. The encoder takes an

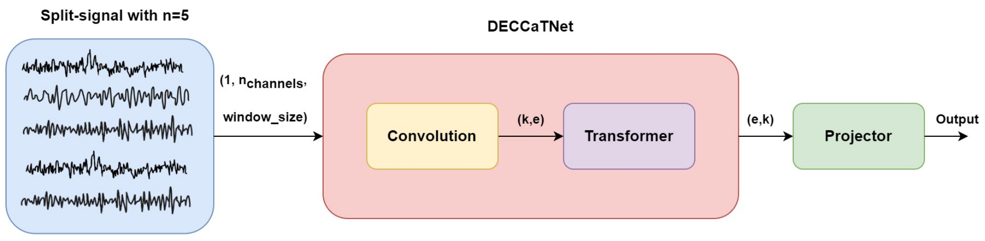


Figure 3.7: Overview of DECCaTNet encoder, tensor shapes in parentheses

input with dimensions $(BS, n, window\_size)$, where BS is batch size, n is number of channels per group, and $window\_size$ is the number of samples per window. The encoder output has dimensions (e, k), where e is the fixed embedding size, and k is a variable linearly related to $window\_size$.

The encoding produced by the encoder is finally fed to the projector for calculating contrastive loss during pre-training, or to the classifier for downstream classification during fine-tuning.

Combining CNN and transformers The rise of transformer-based models in the machine learning community is documented in section 2.3, which includes success in the time series domain [5, 9, 41, 42, 45].Several transformer-based models applied to the time series domain are implemented as a combination of CNN and transformers, and outperform their pure transformer counterparts [9, 42].

The Conformer [45] captures spatial and temporal dependencies in its CNN module, before feeding the learned embeddinds to a transformer. BENDR, a self-supervised hybrid model, reports similar results but states that its architecture could benefit from an adjustment to better handle EEG-data [5], showing that there is a need to further explore self-supervised hybrid models for EEG-classification.

Taking into account the supervised results obtained, its EEG adjusted design, and the possibilities for handling variable input lengths, we have chosen to follow the work of Song *et al.* [45] and implement an encoder inspired by the EEG Conformer architecture. All parameters and design choices of the convolution module follow those of the Conformer.

Architectual overview

A detailed implementation of our DECCaTNet encoder can be seen in Figure 3.8. As previously mentioned in subsection 3.3.6, the module following the encoder is decided by the task performed. It is either a projector during pre-training, or a classifier during fine-tuning. The convolution block follows the conformer of Song

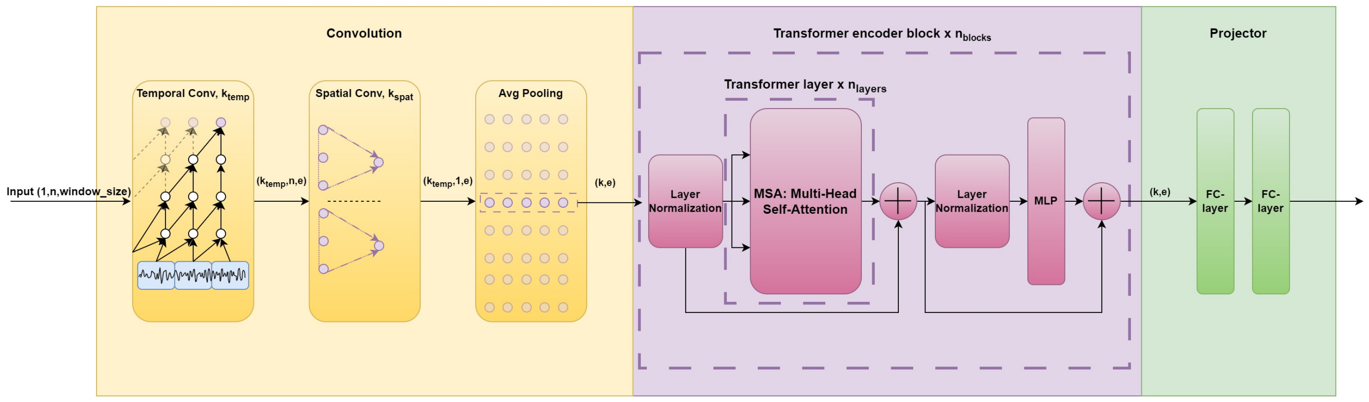


Figure 3.8: Detailed overview of DECCaTNet encoder, consiting of a convolution-module, a transformer-encoder and a projector module. The projector is replaced by a classifier during fine-tuning and classification.

et al. [45], and the transformer encoder (also known as the self-attention module) is inspired by the work of Vaswani *et al.* [4].

Temporal and spatial convolution We perform a two-dimensional convoliution operation when extracting both temporal and spatial features from the input signal. Following Song *et al.*, we separate the two-dimensional operation into two one-dimensional convolution layers. First, a convolution with $k_{filters}$ kernels of size $(1,25)$ and a stride of $(1,1)$ is applied. By selecting the mentioned kernel size and stride, the convolution is applied on the time dimension, capturing temporal dependencies.$k_{filters}$ is set to 40, mirroring Conformer.

The second layer has $k_{filters}$ kernels of size $(n,1)$ with stride $(1,1)$, enabling it to learn representations of spatial dependencies between different channels. n equals the number of channels per group in the pre-training dataset. Resulting in spatial convolution learning the dependencies between the channels in a subgroup of channels. A higher n would allow the encoder to learn more spatial dependencies, while the fine-tuning classifier would need to learn less, and vice versa for a lower n.

Average pooling and tokenization After spatial convolution, we apply batch normalization, a dropout layer, and an activation function (exponential linear unit (ELU)) for nonlinearity following [45], and increase the generalization of our learned representations. Then we apply an average pooling layer with kernel size $(1,75)$ and stride $(1,15)$. This layer has two purposes; smooth out the learned representations to avoid overfitting and decrease the size of the representations, reducing the computational complexity. Finally, to mimic the behavior of embeddings and create tokenizations for the transformer encoder, we squeeze the dimension to the correct embedding size and transpose (also referred to as rearange) the representation space so that each temporal point can be treated as a token by the transformer.

Transformer encoder The transformer encoder follows the standard implementation of Vaswani *et al.* As seen in Figure 3.8, we only use the self-attention and feedforward modules, which are referred to as one transformer encoder block. We then use n_{blocks} encoder blocks sequentially to obtain our encodings. The inner workings of the transfomrer are described in subsection 2.3.1 and will not be elaborated here. The implementation is done using PyTorch's transformer encoder layer and the transformer encoder [95].

As Song *et al.* [45] did not perform self-supervised learning, we do not wish to follow their choices of transformer encoder parameters, as it may not be relevant for our use case. However, since BENDR by Kostas *et al.* [5] also follows the implementation of Vaswani *et al.* [4] and performs self-supervised learning with a hybrid architecture, its parameters are more relevant to DECCaTNet. BENDR uses eight transformer encoder layers and has eight repeated transformer encoders in their work. We choose to follow these parameters to save computational costs and time.

Embedding size After a signal is sent through the encoder, we obtain an embedding of shape (k, e), where e is the embedding size chosen. The encoder output is flattened to a fully connected neural layer, and the final embedding size becomes $k \times e$. Because different window lengths should preferably be supported (see section 2.7.2), our model dynamically adapts to the window size selected for each task. This comes with a drawback, since when window sizes vary, the output length k of DECCaTNet also varies. The encoder weights remain the same and the same encoder can be used with different window sizes; however, each task requires a projector adapted to the output of size $k \times e$. Deciding the embedding size is therefore highly dependent on the window size of the task at hand.

In the surveyed literature, several different embedding sizes are proposed. Mohsenvand *et al.* [6] propose an embedding space of $(4 * input\_length)$.On the other side of the scale, Kostas *et al.* [5] propose a 1536-dimensional vector space with up to 15000 samples per input. We observe that the embedding-to-input ratio ranges from 4:1 to 1:10. Our selected k and e are, respectively, 49 and 40, creating an embedding size of 1960, and our input size for pre-training is 12000 samples per input. This gives an embedding-to-input ratio of about 1:6. This places us in the middle of the selected examples. When the window size varies, k will vary with it, keeping the same order of magnitude between the number of input samples and the embedding size.

Pre-traning projector To aid contrastive loss calculation during pre-training, the learned encodings are projected into a latent space for contrastive loss calculation, following several other relevant papers [6, 54, 56, 58, 67]. This is done to reduce the dimensionality and introduce nonlinear transformations. Introducing nonlinear transformations in fully connected layers may enable the model to capture more complex relationships and patterns.

For the pre-training task, DECCaTNet follows the simple implementation of Cheng *et al.*[67]. We deploy two fully connected layers with a nonlinear activation function (ReLU()) and a batch normalization layer to further encourage the model to generalize. Several more advanced projector implementations have been applied in the literature. However, since we wish to examine the transfer learning abilities of our DECCaTNet encoder, we do not wish to introduce unwanted noise from other complex models.

Model parameters The DECCaTNet encoder has some additional parameters presented in Table 3.7. All parameters are derived from similar papers. The size of the latent space was selected following the work of Cheng *et al.* [67]. The dropout rate follows the official Github implementation of the conformer for EEG [45].

Parameter	**Description**	**Value range**
Number of transformer heads	Increased numbers of heads allow for more fine-grained attention, enables capture of more detailed and diverse relationships in the input.	8
Number of transformer layers	Deeper layers lead to richer representations, but may affect the models ability to generalize. However, more layers increase the computational complexity.	8
Latent space size	Size of vector space used to represent the data after projecting.	128
Dropout rate	The rate at which randomly selected neurons are set to 0 during training to prevent overfitting. When performing validation, the dropout rate is set to zero.	0.5

Table 3.7: Parameters for DECCaTNet-encoder

3.4 Fine-tuning

Fine-tuning architecture The fine-tuning architecture differs from most SSL architectures, as the encoder only takes a group of channels as input. Therefore, the fine-tuning is structured similarly to SeqCLR [6] with its single-channel encoder, but instead with channel groups. See Figure 3.9 for an overview of the fine-tuning architecture. When fine-tuning the model, we again split the recordings into channel groups of size n, and pass each of the groups through the pre-trained encoder, now with frozen weights. The encodings produced by the encoder are then con-

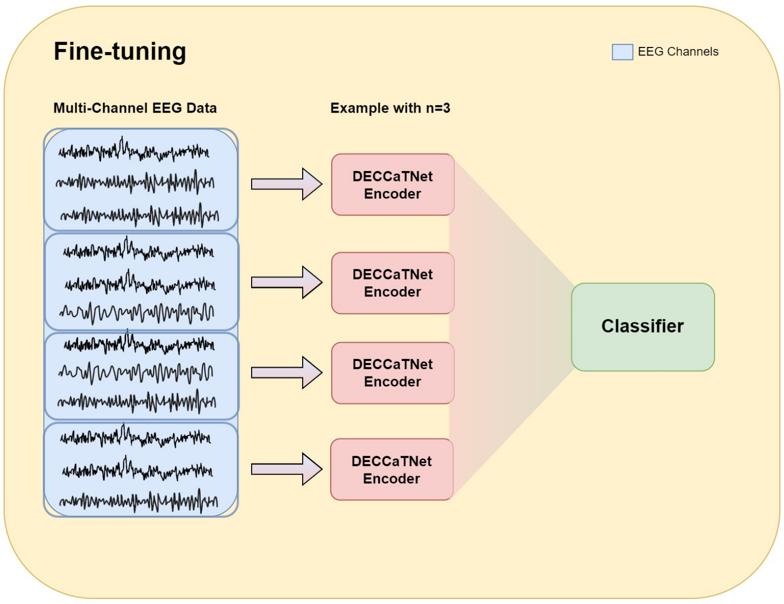


Figure 3.9: Architecture used for fine-tuning. This scenario has an EEG signal with 12 channels and $n = 3$. The signal is split into groups, then each group is encoded by the same encoder. The encodings are concatenated and fed to the classifier.

catenated and fed to a simple classifier. This classifier is the module that is trained during fine-tuning.

The data used for fine-tuning, and later the classification task, all need to have the same number of channels. This is because the classifier is trained with a fixed input size and the encoder produces an encoding for each group of channels. For the classifier to actually learn the correlations between the channel groups, the groups must be the same for every recording.

Splitting the fine-tuning recordings into channel groups is not done as a separate preprocessing step, but during training or classification. This is possible because, as mentioned in section 3.2, we have ensured that all the recordings in the fine-tuning dataset have the same channels in the same order. The fine-tuning module has a dictionary saved with the channel indexes for each channel group. When a recording is passed through the model, groups are made by extracting the correct channels from the original file by their indexes read from the dictionary. Each group of channels is encoded by the encoder, and the encoded outputs of all the groups are then concatenated into one tensor before being fed to the classifier.

The classifier is kept simple because we believe that the encoder has learned to create encodings with high expressive power, which can be classified without the need for a complex model. It is a simple fully connected feed-forward network with two hidden layers, each with ReLU activation and a set dropout rate. The parameters of the classifier network are shown in Table 3.8. The output uses softmax activation to make the output a probability distribution of the possible classes, as is common for classifiers.

Parameter	**Value**
Input size	*n_groups* × *encoding_size*
Dropout 1	0.4
Hidden layer 1 size	256
Dropout 2	0.2
Hidden layer 2 size	64
Output size	2 (Normal/Abnormal)

Table 3.8: Downstream classifier parameters

Training and validation We train the classifier for multiple epochs and report loss and classification accuracy on both the training and validation set. Details of the number of epochs are discussed in Chapter 2.

3.5 Experimental cycle

In order to conduct experiments and optimize the parameters of our DECCaTNet framework, we have implemented evaluation functions, parameter search, and training functionalities. This decision was driven by the realization that investing time in developing an efficient experiment regime would yield significant time savings during experimentation and tuning processes. To facilitate these tasks, we leverage the Python library Ray Tune [102], which provides preexisting functions that align with our requirements. The experimental cycle, depicted in Figure 3.10, follows a cyclical pattern, allowing us to run multiple trials within the same execution and utilize results from previous cycles to influence upcoming cycles.

RayTune Ray Tune is a flexible and powerful library used for hyperparameter optimization [102]. It seamlessly integrates with various optimization libraries, providing users with a wide range of options. Notably, the BOHB scheduler combines Bayesian Optimization and the Hyperband algorithm, enabling efficient and adaptive hyperparameter search [103]. With Ray Tune, we can easily explore the hyperparameter space and improve model performance.

Trainable Ray Tune requires minimal code changes. We wrap our training function in the `trainable` parameter, allowing Ray Tune to parallelize the process, report results, and change hyperparameters during runtime. Our extensive framework is accommodated by using a high-level `trainable` function that calls the appropriate operations based on the configuration file. We use `tune.with_parameters()` to incorporate the configuration file.

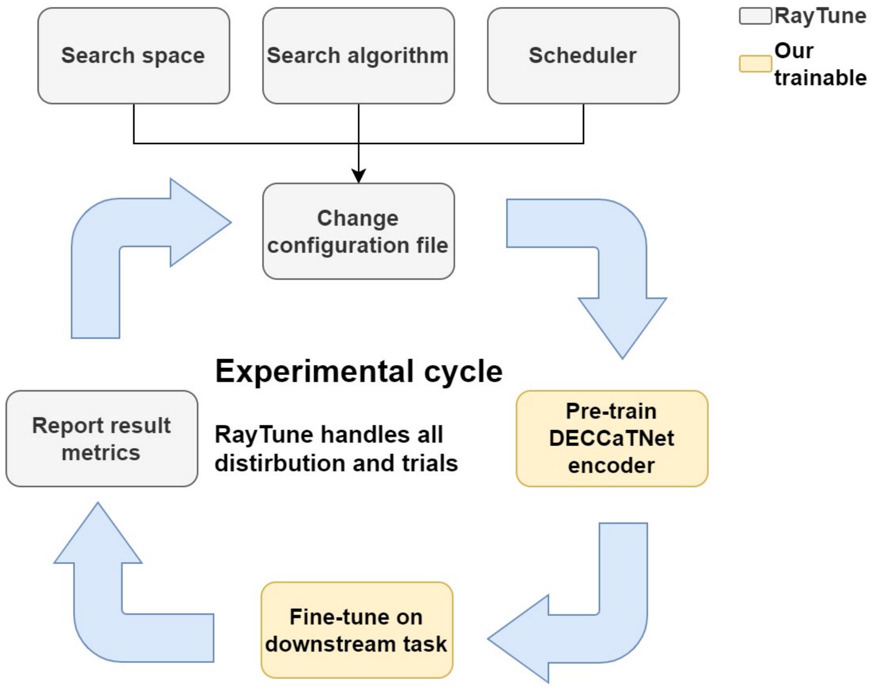

Search space
Search algorithm
Scheduler
RayTune
Our trainable
Change configuration file
Experimental cycle
RayTune handles all distirbution and trials
Pre-train DECCaTNet encoder
Fine-tune on downstream task
Report result metrics

Figure 3.10: The experimental cycle which is used to perform experiements and optimize hyperparameters.

BOHB and ConfigSpace Ray Tune integrates ConfigSpace, a powerful configuration space library, with its BOHB scheduler. ConfigSpace simplifies hyperparameter specification and management, supporting various types and constraints. By combining ConfigSpace with BOHB, we optimize the hyperparameter search process efficiently and effectively.

Configuration We use a YAML configuration file to run our entire framework. Ray Tune selects a defined set of parameters for each trial and updates the corresponding values in the framework. By updating the configuration dictionary for each run, we achieve efficient hyperparameter search and experiments with minimal code additions.

RayTune parameters Ray Tune requires som additional parameters to run as desired. These parameters are presented in Table 3.9. In summary, we run Ray Tune with a BOHB scheduler for hyperparameter tuning. Furthermore, a grid search is used in each experiment. The grid search ensures that we avoid the BOHB scheduler from early stopping a run or changing experiment parameters during the run. This will ensure that our hyperparameters are optimized in an efficient way, and our experiments will run in a straightforward smooth manner.

Parameter	**Description**	**Value range**
Fine-tune and pre-train split	Percentage split of available data used for training when running Ray Tune. This is in order to save computational time. Initial experiments which is used to reduce the search space utilizes a smaller amount of available data than final experiments	(0.1, 1)
Epochs	Maximum number of training iterations (epochs) for each trial of Ray Tune. Used to limit the resource usage for each run. Different size for pre-training and fine-tuning. Set the upper limit for resource usage for each trial.	(50, 100)
Number of samples	The number of trials to be performed for the given search. A larger value leads to a large search space which is computational expensive, but leads to more precise results. If a grid search is performed, a value of one will lead to all possibilities being explored.	(1, 25)
Mode	The goal of the hyperparameter search and experiment, do we want to maximize or minimize the given metric.	[min, max]
Metric	The chosen measurement for each run, will be handled according to the chosen mode.	[val_loss, val_accuracy]
Config	The parameters to be optimized or explored as well as the search space associated with each parameter. For example a list, range or boolean values.	see experiments (chapter 4)

Table 3.9: Parameters for RayTune experimental setup

Chapter 4

Experimental setup and results

This section presents the experimental setup and results on EEG classification using self-supervised learning and transfer learning techniques. The focus is on determining the optimal number of channels for pre-training, evaluating the impact of frozen encoder weights, and exploring different amounts of labeled data during fine-tuning. The findings highlight the importance of learned representations, the increased performance of pre-trained encoders, and the significant difference in classification accuracy between pre-trained and randomly initialized encoders when using limited labeled data.

4.1 Experimental setup

The experiments presented in this section were conducted with the goal of answering the research questions of this thesis. The model and data processing are described in chapter 3, hence this section only describes the experimental setup. After the experiments' implementation and used computation resources are introduced, the rest of the section introduces the conducted experiments one by one.

Experimental plan First, the optimal number of channels, n, in each channel group is examined. This is done by pre-training several n channel encoders with different values of n and obtaining results for each encoder on our selected fine-tuning dataset section 3.2. However, to ensure that the results obtained from this extensive search are valid, we first review the hyperparameters for fine-tuning and pre-training through a hyperparameter search. This initial search will also give a hint whether an n-channel encoder is able to perform fine-tuning on a downstream task after pre-training. After sufficient hyperparameters have been obtained, we will examine the optimal number of channels in each group for pre-training. With the optimal number of channels, we can finally document how well an encoder of n channels performs on different transfer learning tasks and further examine different aspects of our proposed method.

Each experiment is performed using the experimental cycle presented in method3.5. To define the search space for each run in Ray Tune, we pass a configuration file as a dictionary when initializing Ray Tune. The configuration file consists of the parameter we wish to optimize and how Ray Tune should traverse the search space. First, we will present the traversing options used.

tune.loguniform(a,b) With tune.loguniform(a,b) values are sampled uniformly between log(a) and log(b). As we use logarithmic, instead of a normal uniform distribution, we can sample fairly for values that vary over several orders of magnitude, such as the learning rate or weight decay. This search space definition is helpful when narrowing down a search space with initial sweeps over a large range of values.

tune.choice([a,b,...,c]) Samples randomly from a list of categorical values, [a,b,..,c], uniformly. Helpful when we wish to sample from a defined set, but do not need that all values are sampled.

tune.grid_search([a,b,...c]) Grid search is used mainly during the experimental phase in our case of experiments, for example with selecting the number of channels. Ensures that every value from the given list will be sampled. Additionaly, the number of samples presented in Table 3.9 decides how many times each value in the list will be samples, meaning that we can perform additional parameter searches on each value in the list.

4.1.1 Computation resources

Training was performed on a local machine at the Department of Computer Science at NTNU. Pre-processing was run on a CPU cluster with 56 Intel® Xeon® Gold 6132 CPUs running at approximately 2.97 GHz. The use of all 56 CPUs was not needed, as large parts of the preprocessing pipeline are not parallelized due to the Braindecode bug described in subsection 3.2.2.

Both pre-training and fine-tuning were run on the same machine's GPU using CUDA. The machine has two NVIDIA Tesla V100 PCIe 32 GB GPUs, which were both utilized during hyperparameter search, training, and testing. The hyperparameter search experiments were parallelized on the GPUs and took between 8 and 24 hours to complete, depending on the search.

4.1.2 Fine tuning parameters

In order to determine the best fine-tuning parameters, we used Ray Tune to perform a hyperparameter search. Listed in Table 4.1 are the parameters explored, as well as the respective search spaces and values. We ran Ray Tune using BOHB scheduler section 3.5, which leads to the scheduler allocating resources and controlling the hyperparameter search. As a result, we only need to determine the number of samples, which was 20. We selected maximize as mode and validation accuracy as metric.

Parameter	**Search space**	**Values**
Learning rate	tune.loguniform()	(1e-5, 1e-1)
Weight decay	tune.loguniform()	(1e-4,1)
Hidden layer 1	tune.choice()	[32,256]
Hidden layer 2	tune.choice()	[4,32]
Batch size	tune.choice()	[32,64,128]

Table 4.1: Experimental setup hyperparameter search for fine-tuning

Unfortunately, we were not able to experiment with the batch size because the available training resources were not sufficient. We ran into a memory allocation problem when using a batch size of more than 32, which led to the use of 32 as the batch size in this experiment.

Dataset and n We used all of TUAB (section 3.2.1) for fine-tuning and used a 0.8 validation / train split, which means that 80% of the data was used for fine-tuning, while remaning 20% was used for validation. Furthermore, we decided to select n channels equal to 2 while fine-tuning. This could induce bias when searching for the optimal number of channels; however, we had to select a value.

4.1.3 Pre-training parameters

For pre-training parameters, we use Ray Tune to pretrain several encoders and then freeze the encoders for fine-tuning, such that only the last layer is changed during fine-tuning. We use the parameters obtained from our fine-tuning parameter search for fine-tuning and explore the parameters listed in Table 4.2 for pre-training. The respective search spaces and values are also listed in Table 4.2. The batch size had to be adjusted to the aviable memory restrictions. The same scheduler, number of samples, mode and metric were used as in subsection 4.1.2.

Parameter	**Search space**	**Values**
Learning rate	tune.loguniform()	(1e-5, 1e-1)
Weight decay	tune.loguniform()	(1e-4,1)
Temperature	tune.loguniform()	(1e-3,1)
Batch size	tune.choice()	[64,128,256]

Table 4.2: Experimental setup hyperparameter search for pre-training

Dataset and n Similar to subsection 4.1.2 we used all TUAB for fine-tuning, 0.8 validation / train split and n channels equalt to 2. Furthermore, we used 10% of TUEG, 40% of TUAB, all SEED and all BCICIV for pre-training. The reason for not using all data for hypersearch is to save computing time. We trained each model for 4 epochs during pre-training. We prioritized the amount of data over the number of epochs.

Pre-training parameters 2 As we observed that our initial value ranges for learning rate and batch size were not sufficient, we decided to conduct another experiment to further examine the parameters used for pre-training. We follow the same strategy as in subsection 4.1.3, however, as we experienced that the temperature and batch size parameters were optimized, these were removed from the search. We also added the number of pre-training epochs performed, as the previous hyper-parameter search went faster than expected. The new selected values and parameters are presented in Table 4.3

Parameter	**Search space**	**Values**
Learning rate	tune.loguniform()	(1e-7, 2e-5)
Batch size	tune.choice()	[128,256,512]
Pre-training epochs	tune.choice()	[2,4,10]

Table 4.3: Experimental setup hyperparameter search for pre-training 2

As each sample in a batch now is a 2-channel recording, compared to fine-tuning which uses 23-channel recodrings. We can experiment with higher batch sizes for pre-training. Unfortunately, a batch size of 512 caused memory errors, which meant that only 128 and 256 as batch sizes were surveyed.

4.1.4 Optimal number of channels

In order to find the optimal number of channels, we use Ray Tune to perform a grid search over some chosen values for n. We selected a range of values, which we think will indicate which order of magnitude the optimal number of channels is. n equal 23 was included since this is the number of channels in TUAB, meaning that this encoder will mimic the behavior of most previous research, with one encoder for both pre-training and fine-tuning.

Parameter	**Search space**	**Values**
Number of channels	tune.grid_search()	[1,2,3,5,8,12,23]

Table 4.4: Experimental setup optimal number of channels

Optimal number of channels 2 Following the results of the previous experiment, we performed a second search to find the optimal number of channels. Since the most promising results were obtained by selecting $n = 5$, we have selected a new range of values of $[4, 7]$ presented in Table 4.5. All other parameters follow the previous experiment.

Parameter	**Search space**	**Values**
Number of channels	tune.grid_search()	[4,5,6,7]

Table 4.5: Experimental setup optimal number of channels

4.1.5 Frozen encoder weights and random initialized encoder

Now that we have found good hyperparameters and the optimal number of channels ($n = 4$) in each group, it is time to examine the quality of the learned representations during pre-training. In order to do this, we would like to examine how a pre-trained encoder performs compared to a randomly initalized encoder during fine-tuning. Additionally, we will also examine how freezing the weights of the different encoders affects the downstream classification performance. The search space is shown in Table 4.6, two coherent grid searches result in 4 trials, one for each possible combination. By doing this, we can examine whether the learned representations can be further advanced by adapting to the downstream task and whether the learned representations are actually useful.

Parameter	**Search space**	**Values**
Freeze encoder weights during fine-tuning	tune.grid_search()	[True, False]
Load pre-trained encoder	tune.grid_search()	[True, False]

Table 4.6: Experimental setup freeze encoder and initalization

Dataset and epochs We use the same hyperparameters for pre-training and fine-tuning as before. However, since we will only train one encoder for this experiment, we increase the pre-training dataset to inlude all of TUAB and 25% of TUEG. We also increased the number of pre-training epochs to 100; however, as this takes approximately a day to perform, we conduct experiments underway with encoders stored each tenth epoch.

4.1.6 Transfer learning

Now that we have conducted a search that determines the usefulness of the learned representations on TUAB classification. We want to survey how the learned representations are affected by the number of datasets used for pre-training. Previously, all datasets were included during pre-training. In this experiment, we will remove all datasets except TUAB during pre-training. By doing this, we can examine the importance of transfer learning during pre-training. Transfer learning constitutes that other datasets than the fine-tuning dataset are used during pre-training. We will run the same search space as in Table 4.6 and the same parameters for fine-tuning and pre-training as subsection 4.1.5. We will benchmark the pre-trained model without transfear learning with 50 pre-training epochs, which is the same as in subsection 4.1.5.

4.1.7 Adding additional noise

As discussed in Method subsection 3.3.4, we wanted to examine how adding a small noise in addition to already applied augmentations to each sample during pre-training would affect the learned representations. All previous pre-training experiments have been conducted without adding any additional noise. DUring this experiment, we will follow the pre-training strategies presented in subsection 4.1.5, but we will add a small noise to each signal during pre-training following the work done by Eldele *et al.* The results will then be compared to those obtained from subsection 4.1.5. We will also conduct an experiment where we make the CL pretext task harder by adjusting the augmentation parameters. The only goal of this experiment is to show the effect on number of required pre-training epochs, so we increased the parameters with a significant amount each. The new augmentation parameters are presented in Table 4.7.

Augmentation	**Parameter description**	**min**	**max**	**Visualization**
Permutation	number of permutations	5	10	Figure 2.7c
Masking	masked timepoints in a row	1500	4000	Figure 2.4b
Bandstop-filter	5 Hz width, Hz filter	20	82.5	Figure 2.8d
Additive noise (jitter)	standard deviation GS[1]	0.05	0.3	Figure 2.5c
Frequency-shift	shift in µV	-15	15	Figure 2.5b
Amplitude scale	scaling factor	0.1	3	Figure 2.4c
Time-shift	size of shift timepoints	-500	500	Figure 2.4a
Additional noise	standard deviation for GS[1]	0.02	0.1	Figure 2.5c

Table 4.7: Adjusted augmentations to create harder pretext task.

4.1.8 Percentage of labeled data

TUAB is currently one of the largest labeled EEG data sets available for public research. As several other EEG-datasets contain a significantly smaller amount of labeled data, we wish to examine how pre-training affects performance on limited labeled data. We introduced a new parameter: percentage of labels available during fine-tuning. This parameter is applied after the train/validation split, in order to ensure a large and balanced validation set. The different percentages examined are presented in Table 4.8, these percentages are similar to thos used by Mohsenvand *et al.* We enable trainable encoder weights during fine-tuning to simulate actual performance and benchmark the performance against a randomly initialized encoder. All other pre-training and fine-tuning parameters follow subsection 4.1.5. We use the encoder that are pre-trained for 10 epochs for fine-tuning.

[1]Gaussian Noise https://en.wikipedia.org/wiki/Gaussian_noise

Parameter	**Search space**	**Values**
Percentage of labels avaiable during fine-tuning.	tune.grid_search()	$[0.01, 0.1, 0.5, 1]$
Load pre-trained encoder	tune.grid_search()	[True, False]

Table 4.8: Experimental setup percentage of labelled data

4.1.9 Optimize performance on TUAB

Finally, opmitizing the performane of DECCaTNet on TUAB will give an indication of how our proposed framework performs compared to comparable studies. First, the hyperparameter search was performed as described in Table 4.1 with new value ranges as shown in Table 4.9. All parameter ranges have been adjusted according to our knowledge gained throughout this thesis. The encoder used is pre-trained as described in method for 10 epochs and has trainable weights during fine-tuning. Additionally, the final hyperparameters are listed in column final values highlighted in gray.

Parameter	**Search space**	**Values**	**selected values**
Learning rate	tune.loguniform()	(1e-7, 1e-4)	4e-5
Weight decay	tune.loguniform()	(1e-6,1e-3)	3e-5
Hidden layer 1	tune.choice()	[256,512,1024]	1024
Hidden layer 2	tune.choice()	[32,64,128]	64

Table 4.9: Setup final hyperparameter search

Lastly, the model with the best validation accuracy after 30 fine-tuning training epochs was saved and applied to the unseen official test set of TUAB. We fine-tuned on all available data in 2 minute windows before we validated DECCaTNet's performance on the test set by classifying using the first 2 minutes of each recording.

4.2 Results

4.2.1 Initial hyperparameter search

Fine-tuning The results of our hyperparameter search are presented in Appendix A. Table A.1 contains the results obtained from our hyperparameter search fine-tuning. We observe that our validation accuracy is almost on par with some papers surveyed in our project thesis, however, this is only on a validation set, and not the official test set for TUAB. Furthermore, we observe that the validation accuracy is moving strongly with the learning rate and weight decay used. Indicating that our model is sensitive to large variances in important hyper-parameters. Finally, the best results are obtained with the lowest possible learning rate, indicating that our search space for the learning rate needs to be extended for additional searches.

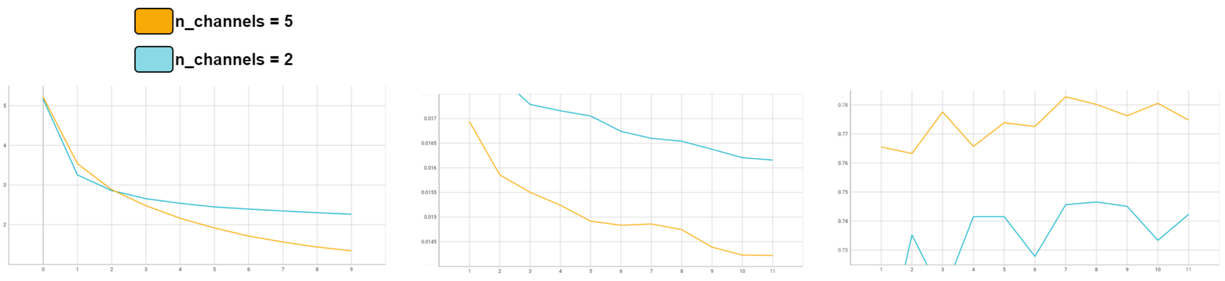
Pre-training From Table A.2 we observe that the accuracy obtained with a freezed, pre-trained encoder is worse than its fine-tuned counterpart. However, this is only a hyperparameter search, and further results will be reported later on. Here, we see that the correlation between pre-training learning rate and obtained validation accuracy is even stronger than for fine-tuning. As before, the lowest tested learning rate performed best, and further search is required to find a sufficient learning rate. Furthermore, a too high learning rate forced the encoder to learn no useful repsentations for donwstream tasks at all.

Second pre-training We ran an additional pre-training hyperparameter search with the reported results presented in Table A.3. Here, we survey the number of epochs used in pre-training, the batch size, and an even lower learning rate. The best-performing trial used 10 pre-training epochs, 128 batch size and a learning rate of approximately 2.4×10^{-6}. It achieved a validation accuracy of 76.6% with the second-best-performing trial at a validation accuracy of 74.0%. This large gap could be due to coincidences and the way BOHB chooses which trials to continue exploring; however, there seem to be some trends in the results, indicating which parameters to choose moving forward. All trials with 10 pre-training epochs achieve stable results, with no validation accuracies below 72.9%, which is higher than the averages of the trials with 2 and 4 pre-training epochs. The learning rate found in the previous experiment was sufficient, but a learning rate in the magnitude of the best trial is slightly better, and if it drops to low, the validation accuracy drops. ps with it. The batch size is demanding to optimalize, as a too large batch size leads to erros. However, we observe little difference between the trials with 256 and 128 in batch size. 128 is chosen because this might lead to fewer errors later, as a higher number of channels in each group will lead to larger memory requirements.

4.2.2 Optimal number of channels

Our first search for the optimal number of channels for pre-training in TUAB classification resulted in Table A.4. The best performing trial was with $n = 5$ channels. It is important to note that some trials suffered from overfitting, leading to a lower final validation accuracy than its maximum. Therefore, we have included train loss and validation loss to give an indication of which trials are overfitted and not. For example, the trial with $n = 12$ channels, which was most exposed to overfitting, reported 71.0% validation accuracy at its best. However, we do not observe a clear trend with regard to which side of $n = 5$ is the most beneficial and will perform another experiment for the optimal number of channels. The fact that all trials are able to learn useful representations for downstream tasks is evident from Table A.4.

Loss propagation To show some interesting behavior that occurred during the search, we have included Figure 4.1. Looking at Figure 4.1a and Figure 4.1b we



(a) Pre-training loss **(b)** Fine-tuning training loss **(c)** Validation accuracy

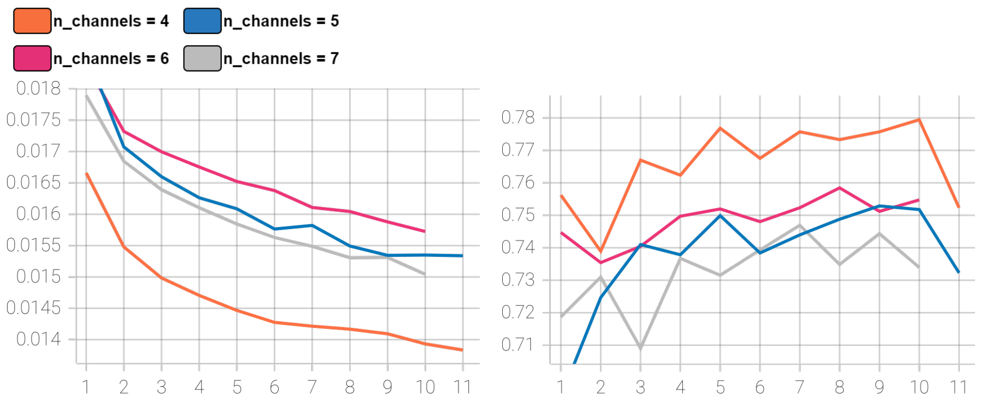
Figure 4.1: Metrics from channel selection for $n = \{2, 5\}$

observe that the difference in pre-training loss propagates from pre-training to fine-tuning. We also observed this for the other values of n, however, not as clearly as the presented example. Looking at Figure 4.1c, it is clear that the propagation of pre-training loss affects the final validation accuracy.

Second search We performed a second experiment after reviewing the results presented in subsection 4.2.2. The results are reported in Table A.5 and shown in Figure 4.2. Additionally, we observed the same behavior regarding loss propagation as in subsection 4.2.2. For $n = 5$ channels, the accuracy reported for the second experiment is approximately 1% lower than in the first, showing that there is variance outside of our control. Optimally, we would perform more trials in each experiment, but we are limited by our computational resources and time. However, these results show that both $n = 4$ and $n = 5$ channels are good choices, with high accuracy and low loss reported. We also observe that selecting n outside of $n = \{4, 5\}$ leads to worse performance. We use $n = 4$ channels moving forward for the next experiments, as $n = 5$ did worse than $n = 4$ in the second experiment.

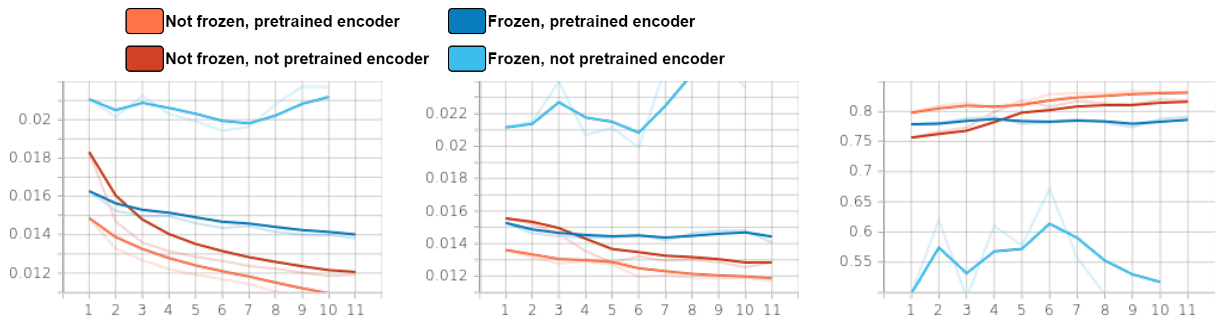
4.2.3 Frozen encoder weights and random initialized encoder

This section present results which indicate the usefulness of the representations learned during pre-training. The metrics obtained are presented in Table A.6. Whether the encoder is frozen or not, indicates if the weights of the encoder will be updated during fine-tuning or not. Pre-trained encoder indicates if the encoder used during fine-tuning is pre-trained or has randomly initialized weights. We also included plots of each metric and how they progress during the epochs in Figure 4.3. We observe that the best results are obtained with a pre-trained encoder with non-frozen weights (83. 3%), beating a randomly initialized encoder with non-frozen weights of more than 1% (81.9%). The pre-trained encoder with frozen weights also performs better than previously reported encoders with 79.1% validation accruacy. These results were obtained with an encoder pre-trained for 50 epochs.



(a) Fine-tuning training loss **(b)** Validation accuracy

Figure 4.2: Metrics from channel selection for $n = \{4, 5, 6, 7\}$. We see the validation accuracy degrades for epoch 11, indicating overfitting, but the reported results are sampled from epoch 10. Pre-training loss is not shown as the different values of n all show the same behavior.

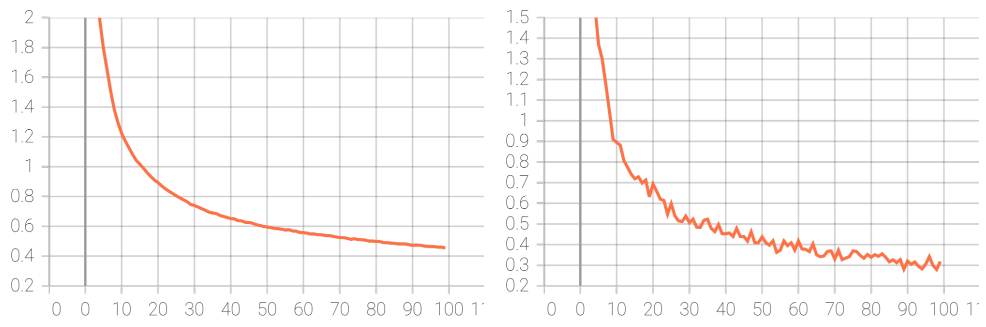


(a) Training loss **(b)** Validation loss **(c)** Validation accuracy

Figure 4.3: Metrics from finetuning with frozen/unfrozen encoder weights and random initialized encoder vs pretraining

Fine-tuning trends There are several interesting properties emerging in Figure 4.3. Firstly, the performance gap between pre-trained and randomly initalized encoders is persistent during all fine-tuning epochs. We observe that the gap narrows during the first epochs and eventually stabilizes, indicating that pretarning is essential to learn some valuable representations useful for classification on TUAB.

Second, frozen encoders struggle to increase their performance while fine-tuning. With a pre-trained encoder, the validation accuracy is marginally improved during fine-tuning, while a randomly initalized frozen encoder is able to learn little. This indicates that the classifier is able to project the learned representations but is not able to learn any representations itself. Showing that the results heavily depend on the quality of the encoder



(a) Training loss **(b)** Validation loss

Figure 4.4: Metrics from pre-training for 100 epochs. Both validation and training loss behaves gradually decreasing.

Number of pre-trainin epochs During this experiment, we pre-trained an encoder for 100 epochs, and saved copies of the encoder for each tenth epoch. To determine the effect of a larger number of pre-training epochs, we selected some of the saved models and evaluated their performance on TUAB. The obatined results are presented in Table A.7.

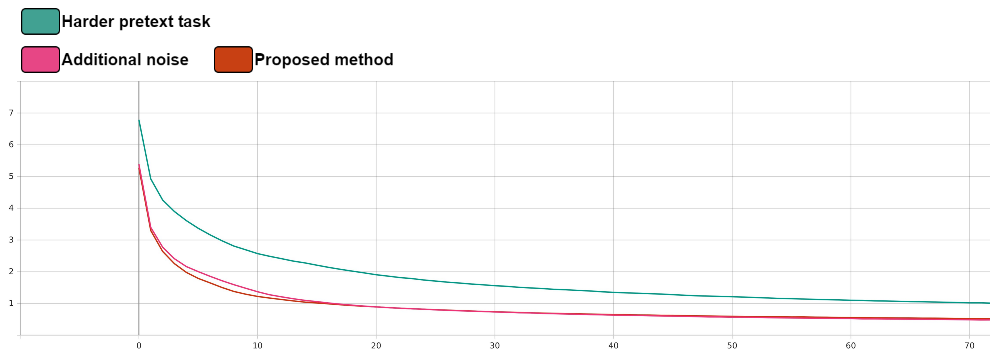
Looking at Figure 4.4 indicates that our model is gradually becoming better and better at contrastive learning, decreasing both training and validation loss for each epoch. These results are not reflected in the results reported in Table A.7, as the least pre-trained model achieves the best validation accuracy (81.5%). There is a negative correlation between the number of pre-training epochs and the validation accuracy obtained, with the model with the most pretraining reaching the lowest validation accuracy of 79.5%.

4.2.4 Transfer learning

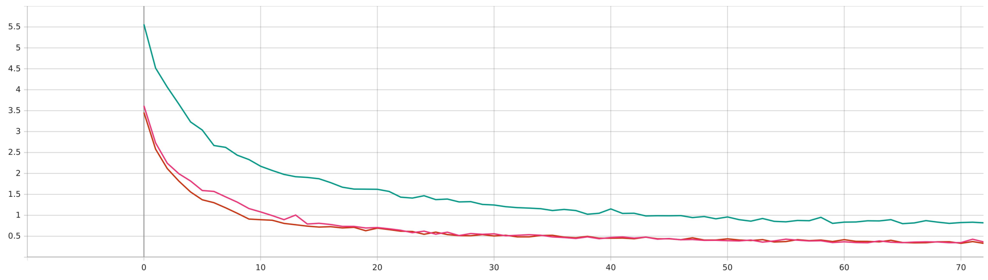
In order to determine the effect of transfer learning, we examined how an encoder that only use TUAB for pre-training performs on TUAB during fine-tuning and compare it with previously obtained results. The results presented in Table A.8 suggest that both the frozen (77.7%) and nonfrozen (82.3%) encoder only pre-trained on TUAB perform approximately 1% worse than its counterpart pre-trained on several datasets with the same number of pre-training epochs (50).

4.2.5 Additional noise and harder pretext task

To determine the effect of harder augmentations, we performed two experiments. The results of adding additive noise can be seen in Table A.9 and the results from making the pretext harder by changing the augmentations can be seen in Table A.10. The pre-training loss and validation for the different pre-training trials can be observed in Figure 4.5. It is obvious that the changed augmentations actually made the pretext task harder, hence both the validation and training contrastive loss being higher during pre-training. The additional additive noise seams



(a) Training loss



(b) Validation loss

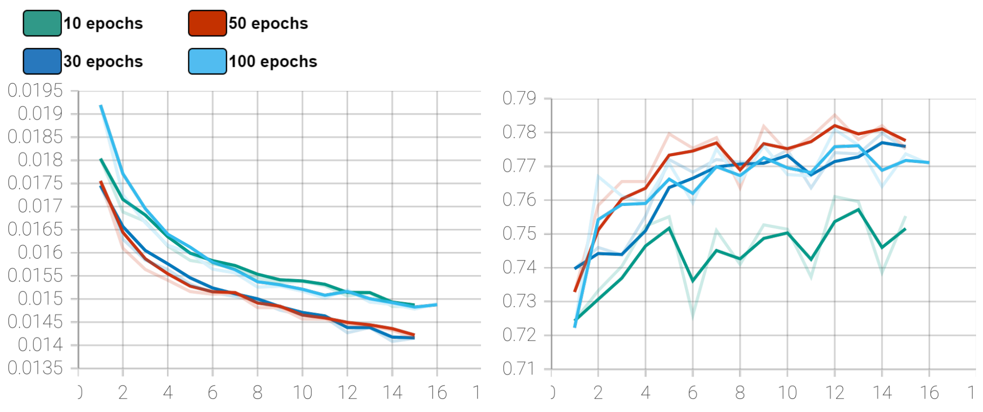
Figure 4.5: Comparison of metrics from 100 pre-training epochs. Presenting results from the proposed method, additional additive noise, and harder augmentations. The last 30 epochs are omitted from the figure, as no new trends appeared.

have very little impact on the loss, except for a small deviation during the first 15 epochs.

We observe that both experiments reduce the validation accuracy; however, the nature of the obtained accuracies has changed. The differences between the best and worst obtained valiation accuracy are less than 1% for additional noise (Table A.9) compared to 2% difference (Table A.7) for the proposed method. For the harder pretext task, the best validation accuracy is from the encoder pre-trained for 50 epochs (Table A.10). Also, the worst performing encoder is the one pre-trained for 10 epochs. By looking at Figure 4.6 we suggest that the previously observed negative correlation between the number of pretraining epochs and validation accuracy is not present when increasing the difficulty of the pretext task.

4.2.6 Percentage of labeled data

To further document the effect of SSL, the effect of different amounts of fine-tuning data on classification performance in TUAB is examined. The results obtained are presented in Table A.11. There exists a notable difference between pre-trained and random initialized encoders in classification accuracy when using as much as 10% of labeled data during fine-tuning. Moreover, the amount of



(a) Training loss **(b)** Validation accuracy

Figure 4.6: Metrics for experiment with changed augmentations. Evidently higher correleation between number of pre-training epochs and fine-tuning accuracy.

pre-training data from 50% to 100% seams to have little effect on classification performance. The previously best classification validation accuracy obtained on TUAB is 83.3%, so results with 50% of the available labeled data for fine-tuning are comparable (83.0%).

4.2.7 Optimize performance on TUAB

Following 30 epochs of fine-tuning with TUAB, our model achieved its highest validation accuracy of 84.55% in epoch 24. Subsequently, we saved this model and evaluated it on the TUAB holdout test set, where it achieved a test accuracy of 84.26%. Although our thesis does not primarily focus on achieving the highest possible accuracy, we provide a comparison of our results with other relevant research studies in Table 4.10.

While our study yields promising results in EEG classification using SSL and transfer learning techniques, it is important to note that our performance is still relatively distant from the SOTA performance achieved in other research studies [59, 80].

[2]Amin *et al.*
[3]Oord *et al.*
[4] Banville *et al.*
[5]Wagh *et al.*
[6] Most precise reported accuracy shown
[7]Mohsenvand *et al.*

Model	**Accuracy**	**SSL**
AlexNet [2]	87.32	✗
Alhussein *et al.*	87.68	✗
CPC [3,4]	83.51	✓
RP [4]	83.37	✓
TS [4]	84.99	✓
HS-BSE-AC [5]	85.0 [6]	✓
Fine-tuned SeqCLR - C [7]	87.21	✓
DECCaTNet (ours)	84.26	✓
Ko and Suk	87.7 [6]	✓

Table 4.10: Final result and comparable results

Chapter 5

Discussion

This chapter contains a discussion of our model. First, the general discussion is structured by parts of the model, discussing the general attributes and behavior of the DECCaTNet architecture and how it relates to the experimental results. Finally, possibilities for future research related to the model are discussed in section 5.4.

5.1 Grouping channels

Inspiration As mentioned earlier, the idea to group channels and use the same encoder on each group was inspired by Mohsenvand *et al.*'s SeqCLR model [6]. However, as their encoder only looks at a single channel, the encoder does not capture inter-channel dependencies, which leaves them to be learned by the classifier during fine-tuning. As the fine-tuning dataset is typically much smaller than the pre-training dataset, we view this as a potential weakness. We reintroduce spatial inter-channel dependencies to the encoder while preserving the flexibility of SeqCLR by grouping the channels into groups of size n. With $n = 1$, DECCaTNet is a variant of SeqCLR with different encoder, projector, and classifier architectures, but otherwise largely similar.

Group size n Deciding the correct group size n is a balancing act between flexibility and task-specific specialization. Lower values of n make it possible to pre-train on a larger, more diverse selection of data sets with fewer channels, as an EEG recording needs to have a minimum of n channels to be able to extract groups of n channels. Higher values of n should allow the encoder to learn more complex spatial dependencies, but for the channel grouping function `make_adjacent_groups` it reduces the number of groups that can be used for training. Higher values of n can also make encodings too varied and worsen the generalizability of the model as the possible semantic space of the EEG channel group grows exponentially for each channel added to the group size.

Our architecture creates one encoding per group of channels. Our encoder implementation uses the same number of spatial convolutions as the number of channels, ensuring that the output size is independent from n. As a result, having more groups and thus stacking more encodings during fine-tuning, will lead to more trainable weights in the initial layer of the classifier. For example, using an encoder with $n = 23$ channels in TUAB would create a classifier input size equal to the embedding size. However, an encoder structure with $n = 1$ channels would require 23 encoders, giving a classifier input size equal to 23 × (embedding size).

However, a higher n leads to additional spatial convolutions in the encoder, compensating for the number of trainable weights in the initial classifier layer and allowing the encoder to learn more complex spatial dependencies.

From our final tests to determine n, we see that $n = 4$ performed best (Figure 4.2). This shows that $n = 4$ is a good middle ground where inter-channel dependencies are learned during pre-training, while generalizability is kept high. For our fine-tuning set where we use $N = 23$ channels, $n = 4$ leads to 6 groups, and thus 6 forward passes through the encoder per prediction.

Order of channels for groups The channel grouping function `make_adjacent_groups` (Algorithm 1), which we have used in our testing, and `make_overlapping_adjacent_groups` both take a list of channel names as input where the order of channels matters. Groups are created by extracting subsequences of channels from the full list, so only channels less than n positions apart in the list can end up in the same group.

After manually reviewing a representative selection of files in each of the datasets used, we decided to keep the channel lists in the original order in which they were saved in the EEG recordings. The channels are in a large majority of recordings saved in an ordered way following the variant of the EEG montage used, often sorted by electrode location in frontal-dorsal, then medial-lateral order.

Another option was sorting the channel list alphabetically, but we decided that in the majority of cases the original channel ordering would better reflect physical location proximity of the electrodes. The initial letters in the channel names of the 10-20 system and its derivatives are chosen for names of parts of the brain, and thus, their alphabetic order does not reflect physical location as well as the original order.

Other channel grouping techniques In total we implemented four different channel grouping functions, but only `make_adjacent_groups` was tested thuroughly. An interesting note about `make_overlapping_adjacent_groups`, which makes $N-n+1$ channel groups, is that the encoder is applied like a convolution filter to the list of channels. This comparison makes the architecture easier to understand for those familiar with CNNs. Another advantage is that it generates more groups which can be used for training or classification; however, in our case, the computational cost and time were the limiting factor, not the amount of data.

`make_all_combinations` and `make_all_permutations` both have two distinct advantages over our adjacent groups functions. They generate more groups and original channel order does not matter as much. `make_all_permutations` makes all possible permutations of n different channels, including those using the same set of channels in a different order. As it generates

$$P(N,n) = \frac{N!}{(N-n)!}$$

groups, which is a truly massive number, we fear it would bloat the dataset too much even if using the best computational power available. As an example, for $N = 23$ channels and group size $n = 4$, a total of 212520 groups would be created from each recording. To reduce this number, it is natural to make all unique combinations instead of all permutations of channels. `make_all_combinations` generates

$$C(N,n) = \frac{N!}{n!(N-n)!}$$

groups. Compared to permutations, combinations significantly reduce the number of groups while only losing groups that are different orderings of the same channels. To reuse the same example with $N = 23$ and $n = 4$, a total of 8855 groups would be created from each recording. Still a massive number when compared to the adjacent group functions, but it could be beneficial in some cases and for some values of N and n. The number of groups for the $N = 23, n = 4$ example for each of the proposed functions are shown in Table 5.1.

Grouping function	Number of groups
`make_adjacent_groups`	6
`make_overlapping_adjacent_groups`	20
`make_all_permutations`	212520
`make_all_combinations`	8855

Table 5.1: Number of groups for the example $N = 23, n = 4$ for each of the four channel groupings proposed by us.

It is important to keep in mind that a different channel grouping function could be used to generate groups for pre-training than the one used for fine-tuning and the classification task. During fine-tuning and the classification task, we are limited by having to pass each channel group through the encoder, which is a major bottleneck if we have a large number of groups. Additionally, without further testing, it remains unclear whether adding more groups with more duplicates of channels aids in improving the classification performance. Due to this, we believe that of the functions we propose, `make_adjacent_groups` or `make_overlapping_adjacent_groups` are best suited for use in fine-tuning and classification. However, this assumption should be verified or disproved through further tests.

Given enough computational power and time, the amount of available EEG data could be the limit to further improve performance. In this case, for example, `make_all_combinations` could be used to generate many additional groups that can be used for pre-training. Generating groups in this way for the sole purpose of pre-training could be viewed as a data augmentation technique.

5.2 Impact of SSL

Several experiments were performed to determine the effect of the proposed SSL framework. In this section, we delve into the outcomes and achievements observed during our experiments, shedding light on the impact of SSL on various aspects of our research domain. By comparing the results obtained from our SSL framework with those achieved through traditional supervised learning methods, we aim to provide a comprehensive understanding of the advantages and advancements brought forth by this novel approach.

5.2.1 Frozen encoder weights

As explained in our experimental plan, we wanted to find sufficient hyperparameters before conducting any experiments. For fine-tuning hyperparameters, we trained both the encoder and the classifier. This was due to unknow hyperparameters for pre-taining, and we wanted to ensure that we found parameters that allow training of the classifier during fine-tuning. However, when aquring pre-training hyperparameters, we froze the encoder weights during fine-tuning. Frozen encoder weights are common when evaluating learned representations [6, 54, 58, 67].

As a result of frozen encoders, we can ensure that the difference in donwstream classification performance is due to the quality of learned representations. A clear example of this is the reported result of a frozen randomly initialized encoder on TUAB (Table A.6,Figure 4.3). Since we know that the representations produced by the encoder are most likely very poor, the fact that the classifier varies and performs poorly shows that the representations produced by the encoder are useful for downstream classification results. We also observe this trend during channel selection (Figure 4.1). The loss propagates from pre-training loss to validation loss, indicating that bad representations are a direct cause of worse classification performance. As a result, when finding the optimal number of channels for pre-training or performing a hyperparameter search, freezing the encoder weights ensures that it is the learned representations that are examined.

The frozen randomly initialized encoder also indicates that our classifier is only able to project good representations instead of learning complex patterns on its own. As our classifier structure is simple, with two fully connected dense layers, this is no surprise. Additionally, with frozen encoders, the computational requirements of fine-tuning are limited as the number of trainable weights is reduced drastically. We can use this to increase the size of our experimental searches, exploring more parameters for both fine-tuning and pre-training.

In subsection 4.2.3 we also observe that due to the simplicity of our classifier, we struggle to increase the performance of frozen encoders during pre-training. However, we observe that trainable encoders lead to increased performance, outperforming its fully supervised counterpart. This indicates that to maximize the performance of our current architecture, we need to use trainable encoders during fine-tuning. This should also raise some questions about our classifier structure in general, as some articles reported their best results with frozen encoders/backbones [58, 67].

5.2.2 Transfer learning

Transfer learning aims to leverage and learn knowledge about a specific context such that it becomes useful for a different one. To determine whether our encoder is capable of showing transfer learning capabilities, we conducted the experiment described in subsection 4.1.6. As presented in subsection 4.2.4, the encoder pre-trained on TUAB performs approximately 1% worse than the encoder utilizing data from other contexts as well. This may indicate that our model is able to learn knowledge from other contexts and deploy it in a similar domain. Our findings follow those of Mohsenvand *et al.*, who increased their performance by some percentages by using dataset fusion during pre-training. Dataset fusion constitutes the use of several datasets during pre-training compared to only using the fine-tuning dataset during pre-training.

An interesting feat is that when looking at the results obtained with frozen encoders with and without dataset fusion (Tables A.6 and A.8), we observe that dataset fusion increases the validation accuracy almost 2%. One might think that pre-training on only TUAB would create the best encodings for classification on TUAB, especially since the encoders remain unchanged during pre-training. However, these results show that our model is able to use meaningful features learned from other datasets to increase performance during fine-tuning. This is sustained by the learned representations of dataset fusion remaining better even when training the encoder during fine-tuning.

However, labeling the mentioned experiment as showing true transfer learning capabilities might not be correct. This is due to the fact that TUAB was used during both pre-training runs. A true test would be if TUAB were applied in pre-training as the sole dataset and compare it with the performance of a run that did not use TUAB at all during pre-training such as Kostas *et al.*, Eldele *et al.* Further research on this matter would be very interesting moving forward.

5.2.3 Percentage of labeled data

The notable difference discovered in subsection 4.2.6 highlights the need for SSL in EEG classification. One percent of labeled data equal 30 patients, which is a comparable number of patients to most EEG datasets [1]. DECCaTNet is unable to generalize at all without a pre-trained encoder with one percent of labeled data, achieving a validation accuracy of 52.5% which is comparable to random guessing. Its pre-trained counterpart on the other hand reports accuracy in the same order of magnitude as a model fine-tuned on all labeled data with accuracy 78.7%. The difference between random guessing and obtaining comparable results is significant, showcasing the usefulness of our learned representations during pre-training. After performing previous experiments, we did not expect such clear differences and regard these findings as our strongest proof of the quality of our learned represntations. Even with 10% of the labeled data (300 patients) used for fine-tuning, which is considered a large data set in the EEG domain, the difference is notable.

Comparison with other models It is common practice in SSL for EEG to use limited aviable data to fine-tune the learned representations [6, 54, 58]. Eldele *et al.* went as far as only reporting results with 1% and 5% of the available data to fine-tune, not touching the remaining 95%. This could indicate that others have also discovered that it is difficult to show the difference between pre-trained and random initialized encoders when using large amounts of data for fine-tuning. Following Mohsenvand *et al.*, we can compare our results with limited label data for fine-tuning with seqCLR [6], Contrastive Predictive Coding (CPC) [8, 101], Temporal Shuffling (TS) [8], and Relative Positioning (RP) [8].

Model	**Accuracy**		
Percentage of labels	1%	10%	50%
CPC [8, 101]	72.04	75.62	81.24
RP [8]	75.17	77.13	82.16
TS [8]	77.84	79.50	82.46
SeqCLR - C [6]	78.53	85.44	85.52
Fine-tuned SeqCLR - C [6]	83.19	86.98	87.21
DECCaTNet	78.87	81.46	82.95

Table 5.2: Model accuracy on TUAB with percentage of labels compared to relevant papers

Table 5.2 shows the classification accuracy compared to other SSL-based approaches. We see that we cannot improve the results of Mohsenvand *et al.*, however, we present a consistent improvement compared to other SSL-approaches for all percentage of labels. Intrestingly, Banville *et al.* uses a similar classifier to DECCaTNet and has the same tendency to have a distinct increase in performance with more labeled data. Our 1% classification accuracy is similar to SeqCLR with frozen weights, but even with frozen encoder weights, SeqCLR is able to better utilize obatined representations during fine-tuning. Mohsenvand *et al.*

[1]`https://github.com/meagmohit/EEG-Datasets`, Accessed: 2023-06-06

have a complex classifier based on several bi-directional LSTM's in parallel with donwsampling and dense layers. Again, this shows that our learned representations are useful, but that our classifier may not be sufficient to achieve near SOTA performance.

5.2.4 Optimized performance on TUAB

Our final test results, as shown in Table 4.10, align closely with our expectations. Considering that this work represents the first exploration of channel grouping with $n > 1$ for pre-training across multiple datasets in any domain, it is not surprising that we did not achieve SOTA performance right from the start. Furthermore, it is important to note that the primary objective of our fine-tuning classifier was not to maximize accuracy, but rather to explore and evaluate the effectiveness of our proposed framework.

However, achieving comparable results to existing studies in the field is a significant step forward. It showcases the potential of using transformers for SSL in the EEG domain. While other deep learning architectures could have been employed within the same framework, these results highlight the promise and worth of further exploration into the use of transformers for EEG classification.

There is a possibility of increasing the performance of the fine-tuning test set. As mentioned earlier, we only classified the first 2-minute windows of each recording. Previous research typically focuses on classifying the first minute of recordings, considering the decline in signal quality over time due to sensor drying and sweating. We obtained optimal validation results using 2-minute windows and decided to stick with this configuration during testing. However, studies by Roy *et al.* and Mohsenvand *et al.* have demonstrated that utilizing the first 11 minutes of recordings in 1-minute windows can lead to improved classification results. Unfortunately, the details regarding the voting procedure over the 11-minute recording were not clarified in their work, and we did not have sufficient time to implement our own version. Exploring this approach could potentially further enhance the performance of the test set.

5.3 Architecture

This section discusses the architecture of our model, from its SSL architecture and pretext task, to the details of its submodules, such as the encoder and classifier networks. We discuss our architectural choices tied to the experimental results.

5.3.1 Pretext task

Our pretext task contrastive learning is popularly deployed in SSL for EEG-classification (subsection 2.4.1). Initially, we excpected that more pre-training would increase our performance. Looking at Mohsenvand *et al.*, which pre-trained their encoders for 300 epochs without performing any ablation study on the number of pre-training epochs, we imagined that we would only be limited by training resources. However, considering the results presented in section 4.2.3, we experience a decrease in performance after only 10 epochs and observe a negative correlation between the number of pre-training epochs and the obtained validation accuracy. It is quite clear that the pretraining loss decreases from Figure 4.4. This indicates that the observed correlation between pre-training loss and validation found when selecting the number of channels is not valid when comparing the same model with different number of pre-training epochs.

However, the number of pre-training epochs used in the litterature varies. For example, Jiang *et al.* used 70 pre-training epochs without supporting this choice, indicating that they may have also struggled with overfitting on the pretext task. Finally, Eldele *et al.* pre-trained for 40 epochs, as they noticed that their performance did not increase with further training. We are not alone in encounting this scenario, but we require fewer pre-training epochs before reaching our optimal number of epochs, which could be due to several reasons.

The first reason could be that compared to Mohsenvand *et al.*, we pre-train on twice as much data. Larger amounts of data could mean that a fewer number of epochs are required in order to learn useful representations for the donwstream task, and that all epochs after this create too specialized representations overfitted on the pretext task. However, increasing the amount of data is often seen as a way to increase the generalization in machine learning (ML).

Another reason could be due to the nature of our pretext task implemented. We hypothesize that an easy pretext task would be easier to overfit on than a more complex one during pre-training. Meaning that our early overfitting during pre-training could be a consequence of an easy pretext task. We will not prioritize changing the nature of our pretext task, as this requires a complete overhaul of our implementation. However, thoughts on how this could be achieved are included in further work (section 5.4.1). Althought, what we can examine is if we can make our already implemented contrastive learning task harder by adjusting our augmentations, and acquire knowledge about how this affects overfitting during pre-training.

5.3.2 Augmentations

As previously stated in subsection 3.3.4 most of our data augmentations were copied from Mohsenvand *et al.* [6], who chose their augmentations to be domain-specific with the help of neurologists. They performed sufficient tests to justify their augmentation choices, and therefore we believe their augmentations to be good enough to test the performance of our model.

We added a single augmentation that utilizes the multi-channel property of our encoder, which is channel permutation. This augmentation, which randomly reorders the channels in a group, makes the encoder stronger, as channel groups in a data set can consist of many different channel combinations. If the encoder learns to see inter-channel dependencies regardless of channel position, it makes it more robust.

Augmentation parameters In subsection 4.2.5 results that further elaborate our understanding of pre-training overfitting are presented. The main finding from these results is that the negative correlation between the number of pre-training epochs and validation accuracy is not necessarily valid for all augmentation selections. First, by making the pretext task slightly harder by introducing additive noise for all samples during pre-training, the variation in obtained validation accuracy decreased from 2% to 1%. Lastly, when altering most augmentation parameters, the model pre-trained for 10 epochs was evidently worse than the more pre-trained models, suggesting that our framework may be able to take advantage of more pre-training epochs in the future.

Notably, even though increasing the hardness of the pretext task enabled more pre-training epochs, the validation accuracy decreased significantly. This could indicate that our initial augmentation parameters, inspired by Mohsenvand *et al.*, were appropriate for the task at hand. However, as we are able to utilize less pre-training than similar studies, we suggest that our pretext task is possible to improve. This would be regarding both designing a new pretext task or altering the augmentation selection technique.

Seen in the conetx of Eldele *et al.*, all our chosen augmentations, except permutation, would be labeled as weak augmentations. With randomized augmentation selection, the majority of positive pairs fed to the contrastive learning module come from two weak augmentations. Simply forcing weak augmentations closer to strong augmentations by altering the parameters may confuse the encoder more than enrich the learned representations. Instead of forcing weak augmentations closer to strong augmentations, perhaps more strong augmentations should have been introduced in the first place.

The magical effect of additional additive noise seen in Eldele *et al.* was not present in our results (subsection 4.2.5). For Eldele *et al.*, adding additive noise was a way of introducing randomness, as a fixed strong and weak augmentation was used. In our case, we have alread introduced randomness when selecting which augmentations to apply and their corresponding parameters. Intorudcing additional randomess may have led to confusion for the encoder, thus the worse performance reported. This highlights a minor weakness of our study, that time was not prioritized towards an augmentation ablation study. Restructuring the proposed augmentation technique is included in our further work.

5.3.3 Encoder architecture

The objective of enabling large-scale pre-training and transfer learning led us to select an architecture that excels with extensive pre-training. Existing research has consistently shown that transformers perform exceptionally well when pre-trained on massive datasets. Our experiments further support this notion, as our architecture, based on Conformer [104], demonstrates improved performance with pre-training compared to without.

However, as previously discussed in subsection 5.3.2, we encountered challenges regarding the number of pre-training epochs required before overfitting occurs. This raises the question of whether this issue stems from the architecture of the encoder, classifier, or the nature of the pretext task employed. Unfortunately, our experiments did not incorporate alternative encoder architectures, limiting our ability to conduct a comprehensive analysis of transformers' complete learning capabilities for EEG. Exploring different architectures within the same framework is an important avenue for further research, as mentioned in the further work section.

While we acknowledge the limitation of not comparing our architecture to other alternatives in the self-supervised learning framework, we believe our results, when considered alongside the successes reported in related studies Kostas *et al.*, Bagchi and Bathula, highlight the untapped potential within EEG classification using transformers. Despite the absence of direct comparisons, the comparable performance achieved by our architecture and the success observed in other studies suggest that transformers can play a pivotal role in advancing EEG classification tasks.

Window size From preliminary experiments it was observed that fine-tuning and classification results suffer greatly when the window size is too small. We initially tested fine-tuning on 30 s windows of TUAB normal/abnormal samples, but the model was unable to classify windows this short. To quote Mohsenvand *et al.*, "For longer sequences the contrastive loss falls rapidly during training since distinguishing between long samples is easier; however, the features learned on longer sequences, do not perform as well in classification" [6].

From this statement, and our own results from early experiments, we see that for fine-tuning longer window sizes are beneficial, while for pre-training we prefer shorter window sizes. We chose to use 60 second windows for both pre-training and fine-tuning during our own tests to keep our preprocessed datasets consistent.

5.3.4 Fine-tuning classifier

Our fine-tuning classifier architecture is as simple as possible. This is to ensure that the representations learned from SSL determine the performance, and not a complex projector or classifier. A complex classifier could be used to obtain better classification results [6], however, a simple classifier generates reliable results which correlate with the representations learned for internal ablation stuides [58], which is our main goal.

During the experiments, the fine-tuner classifier showed a tendency to overfit during fine-tuning and getting worse validation accuracy as early as in epoch 11, with the encoder frozen after being pre-trained for 10 epochs on about 50 GB of data (Figure 4.2). This may indicate that the classifier is too wide and overfits the data, but further experiments with the encoder pre-trained for more epochs showed no signs of overfitting after 20 epochs of pretraining. Improving the encoder seems to improve classification stability.

Throughout all experiments, we observed fluctuations in the validation accuracy often around 1 and 2% from epoch to epoch during training, but with an evident rising trend for the configurations that showed the most promise. The only generalization features included in our classifier are the dropout layers after each dense layer.

We believe it may be beneficial to experiment with more sophisticated classifier architectures (see section 5.4, Further work), but at the same time our simple fully connected network shows satisfactory results when the encoder has learned a good encoding. Its simplicity is also valuable when we mainly want to evaluate the learned representations of the encoder. However, the encoders limited regularization mechanisms make model tuning harder as validation results are somewhat unstable.

5.4 Further work

In this section, possible avenues of further research are discussed. Possible variations to the model that we considered while designing DECCaTNet, but did not test due to time constraints, are presented. The suggested future research covers the SSL pretext task, encoder and classifier architectures, and large-scale experiments.

5.4.1 Architecture

SSL architecture The main new contribution of DECCaTNet is the channel group encoder, but it would be beneficial to further test in what SSL framework it should be implemented. In our implementation we use a simple contrastive SSL architecture based on SimCLR [56], but CL architectures have developed further in the three years since SimCLR was published. Contrastive architectures more inspired by MoCo [105], SimSiam [106], BYOL [107], or the like should be explored further with a group-wise encoder.

Furthermore, the group-wise pre-training approach to multi-channel time series could work just as well, or even better, with a reconstructive pretext task, such as masking. Transformers have shown great learning capabilities using masking in NLP [4], so reconstructive pretext tasks warrant further exploration using a channel group encoder. Masking out random words has shown good results in NLP, but for EEG channel groups multiple other augmentations are applicable as well and would be interesting to implement and compare to our model. In subsection 2.5.3 we present multiple predictive pretext tasks that could be applied.

In related work section 2.6, the need for specific augmentation techniques for each CL based approach was clearly stated. To further enrich our work, an extensive ablation study on augmentations and selection techniques could be carried out. As our results indicate that the parameters used for this experiment are sufficient, exploring new selection techniques could be the way to go. We have shown that there is an unused pre-training potential in our framework, which is sensitive to augmentations. This potential could be leveraged in several ways. Having less randomization and a clearer distinction between the different augmentations like Eldele *et al.* [58] would be a possible first step.

Our CL architecture does not utilize any augmentations that capitalize on the signals having multiple channels. It is possible to further capitalize that for $n \geq 2$ the encoder works on multi-channel input by designing more specialized augmentations. Masking, which in our implementation masks out all channels in a given time band, could be adjusted to instead mask out different time bands in different channels independently, as done by Zerveas *et al.* [61]. Also, channel permutation, *i.e.* reordering the channels in a group, could be helpful. Another idea that could be interesting to explore would be to randomly select augmentation parameters independently for each channel, such as scaling each channel by a different amount or filtering out a different frequency band of each channel.

Grouping channels There are other options for how to divide channels into groups which we initially wanted to explore but had to abandon due to the narrowing of the scope of the project as the project developed. One way to group channels that immediately seems interesting is according to the exact physical location of the electrode.

As different datasets use different electrode placements, naming schemes, and only some include the coordinates of each electrode, we found it would be too time consuming to implement location-based groups. Some datasets have electrode coordinates stored in the 3D space (SEED), projected to the 2D space (BCICIV1), or coordinates could be derived from electrode names.

Given a standardized electrode coordinate scheme, channel groups could be created according to the exact location of the electrodes. It would be interesting to group channels that are close, to learn local relations, or on opposite sides of the head, to learn symmetric relations.

Encoder architecture Our encoder architecture is strongly inspired by the EEG Conformer [45], which achieved SOTA performance in 2022 using fully supervised learning and combining convolution and transformer modules. We did not test other encoder architectures or even major adjustments to the conformer model, such as the number of attention heads or the number of self-attention submodules. Other, simpler or more complex, encoder architectures should be tested.

As discussed in section 5.1, the output size of the encoder is independent of the number of channels n, making the number of trainable weights in the fine-tuning classifier proportional to the number of groups. If the output size of the encoder was made to depend on the number of channels, it could make the size of the classifier input less dependent on group size n. This change would, in most cases, increase the number of weights in the encoder and decrease the number of weights in the classifier and probably affect the optimal value for n.

For further development of a channel-group based encoder we suggest the encoder to have output shape equal to

$$(\text{input length}, \; kn)$$

where k is a small constant (Mohsenvand *et al.* used $k = 4$), and n is channel group size.

Classifier architecture We use a very simple fully connected network with only two hidden layers as our classifier. More complex classifier architectures can be tested, similar to SeqCLR which uses an LSTM based classifier. The classifier could be based on self-attention, CNN, LSTM or other more complex architectures.

For example, if the classifier was changed to a CNN-based module using convolution and pooling, it could make the classifier work on different signal lengths without the need to trim or window each signal to the exact same shape. The convolution kernels can learn the same features regardless of shape, and combined with pooling it could make much of the classifier architecture independent of input size. It could also allow a deeper network without increasing the number of trainable parameters.

5.4.2 Upscaling

A focus area while developing DECCaTNet was scalability to enable easy transfer learning and ability to learn on large datasets. In our experiments we used all the data in TUAB, SEED and BCICIV1, but only 8% of TUEG. In total, we trained our model on about 234 GB of data (see Figure 3.2), but if we were to use all of TUEG, and collect all available public EEG datasets we found during our research, about 2 TB of data could be collected. Our project was limited by training costs, but it would be interesting to see how more training data would affect the model, and if it would enable better results and allow the model to perform better using a deeper encoder. For clinical applications, it is possible to train the network on private data, making even more data available.

As our results show that increasing the pre-training dataset size helps, while training for more epochs does not, we believe that running experiments using more data for pre-training would be beneficial. The results indicate that learning from even more data would play to the strengths of DECCaTNet.

Other, more optimal, hyperparameter combinations could be found if a single, more exhaustive hyperparameter search was conducted. Our approach to approximate an optimal configuration was to use a semi-greedy search, optimizing one subset of parameters at a time. This approach is prone to finding local optima. With more computing power and time, a larger search could be performed that is more likely to find the globally optimal model configuration.

5.4.3 Transferability to other problems

In principle, the concept of the channel group SSL architecture could be applied to all multivariate time series. DECCaTNet is tailored for EEG, but its only EEG specific features are the neurologist-chosen augmentations and the preprocessing pipeline. The rest of the model architecture could be applied to any multivariate (or even univariate) time-series problem given a fitting set of augmentations and model parameters. Experiments in domains other than EEG should be conducted.

Chapter 6

Conclusion

This thesis has explored using a new self-supervised learning approach to EEG signal classification, with a transformer-based encoder, and focused on enabling flexible pre-training across multiple datasets. The objective of this thesis was to create a self-supervised EEG framework, capable of learning inter-channel dependencies and pre-training on a fusion of datasets with different numbers of channels. To accomplish the objective, we have introduced a new model, DECCaTNet, a contrastive, self-supervised learning architecture with a transformer-based encoder applied to a sub-group of n channels at a time.

Groups of size n can be extracted from any EEG recording for pre-training. For fine-tuning and classification, each recording is split into groups, each group is encoded, and finally the classification is done on the combined encodings. To our knowledge, in encoding channel groups of $n > 1$ channels and combining the encodings, DECCaTNet is the first of its kind.

Contributions

C1 A self-supervised learning model, called DECCaTNet, using contrastive learning to pre-train on groups of channels. The encoder of DECCaTNet is a hybrid CNN-transformer model. The model distinguishes itself by splitting each EEG signal into groups of n channels and encoding the channel groups separately.

C2 A preprocessing framework, built on MNE and Braindecode, capable of preprocessing large datasets from different sources, using serialization and limited parallelization. The preprocessing framework applies a standardized pipeline of rescaling, windowing, resampling, rereferencing, and filtering to all data, before splitting the samples into channel groups of size n. The framework can be easily extended to accommodate new datasets.

These are the main contributions of this master thesis. They were designed during our research project, which aimed to answer our research questions. The research questions are listed here, with short answers to each.

Research questions

RQ1 *How does pre-training an n-channel encoder perform, and how is it best implemented for EEG data?*

Our final classification accuracy on the TUH abnormal/normal test set was 84.26%. We split each EEG recording into as few groups as possible, while including all channels at least once.These groups are fed to the encoder, a CNN-transformer hybrid, which performed best if its weights were not frozen during fine-tuning. The validation results show that pre-training on a larger dataset is beneficial, while pre-training for more epochs after 10 did not lead to any improvements. Pre-training on a fusion of different EEG datasets showed improved results compared to only using the relevant dataset.

RQ2 *What is the optimal number of channels in each group when using SSL with grouped channels on EEG-data?*

Our experiments led to us choosing $n = 4$ channels per group as the optimal number for our configuration. However, as the evaluation accuracy using $n = 5$ was negligibly worse, we believe that with variations in the architecture or the dataset, other values for n may be optimal.

RQ3 *How does a transformer-based encoder perform in an SSL architecture for classifying multichannel EEG data?*

The transformer-based encoder performed well when implemented in our contrastive learning architecture. However, since the transformer-based encoder was not directly compared to other encoders in the same SSL architecture, we have not proven the answer to this question to a desirable degree.

A limitation of this work is the lack of testing against different encoders or classifiers using the same channel-group based framework. We have shown that many variations of DECCaTNet can perform well, but do not outperform the state-of-the-art accuracy of other studies. As this model includes innovations in several areas, it is difficult to determine exactly where its shortcomings lie, and we did not have the time required to perform an ablation study.

Another limitation is that we did not test our channel-grouping method in other SSL frameworks using different pretext tasks. As such, there could be better options than our simple contrastive framework. Additionally, the stability of our results suffers from not using cross-validation in the experiments.

Closing remarks As the reader may have observed, the Further work section is long and describes many different avenues of improvement. We have shown that the group-based SSL architecture holds potential, and believe that there are many different routes to success. The model showed greater learning potential from more epochs with a harder pretext task, suggesting that a goal of further improvement could be to make the self-supervised pretext task more difficult. This could be *e.g.* by changing the pretext task, augmentations, or window size. However, we hope DECCaTNet can help the field in the future, bringing a new idea to EEG classification and, we hope, multichannel time series in general.

Bibliography

[1] A. Biasiucci, B. Franceschiello, and M. M. Murray, "Electroencephalography," *Current Biology*, vol. 29, no. 3, R80–R85, Feb. 4, 2019, ISSN: 0960-9822. DOI: 10.1016/j.cub.2018.11.052. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960982218315513 (visited on 11/24/2022).

[2] Y. Roy, H. Banville, I. Albuquerque, A. Gramfort, T. H. Falk, and J. Faubert, "Deep learning-based electroencephalography analysis: A systematic review," *Journal of Neural Engineering*, vol. 16, no. 5, p. 051 001, Aug. 2019, Publisher: IOP Publishing, ISSN: 1741-2552. DOI: 10.1088/1741-2552/ab260c. [Online]. Available: https://dx.doi.org/10.1088/1741-2552/ab260c (visited on 11/22/2022).

[3] I. Obeid and J. Picone, "The temple university hospital EEG data corpus," *Frontiers in Neuroscience*, vol. 10, 2016, ISSN: 1662-453X. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2016.00196 (visited on 11/22/2022).

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (visited on 11/22/2022).

[5] D. Kostas, S. Aroca-Ouellette, and F. Rudzicz, "BENDR: Using transformers and a contrastive self-supervised learning task to learn from massive amounts of EEG data," *Frontiers in Human Neuroscience*, vol. 15, 2021, ISSN: 1662-5161. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnhum.2021.653659 (visited on 11/24/2022).

[6] M. N. Mohsenvand, M. R. Izadi, and P. Maes, "Contrastive representation learning for electroencephalogram classification," in *Proceedings of the Machine Learning for Health NeurIPS Workshop*, ISSN: 2640-3498, PMLR, Nov. 23, 2020, pp. 238–253. [Online]. Available: https://proceedings.mlr.press/v136/mohsenvand20a.html (visited on 11/24/2022).

[7] O. Størmer and S. Kamsvåg, *Transformers and self-supervised learning in EEG classification: A literature review*, Dec. 2022. [Online]. Available: `file:///C:/Users/Styrk/Downloads/Prosjektoppgave_Styrk_og_Oskar%20(3).pdf`.

[8] H. Banville, O. Chehab, A. Hyvärinen, D.-A. Engemann, and A. Gramfort, "Uncovering the structure of clinical EEG signals with self-supervised learning," *Journal of Neural Engineering*, vol. 18, no. 4, p. 046 020, Mar. 2021, Publisher: IOP Publishing, ISSN: 1741-2552. DOI: `10.1088/1741-2552/abca18`. [Online]. Available: `https://dx.doi.org/10.1088/1741-2552/abca18` (visited on 11/22/2022).

[9] J. Sun, J. Xie, and H. Zhou, "EEG classification with transformer-based models," in *2021 IEEE 3rd Global Conference on Life Sciences and Technologies (LifeTech)*, Mar. 2021, pp. 92–93. DOI: `10.1109/LifeTech52111.2021.9391844`.

[10] Z. Yue, Y. Wang, J. Duan, T. Yang, C. Huang, Y. Tong, and B. Xu, "TS2vec: Towards universal representation of time series," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, pp. 8980–8987, Jun. 28, 2022, Number: 8, ISSN: 2374-3468. DOI: `10.1609/aaai.v36i8.20881`. [Online]. Available: `https://ojs.aaai.org/index.php/AAAI/article/view/20881` (visited on 11/25/2022).

[11] M. Teplan, "FUNDAMENTALS OF EEG MEASUREMENT," *MEASUREMENT SCIENCE REVIEW*, vol. 2, p. 11, 2002.

[12] T. Zhang, W. Zheng, Z. Cui, Y. Zong, and Y. Li, "Spatial–temporal recurrent neural network for emotion recognition," *IEEE Transactions on Cybernetics*, vol. 49, no. 3, pp. 839–847, Mar. 2019, Conference Name: IEEE Transactions on Cybernetics, ISSN: 2168-2275. DOI: `10.1109/TCYB.2017.2788081`.

[13] W.-L. Zheng and B.-L. Lu, "Investigating critical frequency bands and channels for EEG-based emotion recognition with deep neural networks," *IEEE Transactions on Autonomous Mental Development*, vol. 7, no. 3, pp. 162–175, Sep. 2015, Conference Name: IEEE Transactions on Autonomous Mental Development, ISSN: 1943-0612. DOI: `10.1109/TAMD.2015.2431497`.

[14] A. Craik, Y. He, and J. L. Contreras-Vidal, "Deep learning for electroencephalogram (EEG) classification tasks: A review," *Journal of Neural Engineering*, vol. 16, no. 3, p. 031 001, Apr. 2019, Publisher: IOP Publishing, ISSN: 1741-2552. DOI: `10.1088/1741-2552/ab0ab5`. [Online]. Available: `https://dx.doi.org/10.1088/1741-2552/ab0ab5` (visited on 11/24/2022).

[15] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. Siegelbaum, A. J. Hudspeth, S. Mack, *et al.*, *Principles of neural science*. McGraw-hill New York, 2000, vol. 4.

[16] N. Bigdely-Shamlo, T. Mullen, C. Kothe, K.-M. Su, and K. A. Robbins, “The PREP pipeline: Standardized preprocessing for large-scale EEG analysis,” *Frontiers in Neuroinformatics*, vol. 9, 2015, ISSN: 1662-5196. [Online]. Available: `https://www.frontiersin.org/articles/10.3389/fninf.2015.00016` (visited on 11/25/2022).

[17] M. Jas, D. A. Engemann, Y. Bekhti, F. Raimondo, and A. Gramfort, “Autoreject: Automated artifact rejection for MEG and EEG data,” *NeuroImage*, vol. 159, pp. 417–429, Oct. 1, 2017, ISSN: 1053-8119. DOI: `10.1016/j.neuroimage.2017.06.030`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1053811917305013` (visited on 11/24/2022).

[18] S. Cole and B. Voytek, “Cycle-by-cycle analysis of neural oscillations,” *Journal of Neurophysiology*, vol. 122, no. 2, pp. 849–861, Aug. 2019, Publisher: American Physiological Society, ISSN: 0022-3077. DOI: `10.1152/jn.00273.2019`. [Online]. Available: `https://journals.physiology.org/doi/full/10.1152/jn.00273.2019` (visited on 11/24/2022).

[19] A. Gramfort, D. Strohmeier, J. Haueisen, M. S. Hämäläinen, and M. Kowalski, “Time-frequency mixed-norm estimates: Sparse m/EEG imaging with non-stationary source activations,” *NeuroImage*, vol. 70, pp. 410–422, Apr. 15, 2013, ISSN: 1053-8119. DOI: `10.1016/j.neuroimage.2012.12.051`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1053811912012372` (visited on 11/25/2022).

[20] M. Clerc, L. Bougrain, and F. Lotte, *Brain-Computer Interfaces 1: Methods and Perspectives*. John Wiley & Sons, Jul. 14, 2016, 335 pp., Google-Books-ID: tjcZDQAAQBAJ, ISBN: 978-1-119-14498-4.

[21] O. Özdenizci, Y. Wang, T. Koike-Akino, and D. Erdoğmuş, “Learning invariant representations from EEG via adversarial inference,” *IEEE Access*, vol. 8, pp. 27 074–27 085, 2020, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: `10.1109/ACCESS.2020.2971600`.

[22] P. M. Rossini, S. Rossi, C. Babiloni, and J. Polich, “Clinical neurophysiology of aging brain: From normal aging to neurodegeneration,” *Progress in Neurobiology*, vol. 83, no. 6, pp. 375–400, Dec. 1, 2007, ISSN: 0301-0082. DOI: `10.1016/j.pneurobio.2007.07.010`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0301008207001451` (visited on 11/24/2022).

[23] G.-Q. Zhang, L. Cui, R. Mueller, S. Tao, M. Kim, M. Rueschman, S. Mariani, D. Mobley, and S. Redline, “The national sleep research resource: Towards a sleep data commons,” *Journal of the American Medical Informatics Association: JAMIA*, vol. 25, no. 10, pp. 1351–1358, Oct. 1, 2018, ISSN: 1527-974X. DOI: `10.1093/jamia/ocy064`.

[24] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, Number: 7553 Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/nature14539. [Online]. Available: https://www.nature.com/articles/nature14539 (visited on 11/24/2022).

[25] K. O'Shea and R. Nash, *An introduction to convolutional neural networks*, Dec. 2, 2015. DOI: 10.48550/arXiv.1511.08458. arXiv: 1511.08458[cs]. [Online]. Available: http://arxiv.org/abs/1511.08458 (visited on 11/24/2022).

[26] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, May 1, 2018, ISSN: 0031-3203. DOI: 10.1016/j.patcog.2017.10.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320317304120 (visited on 11/25/2022).

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html (visited on 11/23/2022).

[28] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-ResNet and the impact of residual connections on learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, Feb. 12, 2017, Number: 1, ISSN: 2374-3468. DOI: 10.1609/aaai.v31i1.11231. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11231 (visited on 11/23/2022).

[29] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html (visited on 11/24/2022).

[30] K. Chen, J. Wang, L.-C. Chen, H. Gao, W. Xu, and R. Nevatia, *ABC-CNN: An attention based convolutional neural network for visual question answering*, Apr. 3, 2016. DOI: 10.48550/arXiv.1511.05960. arXiv: 1511.05960[cs]. [Online]. Available: http://arxiv.org/abs/1511.05960 (visited on 11/24/2022).

[31] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, May 19, 2016. DOI: 10.48550/arXiv.1409.0473. arXiv: 1409.0473[cs,stat]. [Online]. Available: http://arxiv.org/abs/1409.0473 (visited on 11/25/2022).

[32] Y. Kim, C. Denton, L. Hoang, and A. M. Rush, *Structured attention networks*, Feb. 16, 2017. DOI: 10.48550/arXiv.1702.00887. arXiv: 1702.00887[cs]. [Online]. Available: http://arxiv.org/abs/1702.00887 (visited on 11/25/2022).

[33] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, *Megatron-LM: Training multi-billion parameter language models using model parallelism*, Mar. 13, 2020. DOI: 10.48550/arXiv.1909.08053. arXiv: 1909.08053[cs]. [Online]. Available: http://arxiv.org/abs/1909.08053 (visited on 11/25/2022).

[34] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu, *CoCa: Contrastive captioners are image-text foundation models*, Jun. 13, 2022. DOI: 10.48550/arXiv.2205.01917. arXiv: 2205.01917[cs]. [Online]. Available: http://arxiv.org/abs/2205.01917 (visited on 11/24/2022).

[35] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html (visited on 05/23/2023).

[36] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun, *Transformers in time series: A survey*, Mar. 7, 2022. DOI: 10.48550/arXiv.2202.07125. arXiv: 2202.07125[cs,eess,stat]. [Online]. Available: http://arxiv.org/abs/2202.07125 (visited on 11/25/2022).

[37] L. Lu, L. Kong, C. Dyer, N. A. Smith, and S. Renals, *Segmental recurrent neural networks for end-to-end speech recognition*, Jun. 20, 2016. DOI: 10.48550/arXiv.1603.00223. arXiv: 1603.00223[cs]. [Online]. Available: http://arxiv.org/abs/1603.00223 (visited on 11/22/2022).

[38] M.-A. Moinnereau, T. Brienne, S. Brodeur, J. Rouat, K. Whittingstall, and E. Plourde, *Classification of auditory stimuli from EEG signals with a regulated recurrent neural network reservoir*, Apr. 26, 2018. DOI: 10.48550/arXiv.1804.10322. arXiv: 1804.10322[cs,eess]. [Online]. Available: http://arxiv.org/abs/1804.10322 (visited on 11/22/2022).

[39] G. Ruffini, D. Ibañez, M. Castellano, S. Dunne, and A. Soria-Frisch, "EEG-driven RNN classification for prognosis of neurodegeneration in at-risk patients," in *Artificial Neural Networks and Machine Learning – ICANN 2016*, A. E. Villa, P. Masulli, and A. J. Pons Rivero, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing,

2016, pp. 306–313, ISBN: 978-3-319-44778-0. DOI: 10.1007/978-3-319-44778-0_36.

[40] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "Wav2vec 2.0: A framework for self-supervised learning of speech representations," in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 12 449–12 460. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/92d1e1eb1cd6f9fba3227870bb6d7f07-Abstract.html (visited on 11/22/2022).

[41] Y. Wu, C. Lian, Z. Zeng, B. Xu, and Y. Su, "An aggregated convolutional transformer based on slices and channels for multivariate time series classification," *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 1–12, 2022, Conference Name: IEEE Transactions on Emerging Topics in Computational Intelligence, ISSN: 2471-285X. DOI: 10.1109/TETCI.2022.3210992.

[42] J. Xie, J. Zhang, J. Sun, Z. Ma, L. Qin, G. Li, H. Zhou, and Y. Zhan, "A transformer-based approach combining deep learning network and spatial-temporal information for raw EEG classification," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 30, pp. 2126–2136, 2022, Conference Name: IEEE Transactions on Neural Systems and Rehabilitation Engineering, ISSN: 1558-0210. DOI: 10.1109/TNSRE.2022.3194600.

[43] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, *Squeeze-and-excitation networks*, May 16, 2019. DOI: 10.48550/arXiv.1709.01507. arXiv: 1709.01507[cs]. [Online]. Available: http://arxiv.org/abs/1709.01507 (visited on 11/24/2022).

[44] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, *Conformer: Convolution-augmented transformer for speech recognition*, May 16, 2020. DOI: 10.48550/arXiv.2005.08100. arXiv: 2005.08100[cs,eess]. [Online]. Available: http://arxiv.org/abs/2005.08100 (visited on 05/03/2023).

[45] Y. Song, Q. Zheng, B. Liu, and X. Gao, "EEG conformer: Convolutional transformer for EEG decoding and visualization," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 31, pp. 710–719, 2023, Conference Name: IEEE Transactions on Neural Systems and Rehabilitation Engineering, ISSN: 1558-0210. DOI: 10.1109/TNSRE.2022.3230250.

[46] P. Goyal, D. Mahajan, A. Gupta, and I. Misra, *Scaling and benchmarking self-supervised visual representation learning*, Jun. 6, 2019. DOI: 10.48550/arXiv.1905.01235. arXiv: 1905.01235[cs]. [Online]. Available: http://arxiv.org/abs/1905.01235 (visited on 05/03/2023).

[47] D. Spathis, I. Perez-Pozuelo, L. Marques-Fernandez, and C. Mascolo, "Breaking away from labels: The promise of self-supervised machine learning in intelligent health," *Patterns*, vol. 3, no. 2, p. 100 410, Feb. 11, 2022, ISSN: 2666-3899. DOI: 10.1016/j.patter.2021.100410. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666389921002841 (visited on 11/24/2022).

[48] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 16 000–16 009. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2022/html/He_Masked_Autoencoders_Are_Scalable_Vision_Learners_CVPR_2022_paper.html (visited on 11/24/2022).

[49] C. I. Tang, I. Perez-Pozuelo, D. Spathis, and C. Mascolo, *Exploring contrastive learning in human activity recognition for healthcare*, Feb. 11, 2021. DOI: 10.48550/arXiv.2011.11542. arXiv: 2011.11542[cs,eess]. [Online]. Available: http://arxiv.org/abs/2011.11542 (visited on 11/24/2022).

[50] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon, "A survey on contrastive self-supervised learning," *Technologies*, vol. 9, no. 1, p. 2, Mar. 2021, Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2227-7080. DOI: 10.3390/technologies9010002. [Online]. Available: https://www.mdpi.com/2227-7080/9/1/2 (visited on 11/24/2022).

[51] N. Wagh, J. Wei, S. Rawal, B. Berry, L. Barnard, B. Brinkmann, G. Worrell, D. Jones, and Y. Varatharajah, "Domain-guided self-supervision of EEG data improves downstream classification performance and generalizability," in *Proceedings of Machine Learning for Health*, ISSN: 2640-3498, PMLR, Nov. 28, 2021, pp. 130–142. [Online]. Available: https://proceedings.mlr.press/v158/wagh21a.html (visited on 11/24/2022).

[52] R. Tibor Schirrmeister, L. Gemein, K. Eggensperger, F. Hutter, and T. Ball, "Deep learning with convolutional neural networks for decoding and visualization of EEG pathology," Aug. 1, 2017, Publication Title: arXiv e-prints ADS Bibcode: 2017arXiv170808012T Type: article. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2017arXiv170808012T (visited on 11/24/2022).

[53] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," *Journal of Machine Learning Research*, vol. 10, no. 9, pp. 207–244, 2009, ISSN: 1533-7928. [Online]. Available: http://jmlr.org/papers/v10/weinberger09a.html (visited on 11/24/2022).

[54] X. Jiang, J. Zhao, B. Du, and Z. Yuan, "Self-supervised contrastive learning for EEG-based sleep staging," in *2021 International Joint Conference on Neural Networks (IJCNN)*, ISSN: 2161-4407, Jul. 2021, pp. 1–8. DOI: `10.1109/IJCNN52387.2021.9533305`.

[55] X. Zhang, Z. Zhao, T. Tsiligkaridis, and M. Zitnik, *Self-supervised contrastive pre-training for time series via time-frequency consistency*, Oct. 15, 2022. DOI: `10.48550/arXiv.2206.08496`. arXiv: `2206.08496[cs]`. [Online]. Available: `http://arxiv.org/abs/2206.08496` (visited on 11/25/2022).

[56] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, Nov. 21, 2020, pp. 1597–1607. [Online]. Available: `https://proceedings.mlr.press/v119/chen20j.html` (visited on 11/24/2022).

[57] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 18 661–18 673. [Online]. Available: `https://proceedings.neurips.cc/paper/2020/hash/d89a66c7c80a29b1bdbab0f2a1a94af8-Abstract.html` (visited on 11/25/2022).

[58] E. Eldele, M. Ragab, Z. Chen, M. Wu, C.-K. Kwoh, X. Li, and C. Guan, *Self-supervised contrastive representation learning for semi-supervised time-series classification*, Aug. 13, 2022. DOI: `10.48550/arXiv.2208.06616`. arXiv: `2208.06616[cs]`. [Online]. Available: `http://arxiv.org/abs/2208.06616` (visited on 11/25/2022).

[59] W. Ko and H.-I. Suk, "EEG-oriented self-supervised learning and cluster-aware adaptation," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, ser. CIKM '22, New York, NY, USA: Association for Computing Machinery, Oct. 17, 2022, pp. 4143–4147, ISBN: 978-1-4503-9236-5. DOI: `10.1145/3511808.3557589`. [Online]. Available: `https://doi.org/10.1145/3511808.3557589` (visited on 11/24/2022).

[60] S. Tang, J. Dunnmon, K. K. Saab, X. Zhang, Q. Huang, F. Dubost, D. Rubin, and C. Lee-Messer, "Self-supervised graph neural networks for improved electroencephalographic seizure analysis," presented at the International Conference on Learning Representations, Feb. 27, 2022. [Online]. Available: `https://openreview.net/forum?id=k9bx1EfHI_-` (visited on 12/12/2022).

[61] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, "A transformer-based framework for multivariate time series representation learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21, New York, NY, USA: Associa-

tion for Computing Machinery, Aug. 14, 2021, pp. 2114–2124, ISBN: 978-1-4503-8332-5. DOI: 10.1145/3447548.3467401. [Online]. Available: https://doi.org/10.1145/3447548.3467401 (visited on 11/22/2022).

[62] H. Banville, I. Albuquerque, A. Hyvärinen, G. Moffat, D.-A. Engemann, and A. Gramfort, "Self-supervised representation learning from electroencephalography signals," in *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, ISSN: 1551-2541, Oct. 2019, pp. 1–6. DOI: 10.1109/MLSP.2019.8918693.

[63] H.-I. Suk and S.-W. Lee, "A novel bayesian framework for discriminative feature extraction in brain-computer interfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 286–299, Feb. 2013, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2012.69.

[64] M. Perslev, M. Jensen, S. Darkner, P. J. r. Jennum, and C. Igel, "U-time: A fully convolutional network for time series segmentation applied to sleep staging," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/57bafb2c2dfeefba931bb03a835b1fa9-Abstract.html (visited on 12/12/2022).

[65] H. Phan, O. Y. Chén, M. C. Tran, P. Koch, A. Mertins, and M. De Vos, "XSleepNet: Multi-view sequential model for automatic sleep staging," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5903–5915, Sep. 2022, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2021.3070057.

[66] S. Das, P. Pandey, and K. P. Miyapuram, *Improving self-supervised pretraining models for epileptic seizure detection from EEG data*, Jun. 28, 2022. DOI: 10.48550/arXiv.2207.06911. arXiv: 2207.06911[cs,eess]. [Online]. Available: http://arxiv.org/abs/2207.06911 (visited on 11/29/2022).

[67] J. Y. Cheng, H. Goh, K. Dogrusoz, O. Tuzel, and E. Azemi, *Subject-aware contrastive learning for biosignals*, Jun. 30, 2020. DOI: 10.48550/arXiv.2007.04871. arXiv: 2007.04871[cs,eess,stat]. [Online]. Available: http://arxiv.org/abs/2007.04871 (visited on 11/29/2022).

[68] S. Ferrell, V. Mathew, M. Refford, V. Tchiong, T. Ahsan, I. Obeid, and J. Picone, "The temple university hospital EEG corpus: Electrode location and channel labels," *Institute for Signal and Information Processing Report*, vol. 1, no. 1, Apr. 2020. [Online]. Available: https://par.nsf.gov/biblio/10199699-temple-university-hospital-eeg-corpus-electrode-location-channel-labels (visited on 12/06/2022).

[69] L. d. Diego and S. Isabel, “Automated interpretation of abnormal adult electroencephalograms,” 2017, Accepted: 2020-10-27T15:14:14Z Publisher: Temple University. Libraries. [Online]. Available: https://scholarshare.temple.edu/handle/20.500.12613/1767 (visited on 05/11/2023).

[70] R.-N. Duan, J.-Y. Zhu, and B.-L. Lu, “Differential entropy feature for EEG-based emotion classification,” in *2013 6th International IEEE/EMBS Conference on Neural Engineering (NER)*, ISSN: 1948-3554, Nov. 2013, pp. 81–84. DOI: 10.1109/NER.2013.6695876.

[71] T. Song, W. Zheng, P. Song, and Z. Cui, “EEG emotion recognition using dynamical graph convolutional neural networks,” *IEEE Transactions on Affective Computing*, vol. 11, no. 3, pp. 532–541, Jul. 1, 2020, Publisher: IEEE Computer Society, ISSN: 1949-3045. DOI: 10.1109/TAFFC.2018.2817622. [Online]. Available: https://www.computer.org/csdl/journal/ta/2020/03/08320798/13rRUyYjK8G (visited on 04/24/2023).

[72] S. Roy, I. Kiral-Kornek, and S. Harrer, “ChronoNet: A deep recurrent neural network for abnormal EEG identification,” in *Artificial Intelligence in Medicine*, D. Riaño, S. Wilk, and A. ten Teije, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2019, pp. 47–56, ISBN: 978-3-030-21642-9. DOI: 10.1007/978-3-030-21642-9_8.

[73] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, “Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state,” *Physical Review E*, vol. 64, no. 6, p. 061 907, Nov. 20, 2001, Publisher: American Physical Society. DOI: 10.1103/PhysRevE.64.061907. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.64.061907 (visited on 12/12/2022).

[74] R. Hefron, B. Borghetti, C. Schubert Kabban, J. Christensen, and J. Estepp, “Cross-participant EEG-based assessment of cognitive workload using multi-path convolutional recurrent neural networks,” *Sensors*, vol. 18, no. 5, p. 1339, May 2018, Number: 5 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1424-8220. DOI: 10.3390/s18051339. [Online]. Available: https://www.mdpi.com/1424-8220/18/5/1339 (visited on 04/24/2023).

[75] S. Stober, A. Sternin, A. M. Owen, and J. A. Grahn, *Deep feature learning for EEG recordings*, Jan. 7, 2016. DOI: 10.48550/arXiv.1511.04306. arXiv: 1511.04306[cs]. [Online]. Available: http://arxiv.org/abs/1511.04306 (visited on 11/24/2022).

[76] S. Leske and S. S. Dalal, “Reducing power line noise in EEG and MEG data via spectrum interpolation,” *NeuroImage*, vol. 189, pp. 763–776, Apr. 1, 2019, ISSN: 1053-8119. DOI: 10.1016/j.neuroimage.2019.01.026. [On-

line]. Available: https://www.sciencedirect.com/science/article/pii/S1053811919300266 (visited on 11/25/2022).

[77] R. N. Bracewell and R. N. Bracewell, *The Fourier transform and its applications*. McGraw-Hill New York, 1986, vol. 31999.

[78] P. Sarkar and A. Etemad, "Self-supervised ECG representation learning for emotion recognition," *IEEE Transactions on Affective Computing*, vol. 13, no. 3, pp. 1541–1554, Jul. 2022, Conference Name: IEEE Transactions on Affective Computing, ISSN: 1949-3045. DOI: 10.1109/TAFFC.2020.3014842.

[79] W. Qu, Z. Wang, H. Hong, Z. Chi, D. D. Feng, R. Grunstein, and C. Gordon, "A residual based attention model for EEG based sleep staging," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 10, pp. 2833–2843, Oct. 2020, Conference Name: IEEE Journal of Biomedical and Health Informatics, ISSN: 2168-2208. DOI: 10.1109/JBHI.2020.2978004.

[80] M. Alhussein, G. Muhammad, and M. S. Hossain, "EEG pathology detection based on deep learning," *IEEE Access*, vol. 7, pp. 27 781–27 788, 2019, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2901672.

[81] Y. Zhao, C. Dong, G. Zhang, Y. Wang, X. Chen, W. Jia, Q. Yuan, F. Xu, and Y. Zheng, "EEG-based seizure detection using linear graph convolution network with focal loss," *Computer Methods and Programs in Biomedicine*, vol. 208, p. 106 277, Sep. 1, 2021, ISSN: 0169-2607. DOI: 10.1016/j.cmpb.2021.106277. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169260721003515 (visited on 12/07/2022).

[82] S. U. Amin, M. S. Hossain, G. Muhammad, M. Alhussein, and M. A. Rahman, "Cognitive smart healthcare for pathology detection and monitoring," *IEEE Access*, vol. 7, pp. 10 745–10 753, 2019, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2891390.

[83] G. Garcia-Molina, "Direct brain-computer communication through scalp recorded EEG signals," Jan. 1, 2004. DOI: 10.5075/epfl-thesis-3019.

[84] L. V. Marcuse, M. C. Fields, and J. Y. J. Yoo, *Rowan's Primer of EEG E-Book*. Elsevier Health Sciences, Sep. 22, 2015, 280 pp., Google-Books-ID: 62zdCgAAQBAJ, ISBN: 978-0-323-35867-5.

[85] D. Yao, "A method to standardize a reference of scalp EEG recordings to a point at infinity," *Physiological Measurement*, vol. 22, no. 4, p. 693, Oct. 2001, ISSN: 0967-3334. DOI: 10.1088/0967-3334/22/4/305. [Online]. Available: https://dx.doi.org/10.1088/0967-3334/22/4/305 (visited on 05/11/2023).

[86] B. Yang, K. Duan, C. Fan, C. Hu, and J. Wang, "Automatic ocular artifacts removal in EEG using deep learning," *Biomedical Signal Processing and Control*, vol. 43, pp. 148–158, May 1, 2018, ISSN: 1746-8094. DOI: 10.1016/j.bspc.2018.02.021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1746809418300521 (visited on 11/24/2022).

[87] R. Manor and A. B. Geva, "Convolutional neural network for multi-category rapid serial visual presentation BCI," *Frontiers in Computational Neuroscience*, vol. 9, 2015, ISSN: 1662-5188. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fncom.2015.00146 (visited on 11/24/2022).

[88] F. Thabtah, S. Hammoud, F. Kamalov, and A. Gonsalves, "Data imbalance in classification: Experimental evaluation," *Information Sciences*, vol. 513, pp. 429–441, Mar. 1, 2020, ISSN: 0020-0255. DOI: 10.1016/j.ins.2019.11.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025519310497 (visited on 12/06/2022).

[89] D. L. Olson, "Data set balancing," in *Data Mining and Knowledge Management*, Y. Shi, W. Xu, and Z. Chen, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2005, pp. 71–80, ISBN: 978-3-540-30537-8. DOI: 10.1007/978-3-540-30537-8_8.

[90] C. X. Ling and C. Li, "Data mining for direct marketing: Problems and solutions," p. 7, 1998.

[91] V. Gorade, A. Singh, and D. Mishra, *Large scale time-series representation learning via simultaneous low and high frequency feature bootstrapping*, Apr. 24, 2022. DOI: 10.48550/arXiv.2204.11291. arXiv: 2204.11291[cs]. [Online]. Available: http://arxiv.org/abs/2204.11291 (visited on 11/28/2022).

[92] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet," *Circulation*, vol. 101, no. 23, e215–e220, Jun. 13, 2000, Publisher: American Heart Association. DOI: 10.1161/01.CIR.101.23.e215. [Online]. Available: https://www.ahajournals.org/doi/full/10.1161/01.CIR.101.23.e215 (visited on 12/12/2022).

[93] B. Blankertz, G. Dornhege, M. Krauledat, K.-R. Müller, and G. Curio, "The non-invasive berlin brain–computer interface: Fast acquisition of effective performance in untrained subjects," *NeuroImage*, vol. 37, no. 2, pp. 539–550, Aug. 15, 2007, ISSN: 1053-8119. DOI: 10.1016/j.neuroimage.2007.01.051. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811907000535 (visited on 04/25/2023).

[94] E. Niedermeyer and F. H. L. d. Silva, *Electroencephalography: Basic Principles, Clinical Applications, and Related Fields*. Lippincott Williams & Wilkins, 2005, 1342 pp., Google-Books-ID: tndqYGPHQdEC, ISBN: 978-0-7817-5126-1.

[95] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An imperative style, high-performance deep learning library*, Dec. 3, 2019. DOI: 10.48550/arXiv.1912.01703. arXiv: 1912.01703[cs,stat]. [Online]. Available: http://arxiv.org/abs/1912.01703 (visited on 05/10/2023).

[96] Z. Brown, *SSL baselines for biosignal feature extraction*. [Online]. Available: https://github.com/zacharycbrown/ssl_baselines_for_biosignal_feature_extraction.

[97] X. Zhang, L. Yao, M. Dong, Z. Liu, Y. Zhang, and Y. Li, "Adversarial representation learning for robust patient-independent epileptic seizure detection," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 10, pp. 2852–2859, Oct. 2020, Conference Name: IEEE Journal of Biomedical and Health Informatics, ISSN: 2168-2208. DOI: 10.1109/JBHI.2020.2971610.

[98] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball, "Deep learning with convolutional neural networks for EEG decoding and visualization," *Human Brain Mapping*, vol. 38, no. 11, pp. 5391–5420, 2017, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/hbm.23730, ISSN: 1097-0193. DOI: 10.1002/hbm.23730. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/hbm.23730 (visited on 05/10/2023).

[99] A. Gramfort, M. Luessi, E. Larson, D. Engemann, D. Strohmeier, C. Brodbeck, R. Goj, M. Jas, T. Brooks, L. Parkkonen, and M. Hämäläinen, "MEG and EEG data analysis with MNE-python," *Frontiers in Neuroscience*, vol. 7, 2013, ISSN: 1662-453X. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2013.00267 (visited on 05/11/2023).

[100] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu, "Time series data augmentation for deep learning: A survey," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, Aug. 2021, pp. 4653–4660. DOI: 10.24963/ijcai.2021/631. arXiv: 2002.12478[cs,eess,stat]. [Online]. Available: http://arxiv.org/abs/2002.12478 (visited on 04/25/2023).

[101] A. v. d. Oord, Y. Li, and O. Vinyals, *Representation learning with contrastive predictive coding*, Jan. 22, 2019. DOI: 10.48550/arXiv.1807.03748. arXiv:

1807.03748[cs, stat]. [Online]. Available: http://arxiv.org/abs/1807.03748 (visited on 12/12/2022).

[102] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, *Tune: A research platform for distributed model selection and training*, Jul. 13, 2018. DOI: 10.48550/arXiv.1807.05118. arXiv: 1807.05118[cs, stat]. [Online]. Available: http://arxiv.org/abs/1807.05118 (visited on 05/18/2023).

[103] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018, ISSN: 1533-7928. [Online]. Available: http://jmlr.org/papers/v18/16-558.html (visited on 06/05/2023).

[104] S. Bagchi and D. R. Bathula, "EEG-ConvTransformer for single-trial EEG-based visual stimulus classification," *Pattern Recognition*, vol. 129, p. 108 757, Sep. 1, 2022, ISSN: 0031-3203. DOI: 10.1016/j.patcog.2022.108757. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320322002382 (visited on 11/29/2022).

[105] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 9729–9738. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2020/html/He_Momentum_Contrast_for_Unsupervised_Visual_Representation_Learning_CVPR_2020_paper.html (visited on 05/26/2023).

[106] X. Chen and K. He, *Exploring simple siamese representation learning*, Nov. 20, 2020. DOI: 10.48550/arXiv.2011.10566. arXiv: 2011.10566[cs]. [Online]. Available: http://arxiv.org/abs/2011.10566 (visited on 05/26/2023).

[107] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu koray, R. Munos, and M. Valko, "Bootstrap your own latent - a new approach to self-supervised learning," in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 21 271–21 284. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/f3ada80d5c4ee70142b17b8192b2958e-Abstract.html (visited on 05/26/2023).

Appendix A

Experimental results

A.1 Hyperparameter search

LR	FC 1	FC 2	WD	Val. loss	Train loss	Val. acc.	Epochs
0.00212	256	32	0.0854	0.0225	0.0193	0.553	1
0.01132	256	32	0.000244	0.0265	0.0194	0.502	1
0.00355	256	32	0.0517	0.0171	0.0178	0.684	1
0.09083	32	32	0.347	0.0217	0.0218	0.502	1
0.02328	256	32	0.00774	0.0204	0.014	0.657	2
1.234e-05	256	32	0.000796	0.0138	0.0131	0.785	4
2.724e-05	32	4	0.0487	0.0176	0.0187	0.772	2
0.0085	256	32	0.000187	0.0150	0.0110	0.783	16
0.00294	32	4	0.187	0.0229	0.0231	0.544	1
5.519e-05	32	4	0.175	0.0217	0.0217	0.507	1
6.712e-05	32	4	0.184	0.0233	0.0241	0.487	1
0.00128	32	4	0.00294	0.0176	0.0194	0.717	1
0.01622	32	4	0.00379	0.0196	0.0209	0.676	1
1.586e-05	256	32	0.00010	0.0141	0.0124	0.802	8
1.453e-05	256	32	0.000308	0.0186	0.0146	0.723	2
2.915e-05	256	32	0.000991	0.0141	0.0145	0.773	2
0.009622	32	32	0.071	0.0198	0.0209	0.668	1
1.219e-05	256	32	0.000185	0.0127	0.0108	0.821	16
0.00219	32	32	0.00102	0.0132	0.0140	0.812	4
0.0399	32	32	0.0539	0.0180	0.0205	0.686	2

Table A.1: Initial fine-tune net hyperparameter search results. The lowest texted learning rate did the best, further search is needed to find optimum.
LR: Learning rate, FC: Fully connected layer size, WD: Weight decay

BS	**LR**	τ	**WD**	**Val. loss**	**Train loss**	**Val. acc.**	**Epochs**
64	0.00860	0.0104	0.55196	nan	nan	0.5063	1
128	0.0186	0.0355	0.12478	0.0217	0.0217	0.4987	1
128	0.00012	0.0408	0.05328	0.0166	0.0166	0.7413	8
64	0.00036	0.00108	0.00175	nan	nan	0.5111	1
256	0.00030	0.00460	0.00109	nan	nan	0.5155	2
64	0.0403	0.00972	0.9813	nan	nan	0.5037	1
128	5.55e-05	0.177	0.000286	0.0164	0.0152	0.7348	10
64	0.00584	0.0939	0.002674	0.0217	0.0216	0.4957	2
64	0.00756	0.0266	0.1158	0.0217	0.0217	0.5059	2
128	0.0921	0.0408	0.002749	0.0217	0.0217	0.4959	1
128	0.00858	0.314	0.1324	0.0217	0.0216	0.4910	2
64	0.00328	0.440	0.04153	0.0223	0.0171	0.6424	4
128	1.22e-05	0.327	0.003686	0.0156	0.0143	0.7513	10
64	1.77e-05	0.104	0.373	0.0185	0.0182	0.6966	2
64	1.82e-05	0.562	0.003793	0.0173	0.0169	0.7140	4

Table A.2: First pre-training hyperparameter search results. The lowest tested learning rate did the best, further search is needed to find optimum.
BS: Batch size, LR: Learning rate, τ: Temperature, WD: Weight decay

BS	LR	PT epochs	Val. loss	Train loss	Val. acc.	FT epochs
256	2.089e-06	10	0.01687	0.01738	0.7231	1
128	3.636e-07	2	0.01783	0.01830	0.7118	1
128	1.658e-05	2	0.01669	0.01556	0.7383	8
128	4.503e-06	10	0.01683	0.01638	0.7328	4
128	5.384e-07	10	0.01698	0.01708	0.7320	2
128	1.503e-05	2	0.01807	0.01789	0.6895	1
128	2.396e-06	10	0.01448	0.01328	0.7659	10
256	1.175e-06	4	0.01656	0.01547	0.7369	10
128	2.508e-07	4	0.01824	0.01776	0.6956	2
128	1.468e-06	2	0.01709	0.01759	0.7190	1
128	2.328e-06	10	0.01715	0.01679	0.7285	2
256	3.568e-06	4	0.01664	0.01630	0.7374	4
256	2.711e-06	10	0.01672	0.01693	0.7395	2
128	5.512e-07	2	0.01807	0.01730	0.6958	2
128	7.091e-06	4	0.01741	0.01795	0.7166	1
512	8.011e-07	10	-	-	-	-
512	4.321e-07	4	-	-	-	-
512	1.816e-05	2	-	-	-	-
512	7.589e-06	10	-	-	-	-
512	3.282e-06	10	-	-	-	-

Table A.3: Second pre-training hyperparameter search results. Adjusted after the first search with a larger search space for learning rate, batch size and number of epochs. All runs with batch size 512 overfilled GPU memory and failed.
BS: Batch size, LR: Learning rate, PT: Pre-training, FT: Fine-tuning

A.2 Optimal number of channels

n	Val. loss	Train loss	Val. acc.	Epochs
1	0.01674	0.01493	0.7263	11
2	0.01685	0.01615	0.7422	11
3	0.01722	0.01623	0.7294	10
5	0.01478	0.01422	0.7747	11
8	0.01681	0.01537	0.7324	10
12	0.02181	0.01524	0.6466	10
23	0.01753	0.01651	0.7176	10

Table A.4: Results of initial hyperparameter search for optimal n channels. As $n = 5$ showed best results, another search was conducted.

n	**Val. loss**	**Train loss**	**Val. acc.**	**Epochs**
4	0.01596	0.01383	0.7794	10
5	0.01646	0.01533	0.7618	10
6	0.01590	0.01572	0.7547	10
7	0.01640	0.01504	0.7339	10

Table A.5: Results of second hyperparameter search for optimal n channels. $n = 4$ shows best results by a significant margin.

A.3 Frozen Pre-trained encoders

Frozen encoder	Pre-trained encoder	**Val. loss**	**Train loss**	**Val. acc.**	**Epochs**
False	True	0.01171	0.01018	0.8325	11
True	True	0.01405	0.01381	0.7911	11
False	False	0.01284	0.01192	0.8186	11
True	False	0.02358	0.02171	0.6137	10

Table A.6: Results of frozen encoder weights and random initialized encoder, where the pre-trained encoder with unfrozen weights outperform all randomly initialized encoders with unfrozen weights.

A.4 Number of pre-training epochs

Number of pre-training epochs	**Val. loss**	**Train loss**	**val. acc.**	**Fine-tuning epochs**
10	0.01363	0.01346	0.8151	11
30	0.01336	0.01396	0.8060	11
50	0.01405	0.01381	0.8011	11
70	0.01428	0.01453	0.7952	11
100	0.01427	0.01453	0.7950	10

Table A.7: Results of number of pre-training epochs.

A.5 Transfer learning

Frozen encoder	Pre-trained encoder	**Val. loss**	**Train loss**	**Val. acc.**	**Epochs**
False	True	0.01252	0.00971	0.8232	11
True	True	0.01587	0.01392	0.7765	11
False	False	0.01224	0.01171	0.8201	11
True	False	0.02037	0.02065	0.6272	10

Table A.8: Results of frozen encoder weights without transfer learning. Without transfer learning constitutes that only the fine-tuning dataset is used during pre-training.

A.6 Harder pretext task

Number of pre-training epochs	**Val. loss**	**Train loss**	**Val. acc.**	**Fine-tuning epochs**
10	0.01361	0.01385	0.7986	11
30	0.01396	0.01389	0.7948	11
50	0.01427	0.01425	0.7895	11
100	0.01437	0.01501	0.7891	10

Table A.9: Results of additive noise. Observe that metrics are worse than for an encoder pre-trained without additive noise. Remark that there is less degeneration in performance over the number of epochs.

Number of pre-training epochs	**Val. loss**	**Train loss**	**Val. acc.**	**Fine-tuning epochs**
10	0.01554	0.01507	0.7610	14
30	0.01447	0.01427	0.7798	14
50	0.01437	0.01438	0.7852	14
100	0.01464	0.01494	0.7809	15

Table A.10: Results of harder pretext task. Interestingly, the number of pre-training epochs and validation accuracy seam to correlate in contrast with previous experiments.

A.7 Percentage of labels available during fine-tuning

Percentage of labels	Pre-trained encoder	**Val. loss**	**Train loss**	**Val. acc.**	**Epochs**
1%	True	0.01389	0.01299	0.7867	14
1%	False	0.02153	0.01840	0.5251	14
10%	True	0.01226	0.01067	0.8146	14
10%	False	0.01906	0.01948	0.6710	14
50%	True	0.01179	0.00930	0.8295	14
50%	False	0.01192	0.01168	0.8197	15

Table A.11: Results of percentage of labels available for fine-tuning with trainable weights during fine-tuning for all encoders.