Source-free Domain Adaptation for Sleep Stage Classification

Yasmin Niknam, The University of Western Ontario

Supervisor: Boyu Wang, The University of Western Ontario
A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science
© Yasmin Niknam 2023

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation
https://ir.lib.uwo.ca/etd/9171

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.
Abstract

The popularity of machine learning algorithms has increased in recent years as data volumes have risen, algorithms have advanced, and computational power and storage have improved. EEG-based sleep staging has become one of the most active research areas over the last decade. Labeling each sleep stage manually is a labor-intensive and time-consuming process that requires expertise, making it susceptible to human error. In the meantime, training models on an unseen dataset remains challenging due to physiological differences between subjects and electrode sensor configurations. Unsupervised domain adaptation approaches may provide a solution to this problem by borrowing knowledge from a labeled dataset to train an unlabeled dataset. A source-free unsupervised domain adaptation methodology is employed in this thesis to solve the problem of automatic single-channel EEG sleep stage classification. Our study shows that pre-training source domain models followed by supervised fine-tuning improve the learned representations when applied to EEG sleep signals. We further develop weighted diversity loss in order to achieve a model that outperforms state-of-the-art unsupervised domain adaptation techniques without access to source domain data.

Keywords: Sleep Stage Classification, Unsupervised Domain Adaptation, Source-free Domain Adaptation, Self-supervised Learning, Electroencephalogram (EEG)
Summary for Lay Audience

The classification of sleep stages refers to the process of identifying the different stages of sleep that an individual experiences during the course of the night. Sleep has five different stages, each characterized by distinct patterns of brain activity, muscle tone, and other physiological indicators. It is essential to classify the stages of sleep in order to diagnose sleep disorders and related diseases. In spite of this, categorizing sleep stages manually is inefficient in terms of time and requires expert assistance. As a result, there has been extensive research concerning the use of machine learning to classify sleep stages automatically. This approach can be helpful in a variety of applications, such as developing sleep-tracking devices and applications. However, due to individual variability and the configurations of the sensors that collect sleep data, automated sleep stage classification is not always accurate. This is why researchers have employed domain adaptation techniques in automated sleep stage classification. It is a technique of leveraging knowledge from a related domain (also known as the source domain) to enhance the performance of a model in another domain (also known as the target domain). This thesis directly addresses this shortcoming by developing a novel approach to training a model to perform sleep stage classification automatically. It will investigate the factors that contribute to developing a methodology that uses the source domain knowledge to classify unlabeled target data while maintaining the privacy of its subjects.
## Contents

**Abstract** ................................................................. i 

**Summary for Lay Audience** ........................................... ii 

**List of Figures** ......................................................... vi 

**List of Tables** .......................................................... vii 

1 Introduction ................................................................... 1 
   1.1 Motivation .............................................................. 2 
   1.2 Contributions .......................................................... 3 
   1.3 Thesis Outline .......................................................... 4 

2 Background and Related Work ....................................... 5 
   2.1 Background ............................................................. 5 
      2.1.1 EEG Signals ......................................................... 5 
      2.1.2 Sleep Stage Classification Basics .............................. 5 
      2.1.3 Learning Algorithms .......................................... 6 
         Supervised Learning ................................................. 6 
         Unsupervised Learning ............................................ 7 
         Semi-supervised Learning ....................................... 8 
      2.1.4 Neural Network Architectures ................................. 8 
      2.1.5 Convolutional Neural Networks (CNNs) .................... 9 
         Convolutional Layers .............................................. 9 
         Pooling Layers .................................................... 10 
         Fully Connected Layers ......................................... 12 
      2.1.6 Activation Functions ........................................... 12 
         Rectified Linear Units (ReLUs) ................................. 12 
         Softmax .................................................................. 12 
         Sigmoid Function .................................................... 13 
         Hyperbolic Tangent .................................................. 13 
      2.1.7 Dropout ............................................................. 13 
      2.1.8 Batch Normalization ............................................ 13 
      2.1.9 Loss Functions ................................................... 15 
         Cross-entropy loss ................................................... 15 
         KL Divergence loss ................................................. 15
List of Figures

1.1 A comparison of UDA approaches and source-free UDA approaches. It is necessary to have access to the source data for the former, whereas it is not necessary for the latter. ........................................ 3
2.1 EEG brainwave activity during different stages ........................................ 7
2.2 Perceptron Architecture ......................................................................... 9
2.3 Multilayer Perceptrons Architecture ...................................................... 10
2.4 Overall architecture of a CNN module ................................................... 10
2.5 Computation of a convolutional layer ................................................... 11
2.6 Computation of maximum pooling and average pooling layers. .......... 11
2.7 Visualizing how dropout layer performs in a neural network ............... 14
2.9 Overview of the architecture of domain adversarial neural networks (DANNs). Feature extractors are shown in green, label classifiers in blue, and domain classifiers in pink) (image from [19]). ........................................ 19
2.10 Overview of the architecture of Source Hypothesis Transfer (SHOT). (image from [44]). ..................................................................................... 21
3.1 Overall framework of the proposed method .......................................... 26
3.2 Augmented Signal vs. Original signal using our augmentation method ... 28
4.1 Confusion Matrix results showing the target domain alignment for (a) Our Source-only model, (b) SHOT [44], and (c) Our model under the scenario of S1→EDF ......................................................... 38
4.2 Parameter Sensitivity Analysis for the Data Augmentation based on values of $p$ 39
4.3 Parameter Sensitivity Analysis for the Data Augmentation based on values of $\lambda$ 39
List of Tables

3.1 Notation Table ................................................. 26
3.2 The distribution of sleep stage of each dataset ................. 29
4.1 Description of Datasets ........................................ 32
4.2 Comparative Analysis of Various Baselines. Best results are in bold, and the second best are underlined. 36
4.3 Ablation Study Results ........................................ 39
Chapter 1

Introduction

Human health is directly influenced by the quality and duration of sleep [78]. Sleep disorders affect an estimated 50 to 70 million Americans each year. The majority of sleep disorders are not readily detectable and do not elicit symptoms until years later. A variety of chronic health problems have been linked to abnormal sleep architecture, such as sleep apnea, diabetes, depression, strokes, cardiovascular disease, Parkinson’s disease, and Alzheimer’s disease. Monitoring sleep quality is essential to developing medical research and treatment for sleep disorders, as well as aiding in their diagnosis.

In order to diagnose these disturbances, a patient is typically monitored overnight and the recorded signals are subsequently annotated. Sleep specialists evaluate the quality of sleep based on the electrical activity recorded by sensors placed on various parts of the body. These sensors record signals which constitute hospital polysomnogram recordings (PSG). The PSG consists primarily of an electroencephalogram (EEG), an electrooculogram (EOG), an electromyogram (EMG), and an electrocardiogram (ECG). It is normally segmented into epochs of 20 or 30 seconds, after which sleep experts classify them into different sleep stages according to sleep manuals, namely the Rechtschaffen and Kales (R&K) [65] and the American Academy of Sleep Medicine (AASM) [30]. The manual process of determining sleep stages is known as sleep stage scoring or sleep stage classification. However, it is labor-intensive, time-consuming, and prone to human error, since experts must examine a large number of PSG records. Considering the limitations of the number of experts, it cannot be applied on a large scale.

Recent advances in deep learning modules have led to a significant improvement in the automated classification of sleep stages [73, 72, 13, 58]. The models were designed to process EEG data, extract valuable features from it, and then train decent classifiers to achieve acceptable performance. The majority of developed models, however, rely on enormous amounts of labelled data, which may not be available in the case of automated sleep stage classification. Additionally, many sleep labs still relied on manual scoring as a result of a common problem known as domain shifting [55]. As the name implies, domain shift refers to the differences between the training dataset (source) and the testing dataset (target). The classification of sleep stages can be affected by a number of factors, including different measuring electrode attachment locations and different sampling rates of measuring devices. When a model trained on existing data is applied directly to a target dataset, or in other words, unlabeled data, it is possible that the performance of the model may be significantly diminisheds. Consequently, sleep...
laboratories could not trust a model trained on a public dataset for expert judgment.

1.1 Motivation

Several approaches have been proposed to solve the performance degradation problem by adapting the source model to the target domain. The supervised transfer learning method has been one of the most common settings researchers have used to solve the domain shift problem in sleep stage classification\[61, 52, 59, 24\]. They generally adapt a well-trained source model to a labeled target domain by fine-tuning the existing network. In this scenario, the target can be either a clinical site’s sleep records \[61, 24\] or a particular subject’s sleep records \[59, 52\]. However, transfer learning methods require the availability of a labeled target dataset. From a realistic point of view, there may be no or only a small fraction of labeled data in the target domain. This problem can be solved by using Unsupervised Domain Adaptation (UDA)\[19, 70, 11\]. As a specific case of transfer learning, UDA minimizes the mismatch between source and target distributions through learning their shared feature spaces without using any labels from the target domain. Over the past few years, sleep stage classification has significantly benefited from UDA approaches that incorporate adversarial learning. Most of these methodologies rely on domain discriminators to extract features that are invariant across different domains \[15, 55\].

The mentioned UDA approaches, however, require access to raw PSG signals in the source domain, which raises privacy concerns, since PSG signals may contain personal and sensitive information. Nowadays, the privacy of patient data has been one of the most significant concerns regarding interdisciplinary collaboration, particularly for PSG and EEG signals. A conventional UDA method requires the sharing of large, well-labeled source domain data, which may result in data leakage and property rights issues \[4\]. It is common for EEG data to be de-identified to address privacy concerns, but this may not always be sufficient to protect an individual’s privacy. Even after de-identification, some information may remain in the EEG signals that may allow individuals to be re-identified. If these patterns are known or can be inferred from other sources, unique features of EEG signals can be used to re-identify individuals. In addition, even if the EEG signals themselves do not contain any personal information, they could still be combined with other sources of data to reveal sensitive information about individuals. By combining EEG data with other medical or personal data, it may be possible to derive sensitive information about a person’s health status, cognitive capabilities, or other characteristics.

An emerging topic, source-free UDA, is attracting increasing attention primarily because it avoids potential violations of data privacy policies \[44\]. In source-free UDA, as shown in figure 1.1, a well-trained source model is adapted to an unlabeled target domain without making use of any source data or its statistics. It is also possible to reduce the cost of data transmission by only providing a well-trained source model in addition to minimizing privacy concerns.

It should be noted, however, that most source-free UDA approaches have been developed in order to solve problems related to class-balanced datasets, whereas sleep stage classification inevitably involves imbalanced datasets. There is a natural tendency for certain sleep stages to occur less frequently than others. Therefore, sleep stage datasets may be imbalanced for a variety of reasons. For example, deep sleep (also called slow wave sleep or stage N3 of
sleep) typically accounts for a smaller percentage of the total amount of time spent asleep in comparison to lighter stages such as stage N2. Consequently, a dataset that represents typical sleep patterns may include a greater number of lighter sleep stages and fewer deep sleep stages. Class imbalance in sleep stage datasets may also result from sampling biases or methods of data collection. A dataset, for instance, may contain a disproportionate number of examples of certain sleep stages if it is collected from individuals experiencing sleep disturbances or having other factors that affect their sleep patterns.

This thesis presents a source-free UDA approach, which is designed specifically for the classification of sleep stages from single-channel EEG signals. The approach incorporates weighted diversity loss to address the class imbalance problem in sleep datasets. Moreover, we investigated the impact of pre-training the source feature extractor by employing a supervised contrastive objective and developed a method of data augmentation that contributes to the pre-training process. By applying supervised contrastive learning, it is possible to improve the generalization performance of a model, enabling it to make better predictions on new, unseen data.

![Image of UDA and source-free UDA methods]

Figure 1.1: A comparison of UDA approaches and source-free UDA approaches. It is necessary to have access to the source data for the former, whereas it is not necessary for the latter.

### 1.2 Contributions

In this study, we focus on developing a source-free unsupervised domain adaptation approach designed for sleep stage classification. Our research examined how supervised contrastive techniques affected the extraction of more generalized features from source domains. In light of this, we created a novel data augmentation methodology for the relevant stage designed for single-channel EEG sleep data. We then extended SHOT [44] to develop a similar and more
effective approach to sleep staging. This was achieved by introducing major components that are critical to the success of the project. The main contributions we have made are as follows:

- We propose a source-free unsupervised domain adaptation method for Sleep Stage Classification, which provides a solution to both cross-dataset variations and confidentiality concerns at the same time.

- As part of our Sleep Stage Classification framework, we propose a unique approach to data augmentation, in conjunction with the supervised contrastive technique which contributes to the generalization of the trained source model.

- We present a novel framework for cross-dataset sleep staging by incorporating weighted diversity loss in the adaptation phase in order to account for class imbalance.

- Extensive experiments demonstrate that our proposed model, while source-free, provides superior cross-dataset sleep stage classification performance compared with state-of-the-art UDA methods.

1.3 Thesis Outline

In this chapter, the aim was to briefly describe the Sleep Stage Classification problem and explain how this challenge is being addressed by researchers, followed by a unique perspective regarding data leakage and privacy on this problem. A source-free UDA approach is discussed in further detail in the following chapters. We first developed a generalized source model by utilizing supervised contrastive loss and cross-entropy loss. In adjusting the source model for the target domain, we used information maximization loss and a self-supervised pseudo-labeling technique. During Chapter Two, we will describe the fundamentals and background of the problem, followed by a detailed discussion of related works and previous methodologies related to our research. In Chapter Three, we will explain our methodology in great detail. Chapter Four illustrates the results of the proposed method, followed by experiments and ablation tests. Lastly, Chapter Five concludes and outlines future work as well as explains its limitations.
Chapter 2

Background and Related Work

2.1 Background

The purpose of this section is to review the basics of our methodology. First, we provide an overview of the data type that is being used in this thesis, which is EEG signals for determining sleep stages. As a next step, we will discuss a few different learning algorithms in order to grasp the concept of our unsupervised learning method. We then review the basic blocks of a deep learning model that have been incorporated into our methodology. In the final section, we discuss the objectives that we use in training a model from scratch.

2.1.1 EEG Signals

Neurons communicate by exchanging electrochemical signals generated when ions move across the neuronal membrane. In the brain, electrical activity is produced by the activity of millions of neurons communicating with each other. This electrical activity of the brain can be measured with the Electroencephalography, EEG, technique. In EEG, small metal discs (electrodes) are attached to patients’ scalps to detect the activity of neurons in the cerebral cortex. During the simultaneous activation of populations of neurons, the electroencephalogram records the signal from nearby areas of the brain.

EEG offers numerous advantages, including its low cost and ability to measure brain activity within milliseconds. However, there are some limitations associated with using the EEG technique. First, it cannot record the activity of deeper brain regions. In addition, it is difficult to locate the precise location of the activity seen using EEG.

There is a wide range of applications for EEG in both clinical and research settings. It is commonly used to diagnose and monitor neurological disorders such as epilepsy, sleep disorders, and brain tumors. Additionally, EEG data may be valuable in cognitive studies. By examining brain activity during specific tasks or stimuli, researchers can gain a more comprehensive understanding of how information is processed in the brain.

2.1.2 Sleep Stage Classification Basics

While sleep appears to be a uniform state of being, in truth, this is not the case. It is composed of several stages, which can be distinguished by the use of PSG signals, most commonly EEG
brain waves. By measuring both the frequency and amplitude of brain waves, it is possible to visualize changes in brain wave activity in order to identify patterns. Sleep stages can be categorized into two general phases: Rapid Eye Movement (REM) and non-REM stages.

In REM sleep, also called paradoxical sleep, the eyes move rapidly, vivid dreams and muscle paralysis occur, resulting in darting movements of the eyes under closed eyelids. The brain waves during this stage are similar to those during the wake stage, and breathing and heart rate may become irregular. The REM sleep stage is essential for the consolidation of memories, the regulation of mood, and the restoration of brain function.

The non-REM sleep phase can be further divided into three stages:

- **N1 Stage**: This is the lightest stage of sleep, which usually lasts only a few minutes. It is common for one to drift into and out of sleep, as well as have sudden muscle contractions. As the body prepares to enter a deeper sleep stage, brain waves continue to slow down and the body temperature decreases.

- **N2 Stage**: A state of deep relaxation is achieved at this stage when brain waves slow down even further. During this stage, it may be difficult to wake up, and sleepwalking may occur, and it accounts for most of the sleep time. It is estimated that a normal human spends approximately half of their total sleep time in this stage, when the heart rate and breathing rate slow down and the body temperature drops.

- **N3 Stage**: As the brain waves are the slowest during this stage and the body is completely relaxed, this is the most profound sleep stage, also referred to as slow-wave sleep. Due to the fact that people may feel disoriented and groggy, it is difficult for them to be woken up at this time. It is the deepest stage of sleep, so the brain waves slow significantly, resulting in a period during which the body repairs itself and consolidates memories.

On the basis of the explanations above, physicians generally divide sleep into five stages: Wake, N1, N2, N3, and REM. Figure 2.1 illustrates EEG signal examples from each stage.

### 2.1.3 Learning Algorithms

Algorithms used for learning can be categorized into three broad categories based on their use of labeled data. We will briefly describe each of these categories in the following section.

**Supervised Learning**

As a subcategory of learning algorithms, supervised learning is characterized by the use of labeled datasets in order to train algorithms that are able to accurately categorize data or make predictions. In response to input data, the model modifies its weights by incorporating gradient descent until the model is adequately fitted. As a result of learning the feature vectors (inputs) to the labels (outputs) over time, the model is able to map inputs to labels (outputs). Lastly, through a cost function $J$, the algorithm measures its accuracy, adjusting until the error is sufficiently minimized.
Figure 2.1: EEG brainwave activity during different stages

\[
J(\theta) = \sum_{i=1}^{N} L(f(x_i, \theta), y_i),
\]

where \(N\) is the number of training pairs of \(((x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N))\), \(\theta\) is the model parameters, \(L\) is the loss functions that takes the model output \(f(x_i, \theta)\) and the groundtruth label \(y_i\) and outputs their difference.

**Unsupervised Learning**

The term unsupervised learning refers to the training of a machine using information that is neither classified nor labeled. This information allows the algorithm to make decisions without any guidance. It is the task of the machine to discover similarities, patterns, and differences among unsorted data without the need for human intervention. There are three main tasks that are accomplished through unsupervised learning models: clustering (e.g., K-Means), association (e.g., Association Rules), and dimensionality reduction (e.g., Principal Component Analysis [1]). Additionally, in deep learning settings, unsupervised learning tasks are often used to represent data distribution with a model, either implicitly (e.g., Generative Adversarial Networks [2]) or explicitly (e.g., Variational Autoencoders [3]).
Self-supervised learning, nonetheless, is essentially a discriminative unsupervised learning algorithm since it utilizes only unlabeled data. As opposed to conventional unsupervised learning methods, self-supervised learning focuses on solving the problems that are being solved by supervised learning, rather than those that are being solved by unsupervised learning, such as clustering. Often, this is accomplished by creating an automated supervisory signal that can serve as a means to perform an indirect task. We will discuss one of the most popular self-supervised learning paradigms, contrastive learning, later in this chapter.

Semi-supervised Learning

Semi-supervised learning, as its name suggests, is a hybrid technique combining supervised and unsupervised learning. In general, semi-supervision involves treating data points in different ways, depending on whether they are labeled or not: for labeled points, the algorithm trains or fine-tunes model weights and grounds predictions using traditional supervision methods; and for unlabeled points, the algorithm reduces the gap between the predictions derived from similar training examples and acquires an understanding of the distribution of data as a result. It is important to note that, in most cases, the proportion of labeled data is greater than the proportion of unlabeled data.

While fine-tuning is beneficial in complex models with many parameters, it can result in catastrophic forgetting in small models [51, 23, 42]. It is possible for the model to forget the parameters from a previously learned task when learning a new task due to catastrophic forgetting. Since many of the weights that are used to store knowledge have been modified, prior knowledge is not likely to be retained in its original form. This is why catastrophic forgetting occurs. It is evident from this phenomenon that unsupervised learning may be more advantageous from larger models after fine-tuning.

2.1.4 Neural Network Architectures

Neurons are nerve cells in the human brain that are connected and transmit electrical and chemical signals. The human brain contains billions of neurons. To mimic biological neurons, artificial neurons are mathematical functions that receive inputs, weight them individually, add them together, and pass the resulting sum through a nonlinear function to generate an output. A neural network refers to an artificial or biological group of neurons. The perceptron, a fundamental neural network, is a binary classifier used for supervised learning. In this algorithm, neurons are capable of learning and processing elements of the training set sequentially. Perceptron is a single-layer neural network with four main parameters. Input values are multiplied by their weights, then added to create a weighted sum. To obtain the desired output, this weighted sum is applied to the activation function. This process is shown in Figure 2.2.

The primary limitation of perceptron networks is that they can only classify vector sets that are linearly separable. Learning cannot reach a point where all vectors are correctly classified if the vectors are not linearly separable. By stacking perceptrons vertically, we can create a layer of perceptrons. Interconnecting these layers of perceptrons, known as hidden layers, leads to a multi-layered perceptron (MLP), also known as a deep feed-forward neural network, that is capable of approximating a non-linear function and addressing more challenging problems. Figure 2.3 depicts the overall architecture of a MLP. The name feedforward implies that the
output of each layer becomes the input to the next layer. Throughout this process, no loop or feedback connection is established between the previous or current layers. During the training process, the weights of the hidden layers are modified until the margin of error is minimized. If the input data is larger than a certain amount, such as an RGB image, then MLP models will require a high number of parameters and a long training period. One reason Convolutional Neural Networks are being used more in architecture today is because of this.

2.1.5 Convolutional Neural Networks (CNNs)

A class of artificial neural networks known as convolutional neural networks (CNNs) are increasingly being used in a variety of fields, including the analysis of medical data. CNNs are named after a mathematical operation called convolution that is used to construct these networks. Among the first implementations of CNNs were LeNet-5 [39], which were trained on the MNIST dataset [12]. With CNN, spatial hierarchies of features can be learned automatically and adaptively through backpropagation, utilizing a variety of building blocks such as convolution layers, pooling layers, and fully connected layers.

Convolutional Layers

CNNs utilize the convolutional layer as their core, where most computation occurs. This layer consists of three components: the input, the filter or kernel, and the output. Inputs are typically larger than feature detectors, but they contain the same dimensions as outputs. Filters perform element-wise multiplication across the receptive fields of an input, resulting in a single output after moving across the receptive fields. As the kernel slides over each location, it repeats this process, which converts the input matrix of features into a new matrix. This process, which is
known as convolution, can be mathematically described as follows:

\[(f * g)(i) = \sum_{j=1}^{m} g[j].f[i - j] \quad (2.2)\]

where \(f \in \mathbb{R}^n\) is a one-dimensional input vector with length of \(n\), and \(g\) is a 1D kernel vector with the length of \(m\). This operation is performed for each \(i \in n\). A kernel will move a certain number of steps per computation based on the value of stride.

**Pooling Layers**

Once high-level task-relevant features are extracted by convolutions, we are required to reduce unnecessary spatial dependencies between features as we progress to higher levels of representation with the aim of limiting the risk of overfitting and increasing efficiency. A pooling layer
2.1. Background

Figure 2.5: Computation of a convolutional layer

also referred to as a downsampling block, reduces dimensionality, resulting in a smaller spatial size of the generated feature maps. A pooling operation is similar to a convolutional layer in that it sweeps a filter across the entire input, however, this filter does not have any weights. A kernel instead aggregates the values within the receptive field, populating the output array. There are several types of down-sampling operations, but the following are the most commonly employed pooling layers:

- **Maximum pooling:** As the filter traverses the feature map, it identifies the pixel with the highest value, producing a pooled feature map that accentuates the most prominent characteristic within the patch.

- **Average pooling:** As the name implies, the filter moves across the feature map and computes the average value within the receptive field for the output array.

Figure 2.6: Computation of maximum pooling and average pooling layers.
Fully Connected Layers

A commonly employed strategy in CNNs is to incorporate an MLP or fully connected layer as the final convolutional layer. This layer accepts the flattened or globally pooled output of the preceding convolutional layer as its input. By doing so, the fully connected layer can learn nonlinear combinations of features in a cost-effective manner. To facilitate classification, the number of nodes in the final fully connected layer can be modified to correspond to the number of categories, with each node providing the likelihood that the input belongs to that specific class.

2.1.6 Activation Functions

The activation function determines whether a neuron’s input to the network should be activated or not through the use of straightforward mathematical operations. It introduces non-linearity into output by calculating output from input values fed to the node (or layer). Thus, neural networks and architectures are capable of handling tasks of greater complexity than simply linear regression models. In this section, we will review some of the most widely used activation functions.

Rectified Linear Units (ReLUs)

ReLU is one of the popular activation functions, which simply passes positive values as they are, whereas negative values are transformed to zero. As a result of its low computational cost and sparseness, this function can be useful and efficient. Due to the fact that ReLU does not require any exponential calculation, it is considerably less computationally expensive than Tanh or Sigmoid activation functions. Moreover, ReLU is capable of returning an absolute zero value. The result will be sparse and efficient activations since some neurons or hidden units will not be activated. Nevertheless, it suffers from a major disadvantage when the output value is less than zero. In such case, the gradient will always remain zero, resulting in the weights of the network remaining unchanged. To prevent this from happening, researchers have proposed various versions of ReLU, notably, Leaky ReLU [50] and PReLU[26].

\[
ReLU(x) = \max\{0, x\} \quad (2.3)
\]

Softmax

The softmax activation function is commonly used in machine learning for multiclass classification problems. The function takes a vector of real numbers as input and outputs another vector of the same size, in which each element represents the probability of each category.

The softmax function works by first exponentiating each element of the input vector to make them positive, and then dividing each element by the sum of all the exponentiated values. In this case, the output vector contains values between zero and one, and its sum equals one. The formula for the softmax function is as follows:

\[
Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad (2.4)
\]
where $x$ is the input vector. The softmax function is useful for multiclass classification problems because it converts the output of a neural network or other machine learning model into a probability distribution over the classes.

**Sigmoid Function**

The function can receive any real number as input and returns a value between 0 and 1. As the input value rises (becomes more positive), the output value approaches 1.0, whereas as the input value drops (becomes more negative), the output value approaches 0.0. It is essential to note that the Sigmoid activation function is extensively used in neural networks. However, since the Sigmoid function is limited to the range $(0, 1)$, the output becomes saturated as the inputs become very large or very small. Consequently, gradients in that area usually become trivial, resulting in gradients that disappear.

$$
s(x) = \frac{1}{1 + e^x} \quad (2.5)
$$

**Hyperbolic Tangent**

The Tanh activation function bears a resemblance to the Sigmoid activation function in many ways. In fact, both have the same S-shaped curve, but the Tanh output range varies from -1 to 1. When using Tanh, if the input value increases (becomes more positive), the output value will be nearer to 1.0, and if the input value decreases (becomes more negative), the output value will be closer to -1.0. Like Sigmoid, this function is vulnerable to the vanishing gradient issue.

$$
tanh(x) = 2s(2x) - 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.6)
$$

**2.1.7 Dropout**

Deep learning neural networks are likely to overfit a training dataset with limited samples very quickly. Overfitting can be reduced through the use of ensembles of neural networks with different model configurations. However, doing so comes with an additional computational cost in training and maintaining multiple models. One model can simulate a large number of different network architectures by randomly dropping nodes during training. The dropout method is a computationally cheap and remarkably efficient regularization technique that reduces overfitting and improves generalization error in deep neural networks. During training, this layer will be added as an additional layer to the network and set input units to 0 randomly at a fixed *rate*. Inputs that are not set to 0 are scaled up by $\frac{1}{1 - \text{rate}}$ to keep the sum of all inputs unchanged.

**2.1.8 Batch Normalization**

Normalization in statistics aims to standardize a given data distribution by converting values from various scales to a common scale, aligning it with the normal distribution with a mean of 0 and a standard deviation of 1. In the context of deep learning, the data input to the model is normalized by subtracting the mean and scaling the variance of the data.
In this process, known as Batch Normalization (Algorithm 1), an additional layer is added to the neural network, which performs operations on the inputs from the previous layer. This reduces the internal covariate shift, resulting in a faster training process for deep neural networks. Additionally, batch normalization mitigates gradient instability problems because gradients are less dependent on initial values or parameter scales. Thus, higher learning rates can be employed without the risk of diverging.

Algorithm 1 Batch Normalizing Transform, applied to activation $x$ over a mini-batch $[31]$

**Input:** Values of $x$ over a mini-batch: $B = \{x_{1...N}\}$; Parameters: $\gamma, \beta$

**Output:** $\{y_i\}$

$$\mu_B = \frac{1}{N} \sum_{i=1}^{N} x_i$$  \hspace{1cm} \text{\texttt{\# mini-batch mean}}

$$\sigma^2_B = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu_B)^2$$  \hspace{1cm} \text{\texttt{\# mini-batch variance}}

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma^2_B + \epsilon}}$$ \hspace{1cm} \text{\texttt{\# normalize}}

$$y_i = \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i)$$ \hspace{1cm} \text{\texttt{\# scale and shift}}
2.1. Background

2.1.9 Loss Functions

Generally, a loss function in machine learning measures the discrepancy between the expected output and the predicted output. Loss functions are utilized in training machine learning models to calculate how well they are performing and determine how much weight is given to errors in classification or prediction. There are many types of loss functions that can also be combined with one another to form new types of loss. Throughout this section, we will explain the loss functions used in this thesis.

Cross-entropy loss

The cross-entropy loss, or log loss, is a performance metric used in both traditional machine learning and deep learning algorithms. A model’s cross-entropy loss measures how well it is able to predict the true class of a sample. Essentially, it is the sum of the negative log-likelihoods of the true class. To put it another way, the cross-entropy loss represents the negative logarithm of the predicted probability for the true class.

KL Divergence loss

The Kullback-Leibler divergence score or KL divergence loss is a statistical measure of how much two distributions differ. This loss function may be applied to many different algorithms, including logistic regression, support vector machines, and neural networks, to find the optimal match between two probability distributions. Often, the KL divergence between two distributions $P$ and $Q$ is expressed as follows:

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) \log \left( \frac{P(x)}{Q(x)} \right) \quad (2.9)$$

2.1.10 Backpropagation

Artificial neural networks often use backpropagation as a training algorithm for prediction or decision-making models. In this iterative optimization algorithm, weights and biases of connections between neurons are adjusted in order to enable the neural network to learn from labeled data.

The backpropagation algorithm works by propagating the error between the predicted output and the true output of the network backward through the layers of the network. The gradient of the error is computed using the chain rule of calculus in order to determine the weights and biases that minimize the training data error. The gradients are then used to update the weights and biases in a manner that minimizes error.

Deep learning relies heavily on backpropagation algorithms since it is a powerful tool to train large and complex neural networks that require millions of parameters on large amounts of data. It is possible to use the backpropagation algorithm in almost any type of machine learning architecture, including convolutional layers, recurrent neural networks, and deep learning.
2.1.11 Optimizers

There are several categories of optimizers used in machine learning. One of the most commonly used optimization algorithms in deep learning is the first-order and second-order gradient-based optimizer. These optimizers employ the first derivative (gradient) and second derivative of the loss function with respect to the model parameters to update the weights during training. A few examples include Stochastic Gradient Descent (SGD) \[5\], and Adam \[35\]. During this study, we employed SGD and Adam optimizers in the training and adaptation phases.

In contrast to the traditional gradient descent algorithm which computes the gradient using the entire dataset, the SGD algorithm randomly selects a small batch of data points from the training dataset to compute the gradient of the loss function. Using small batches, SGD can compute the gradient more frequently and reach a convergence faster than batch gradient descent.

Adam (Adaptive Moment Estimation) optimizer is an extension of SGD which combines the ideas of both momentum-based and Adagrad-like optimizers. Based on an exponential moving average of both the first and second moments of the gradient, Adam adjusts the learning rate for each weight which can lead to faster convergence and better performance. As a result of its adaptive learning rate, Adam is an appropriate optimizer for sparse gradients and noisy data. However, the Adam optimizer is susceptible to getting stuck in local minima as a result of its adaptive learning mechanism.

2.1.12 Pseudo Labeling

Pseudo-labeling was first proposed by \[40\], where a small set of labeled data is combined with a large set of unlabeled data to improve the performance of a model. Although the technique was initially developed to address semi-supervised learning problems, it quickly became popular for transductive learning problems, such as domain adaptation \[9\]. A primary aspect of this method is the use of maximum predicted probability to approximate labels for unlabeled data and to perform fine-tuning using labeled data, a process that is quite efficient. However, the major disadvantage of this method is that false pseudo-labels are trained into the network, which can negatively affect the process. In the case of unsupervised learning, this is particularly evident.

2.1.13 Data Augmentation

Data augmentation is the process of generating supplementary data points from existing data to artificially increase the amount of training data in order to avoid overfitting. This may involve minor alterations to the data or the use of machine learning models to generate new data points in the feature space of the original data. As a way to have the most comprehensive range of possible data points, this approach has been commonly used for computer vision problems. Thus, the learned representations would be more general rather than specific to the training set, resulting in a model capable of performing well on a test set or similar dataset. Essentially, the fundamental assumption behind data augmentation is that small distortions will not alter the semantic meaning of the data. Depending on the domain of the input data, these distortions may vary. Cropping, contrast, scaling, and rotation are examples of augmentations that can
be applied to images. In contrast, there are augmentations such as mixup [83], which can be applied to a variety of data types, such as images, audio, and time series. As part of this thesis, we introduce a novel data augmentation technique for EEG signals that can improve the performance of a classifier used to classify sleep stages based on single-channel EEG signals.

2.1.14 Contrastive Learning

Contrastive learning is one of the most commonly used self-supervised learning techniques in machine learning, and specifically in the field of computer vision. As the name implies, the basic concept behind this idea is to contrast different data samples to learn the common characteristics among data classes and the characteristics that separate them from each other. In fact, this approach mimics how humans learn about the world around them, and has shown promising results when applied to deep learning modules. The algorithm selects a data point, called an anchor, and augments the data to produce two views based on that data point. The pairs of augmented data samples are known as "positive" pairs, while the rest of the data samples are known as "negative" pairs. This self-supervised learning algorithm attempts to minimize the distance between the anchor and the positive pair while simultaneously maximizing the distance between the anchor and the negative samples.

Supervised Contrastive Learning

For fully supervised problems, this approach adapts the contrastive learning methodology. In many situations, it is a more effective method of learning than supervised learning, particularly where discriminatory representations are required. In fact, Supervised Contrastive Learning utilizes label information to cluster samples belonging to the same class within the feature space. The concept is relatively straightforward. The model should be capable of learning to map the normalized embeddings of samples belonging to the same class closer and those belonging to the other class/classes further away. In Figure 2.8, the supervised contrastive objective is depicted and compared with the self-supervised method. The embeddings of all the cat images should be close together and distant from all the dog images and vice versa.

Contrastive Learning Loss functions

According to [34], for a set of $N$ randomly sampled data/label pairs, $\{x_k, y_k\}_{k=1\ldots N}$ the corresponding batch used for training consists of $2N$ pairs, $\{\tilde{x}_k, \tilde{y}_k\}_{k=1\ldots 2N}$ where $\tilde{x}_{2k}$ and $\tilde{x}_{2k-1}$ are two augmentations of $x_k$ and $\tilde{y}_{2k} = \tilde{y}_{2k-1} = y_k$. We refer to a set of $N$ samples as a batch and a set of $2N$ augmented samples as multiviewed batch.

- Contrastive Learning Objective

\[
L_{\text{self}} = \sum_{i \in I} L_{i}^{\text{self}} = - \sum_{i \in I} \log \frac{\exp(z_i \cdot z_{j(i)}/\tau)}{\sum_{a \in A(i)} \exp(z_i \cdot z_a/\tau)} \quad (2.10)
\]

Where $i \in I \equiv \{1\ldots 2N\}$ is the index of an arbitrary sample, $j(i)$ is the second augmented view initiated from the same source sample, $z = f(x)$ is the feature of a given data point,
Supervised Contrastive Learning Objective

\[ L^{sup} = \sum_{i \in I} L_i^{sup} = -\sum_{i \in I} \log \left( \frac{1}{|P(i)|} \sum_{p \in P(i)} \frac{exp(z_i \cdot z_p / \tau)}{\sum_{a \in A(i)} exp(z_i \cdot z_a / \tau)} \right) \]  (2.11)

Here, \( P(i) = \{ p \in A(i) : \tilde{y}_p = \tilde{y}_i \} \) is the set of indices of all positives in the multiviewed batch distinct from \( i \) and \(|P(i)|\) is its cardinality.

2.2 Related Work

2.2.1 Unsupervised Domain Adaptation

The goal of unsupervised domain adaptation (UDA) is to leverage labeled data from a source domain and apply the knowledge to an unlabeled target domain. Most existing UDA methods can be divided into two categories: discrepancy-based methods\[70, 11, 48\] and adversarial learning methods\[19, 85, 49\].
2.2. RELATED WORK

**Discrepancy-based Methods**

A discrepancy-based method utilizes a well-defined statistic moment matching distribution loss across domains in order to reduce differences between the two domains and obtain domain-invariant feature representations. Different distance loss functions are typically added to the activation layers of networks. Deep Domain Confusion (DDC) [74] attempted to explicitly match different domain feature distributions by minimizing the maximum mean discrepancy (MMD). Unlike DDC, which uses a single layer and linear MMD, the deep adaptation network (DAN) [47] considers several MMDs between several layers and explores multiple kernels to adapt deep representations. [87] employs a local MMD loss to align the distributions of the same class relevant subdomains. In CORAL [70], second-order statistics (covariances) are aligned between cross-domain distributions. As another discrepancy metric, the Wasserstein metric [76] measures the distance between domain samples. Using the Wasserstein distance, [11] matched feature and label space distributions together. The representations were not only aligned between the source and target domains, but they also preserved discriminative information.

**Adversarial Learning Methods**

Adversarial-based methods are becoming increasingly popular for reducing discrepancies between domains by utilizing adversarial domain discriminators. Inspired by generative adversarial networks (GAN) [22], adversarial learning models provide an effective method for identifying invariant representations in domain adaptation. DANN [19] is one of the first adversarial methods for adversarial based DA. As shown in figure 2.9, a gradient reversal layer was incorporated to promote the discrimination of source and target domains. By using the gradient reversal layer during backpropagation-based training, DANN ensures that the feature distributions across both domains are indistinguishable.

Figure 2.9: Overview of the architecture of domain adversarial neural networks (DANNs). Feature extractors are shown in green, label classifiers in blue, and domain classifiers in pink) (image from [19]).
ADDA [75] splits source and target domains via an inverted label GAN loss to allow learning of the features separately. DIRT-T [69] approach integrates virtual adversarial domain adaptation with a teacher model in order to refine the boundary conditions for the target domain. CAN [84] introduced a set of domain classifiers with each domain classifier being linked to one hidden representation from each CNN feature extraction block. Taking inspiration from conditional GANs [53], [49, 56] combined predicted discriminative information with learned features to improve discrimination and alignment of features. [79] developed a GAN-based architecture called adversarial domain adaptation with domain mixup, which facilitates the transfer of domain knowledge through mapping the two domains into a common potential distribution. Discriminator-free adversarial learning network (DALN) [7] proposed a simple yet effective adversarial paradigm that addresses this problem from a new perspective. Using the category classifier as a discriminator, [7] achieves explicit domain alignment and category differentiation through a unified objective.

2.2.2 Source-free Domain Adaptation

Adapting a domain involves reusing the knowledge from the source domain for the target domain. In the case of source data-free domain adaptation (SFDA), however, only unlabeled target data can be examined to extract valuable information from a pre-trained source model. According to [46], there are two main categories: one involves self-supervised training on target samples, and the other involves reconstructing virtual sources so that knowledge can be transferred.

Self-supervised Training

One way to adapt a pre-trained model to an unlabeled target dataset is to generate a collection of imprecise pseudo labels and subsequently fine-tune the target model through optimization of the adaptation loss function. The target model, however, may be misled by the false-labeled samples.

As an early study, the method developed by [43] was based on the nearest-centroid classifier, which searches for subspaces where target domain centroids are moderately shifted from source domain centroids. Later, [44] proposed the Source Hypothesis Transfer (SHOT) approach based on Hypothesis Transfer Learning (HTL). As illustrated in figure 2.10, the source model (hypothesis) classifier has been frozen and the target-specific feature extractor has been fine-tuned. The representations from the target domains are implicitly aligned to the source hypothesis by utilizing both information maximization and self-supervised pseudo-labeling. Basically, the purpose of information maximization is to decrease prediction uncertainty through an iterative process. Additionally, pseudo-labels are generated by weighted k-means clustering [6] and nearest centroid classifiers. A further extension of SHOT is SHOT++ [45], which utilizes a new labeling transfer strategy based on the confidence of predictions. In this strategy, target data is divided into two segments based on the confidence of predictions, and semi-supervised learning techniques are applied to improve the accuracy of less confident predictions. [8] utilizes adaptive batch normalization to reduce the noise ratio and update the target data and then produce more precise pseudo-labels via deep transfer clustering [25]. The target set was also separated into clean and noisy parts for robust training by use of exponential
momentum average (EMA).

Virtual Source Knowledge Transfer

Using virtual source knowledge transfer, some source impressions can be derived from a pretrained source model, or target data can be translated into the source style, which then can be used to replace the source data with the target data. Using synthetic source data will prevent the loss of source knowledge by distilling source knowledge from the pretrained model and transferring it to the target model.

A generative approach was developed by [28] in which target images are converted to source images under style and content constraints, and the feature statistics of generated images are aligned with those stored in pretrained batch normalization layers. [32] proposes to align the data distributions by matching the stored statistics of the source domain Batch Normalization with those of the target domain, which implicitly achieves style transfer. When fine-tuning a training feature extractor, KL divergence is used to measure the discrepancy between domain distributions, and information maximization is employed to adapt the classifier. Using a generative adversarial framework, [41] synthesized source impressions under the supervision of source model priors and target images. To distinguish between the real target data and the synthetic source data, they employed a discriminator. This approach was followed by [38], but to derive domain invariant features, an additional domain discriminator was added.

Figure 2.10: Overview of the architecture of Source Hypothesis Transfer (SHOT). (image from [44]).

2.2.3 Sleep Stage Classification

The classification of sleep stages has been automated for several reasons, one of which is to improve efficiency. By automating the sleep stage classification process, a significant amount of time and effort can be saved compared to manual classification by a human expert. The process of becoming an expert in such a complex field may take several years of study and considerable effort. Furthermore, the task itself is quite tedious and demanding. There is a need
for a rapid and reliable method of analyzing large quantities of data generated by digital sleep monitoring devices with the increasing availability of such devices. In addition, automating the classification of sleep stages allows for the analysis of large amounts of data, enabling researchers to investigate sleep patterns across larger populations and over a longer period of time. Automating sleep stage classification provides researchers and clinicians with a reliable, efficient, and objective method for analyzing sleep data, allowing them to gain valuable insights into sleep patterns and disorders rather than simply labeling them.

An individual typically experiences five stages of sleep during a full sleep cycle. A sleep specialist usually determines the sleep stages using the PSG, which includes an electroencephalogram (EEG), electrooculogram (EOG), electromyogram (EMG), and electrocardiogram (ECG) [33]. Among all these signals, single-channel EEGs have become increasingly popular for sleep monitoring due to their ease of use. Several studies have been conducted on the use of single-channel EEG for automatic sleep classification. As reported in earlier studies, researchers employed traditional strategies to obtain features from EEG signals, i.e., short-time Fourier transforms (STFTs) and discrete wavelet transforms (DWTs) [80], and then utilized statistical classifiers namely support vector machines (SVMs) [2]. In recent years, deep learning based methods [73, 58, 57, 72] have demonstrated substantial advances due to their capability to extract useful features from raw signals automatically, thus reducing the need for traditional methods and hand-engineered features. In these methods, a variety of neural networks have been used to extract features and temporal dependencies from EEG recordings. A number of studies have utilized convolutional neural networks (CNNs) in the top layer to obtain useful features from raw PSG epochs as well as bidirectional recurrent neural networks (RNNs) in the bottom layer to determine transition rules and other temporal information [58, 73, 5]. [73] proposed two CNNs with different filter sizes to extract time-invariant features from EEG signals, followed by a bidirectional Long Short-Term Memory (LSTM) [67] to encode temporal information into the model. Mousavi et al. [54] employed sleep staging by combining a CNN, a bidirectional RNN, and an additional attention network to analyze input sequences and extract useful features.

Differently from this type of structure, sleep stages have been classified in a variety of ways. Perslev et al. [57] is a temporal fully convolutional network inspired by the U-Net [66] architecture, which was initially proposed to segment images. It maps arbitrary input sequences to label sequences on a custom time scale by implicitly classifying each time point and aggregating them over fixed intervals for the final predictions. Eldele et al. [13] employed a multi-resolution CNN with adaptive feature recalibration to extract representative features. Following the structure, a multi-head self-attention approach is used to capture temporal dependencies.

### 2.2.4 Transfer Learning for Sleep Staging

As previously mentioned, automatic sleep stage classification has become increasingly popular in recent times. This is due to the fact that it eliminates the need for subject matter experts to constantly categorize patient data, which is both time-consuming and labor-intensive. There are, however, some difficulties when it comes to classifying sleep stages automatically. One of the most challenging aspects of this problem is the lack of available datasets. Datasets used for this particular problem should contain a significant amount of labeled data in order for a model to distinguish between different stages as an expert would. Unfortunately, a sizeable publicly
available dataset is not always available in this case.

In addition, EEG data are difficult to classify and do not have a high degree of generalizability. Several factors contribute to the phenomenon. One of these factors is the high inter-subject variability of EEG data. Subjects have physiological differences, like skull shape, and their individual neural activities might not propagate similarly [55]. In addition, each clinic may use different hardware and the location of sensors may vary as well, resulting in significant variability in EEG signals.

Due to the challenges outlined above, researchers began focusing their efforts on developing sleep models that could be applied to other domains following extensive research on automatic sleep stage classification. Some early studies investigated the problem of personalized sleep staging to improve classification accuracy [52], [60]. However, cross-dataset studies have been limited, i.e., training a model on one dataset and testing it on another dataset. According to Phan et al. [61], although automated sleep staging represents comparable performance to manual scoring by sleep experts, it has not been widely adopted clinically due to two major technical shortcomings: data variability and data inefficiency. Their model was trained on a large-scale dataset and fine-tuned on a much smaller one.

Generally, transfer learning methods have the disadvantage of being dependent on a large amount of data. In addition, they require labeled target data, which may not always be available. The factors listed above led researchers to propose unsupervised domain adaptation (UDA) for automated sleep staging. Most UDA algorithms for sleep stage classification solve the domain shift problem by finding domain invariant feature representations. As mentioned previously, these approaches can be divided into discrepancy-based and adversarial approaches. Discrepancy-based methods use statically defined distance functions to minimize the distance between the source and target domains [11, 70]. In contrast, adversarial-based methods identify domain-invariant features via the use of two competing networks [19].

Nasiri and Clifford [55] presented the first study that employed adversarial-based methods to solve the UDA problem for the classification of sleep stages. By observing the way clinicians manually label sleep stages, their study utilized two attention mechanisms, local attention, and global attention, to extract transferable information across individuals from different datasets. Zhao et al. [86] employed multiple domain classifiers at different levels of feature extraction to take advantage of both domain-invariant and domain-uninformative features. Yoo et al. [81] introduced two local discriminators, for subject and stage discrimination, alongside the global discriminator, in order to preserve the intrinsic structure of the sleep data. Heremans et al. [27] examined the use of adversarial UDA on wearable sleep datasets collected from diseased patients. Their study demonstrates that the performance of target datasets can be improved by taking pseudo-labels and real target domain labels into account when available. Eldele et al. [15] developed an unshared attention mechanism in conjunction with adversarial training to maintain domain-specific characteristics in both domains. To further adjust the classification boundaries to fit the target domain, they used dual classifiers. Recently, Ragab et al. [63] evaluated the adaptability of these state-of-the-art visual UDA methods to time series data, as well as the recent methods developed specifically for time series data. Based on their findings, visual UDA methods are comparable with time series UDA methods when hyper-parameters are carefully selected. All these methods, however, require access to the source domain data by the target user. This may be considered unsafe, or otherwise not feasible since the source data may be private or distributed.
To the best of our knowledge, Fan et al. [16] is the only methodology that explicitly considers SFUDA settings for sleep staging classification. According to Fan et al. [16], domain-specific statistics are modulated in the networks in order to align the distribution of different domains. While the proposed methods are easy to implement without access to source data, they still require some statistics of the source domain data.
Chapter 3
Methodology

3.1 Preliminaries

This study addresses UDA for the classification of sleep stages based on EEG data using only a pre-trained source model without access to the source data. Specifically, we consider $K$-way classification, where $K$ is the number of sleep stages.

In a vanilla UDA task we are given a set of $N_s$ labeled samples $\{x^i_s, y^i_s\}_{i=1}^{N_s}$ from the source domain $D_s$ where $x^i_s \in X_s$, $y^i_s \in Y_s$, and $N_t$ unlabeled samples $\{x^i_t\}_{i=1}^{N_t}$ from the target domain $D_t$ where $x^i_t \in X_t$. It aims to predict the labels $\{y^i_t\}_{i=1}^{N_t}$ in the target domain such that $y^i_t \in Y_t$.

Additionally, the source task $X_s \rightarrow Y_s$ is considered to be the same as the target task $X_t \rightarrow Y_t$. Here, we propose to learn the target function $f_t : X_t \rightarrow Y_t$ and predict $\{y^i_t\}_{i=1}^{N_t}$, with merely $\{x^i_t\}_{i=1}^{N_t}$ and the source function $f_s : X_s \rightarrow Y_s$. $x^i_s$ and $x^i_t$ in our problem settings are both in $\mathbb{R}^{1 \times T}$, in which the number of electrodes/channels is 1, and the number of time steps in the 30-second EEG periods is $T$. Table 3.1 provides a list of symbols and their definitions for future reference.

3.2 Overview

Through this framework, we address the aforementioned model transfer task for UDA in three stages. As shown in Figure 3.1 we first pre-train the source model feature extractor on source data using supervised contrastive objectives. We then develop a source model from the source data, discontinue the use of the source data, and transfer the model to the target domain without having access to the source data.

3.2.1 Pre-training with Supervised Contrastive Learning

Considering that we are attempting to solve the domain adaptation problem, generalizability is of primary importance. By adapting supervised contrastive objectives [34] as a pre-training method for learning more generalized features in sleep stage classification, we achieved better robustness and stability against model hyperparameter settings, leading to a more generalized feature extractor.
Table 3.1: Notation Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{X}$</td>
<td>Feature space</td>
</tr>
<tr>
<td>$\mathcal{Y}$</td>
<td>Label space</td>
</tr>
<tr>
<td>$K$</td>
<td>Number of categories(sleep stages)</td>
</tr>
<tr>
<td>$\mathcal{D}_s$</td>
<td>Data set of the source domain</td>
</tr>
<tr>
<td>$\mathcal{D}_t$</td>
<td>Data set of the target domain</td>
</tr>
<tr>
<td>$N_s$</td>
<td>Number of data samples in the source domain</td>
</tr>
<tr>
<td>$N_t$</td>
<td>Number of data points in the target domain</td>
</tr>
<tr>
<td>$f_s$</td>
<td>source model</td>
</tr>
<tr>
<td>$g_s$</td>
<td>source feature extractor</td>
</tr>
<tr>
<td>$h_s$</td>
<td>source classifier</td>
</tr>
<tr>
<td>$f_t$</td>
<td>target model</td>
</tr>
<tr>
<td>$g_t$</td>
<td>target feature extractor</td>
</tr>
<tr>
<td>$h_t$</td>
<td>target classifier</td>
</tr>
<tr>
<td>$z_s$</td>
<td>source feature extractor output</td>
</tr>
<tr>
<td>$T$</td>
<td>number of time steps</td>
</tr>
<tr>
<td>$N_{seg}$</td>
<td>number of segments</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>weakening bound</td>
</tr>
<tr>
<td>$\lambda_{seg}$</td>
<td>weakening factor</td>
</tr>
</tbody>
</table>

Figure 3.1: Overall framework of the proposed method

Assuming there are $N_s$ source domain data/label pairs, training data consists of $2N_s$ pairs,
\[ \{\tilde{x}_i, \tilde{y}_i\}_{i=1...2N_s} \] such that \( \tilde{x}_{2i} = x_i \) and \( \tilde{x}_{2i-1} \) is the augmentation of \( x_i \), and \( \tilde{y}_{2i} = \tilde{y}_{2i-1} = y_i \). As in [34], we refer to a set of \( 2N_s \) real and augmented samples as multiviewed data. With the supervised contrastive loss, we train a source feature extractor \( z = g_s(x) \) on multiviewed data that is highly generalizable to different domains.

\[
\mathcal{L}_{sup} = - \sum_{i \in I} \log \left( \frac{1}{|P(i)|} \sum_{p \in P(i)} \frac{\exp(z_i \cdot z_p/\tau)}{\sum_{a \in A(i)} \exp(z_i \cdot z_a/\tau)} \right)
\] (3.1)

As mentioned in section 2.1.14 here, \( i \in I \equiv \{1 \ldots 2N_s\} \) is the index of a sample, \( A(i) \) includes all the samples except for \( i \), \( P(i) = \{ p \in A(i) : \tilde{y}_p = \tilde{y}_i \} \) is the set of indices of all positives in the multiviewed batch distinct from \( i \), \( |P(i)| \) is its cardinality, and \( \tau \in \mathbb{R}^+ \) is a scalar temperature parameter.

**Data Augmentation**

We propose a unique approach for the augmentation of sleep data by segmenting the 30s of sleep epochs into multiple segments. Next, random segments are weakened in order to force the model to extract discriminatory features from all segments, thereby reducing the dominance of the key segments. By doing so, all segments of the EEG signal contribute to the classification process, resulting in improved generalization.

Given \( x_s \in \mathbb{R}^{1 \times T} \) represents a sample of source domain EEG signals, where \( T \) is the number of time steps, a set of source domain EEG signals is divided into \( N_{seg} \) consecutive segments \( X_{seg} = \{x_i^{N_{seg}}\}_{i=1}^{N_s} \), without overlapping, \( x_i^{N_{seg}} \in \mathbb{R}^{1 \times T/N_{seg}} \). In the next step, we define a weakening percentage \( p \), which indicates the number of segments that will be randomly selected for weakening. The signal magnitude of a selected segment \( x_i^{N_{seg}} \) is then weakened by a factor of \( \lambda_{seg} \), which is a uniformly distributed random number in the range of \([\lambda, 1]\), such that \( \lambda \in [0, 1] \) is the weakening bound that controls its magnitude. More specifically, \( x_i^{N_{seg}} = \lambda_{seg} x_i^{N_{seg}} \). An example of this augmentation is shown in Figure 3.2.

### 3.2.2 Training the Source Model

Our approach consists of developing a deep neural network to learn the source model \( f_s : X_s \rightarrow Y_s \), where \( f_s = g_s \circ h_s \) and \( g_s \) is the pre-trained feature extractor explained above, by minimization of the following standard cross-entropy loss along with label smoothing to increase its discrimination ability,

\[
\mathcal{L}_{cls}(f_s; X_s, Y_s) = - \frac{1}{N_s} \sum_{i=1}^{N_s} \sum_{k=1}^{K} q_{k,i}^{ls} \log \delta_k(f_s(x_i^{s}))
\] (3.2)

where \( \delta_k(a) = \frac{\exp(a_k)}{\sum \exp(a_i)} \) denotes the \( k \)-th element in the softmax output of a K-dimensional vector \( a \), \( q_{k,i}^{ls} = (1 - \alpha)q_{k,i} + \alpha/K \) is the smoothed label, \( q \) is the one-of-\( K \) encoding of \( y_s \) where \( q_{k,i} \) is 1 for the correct class \( k \) and 0 otherwise and \( \alpha \) is the smoothing hyper-parameter which is experimentally set to 0.1.
The label smoothing process replaces the hard one-hot encoded label with a smoothed label distribution. As opposed to assigning a probability of 1 for the true class and 0 for all other classes, label smoothing assigns a small probability to all other classes. In other words, the loss function is incentivized to learn a more diffuse decision boundary instead of penalizing the model for being entirely wrong about non-target classes.

Note that the algorithm described above does not need to be used to train the source domain’s model. Any model trained on source data can be adopted using the target model training algorithm that we propose below. Nevertheless, we recommend supervised contrastive learning pre-training, since it is easy to implement and leads to better generalization.

### 3.2.3 Adaptation Phase

**Weighted Information Maximization**

A source-free UDA methodology is not able to perform feature-level alignment since it is impossible to estimate the distribution of $p(X_s)$ without access to the source data. Instead, we propose to use Information Maximization (IM) loss [37, 29, 68], assuming that the ideal outputs should have similar characteristics to one-hot encoding but differ from one another. The IM loss is the result of the combination of two terms. The first term is entropy loss, which is described by a property that encourages the model to produce probabilities that are high in certainty, or, in other words, to assign one-hot encoding as the output.

$$L_{ent}(f_t; X_t) = -E_{x_t \in X_t} \sum_{k=1}^{K} \delta_k(f_t(x_t)) \log \delta_k(f_t(x_t))$$  \hspace{1cm} (3.3)

where $f_t(x) = h_t(g_t(x))$ is the $K$-dimensional output of each target sample.
Diversity loss is the next term that overcomes the trivial situation where all unlabeled data samples fall into the same category. While conventional source-free UDA methods [44] define diversity loss as $L_{\text{div}} = \sum_{k=1}^{K} \hat{p} \log \hat{p}$, we argue that the proposed objective cannot adapt to imbalanced datasets. Automatic sleep staging models generally suffer from an inherent class imbalance problem (CIP), which negatively affects the performance of the classifiers. There is an unequal duration of each sleep stage in a sleep recording. This issue exists in all the sleep datasets available, including the ones evaluated in this study. We have applied our method to the Sleep Heart Health Study1 (SHHS1)[82, 62], SHHS2, and Sleep-EDF datasets[20], which each comprise a distinct domain due to their different sampling rates and channels. Table 3.2 indicates the number of instances of each class found in these datasets. Accordingly, our preference for class balance is uniquely encoded in the empirical label distribution by employing custom weights for each class in order to account for the concept of class imbalance. We propose the following approach:

$$L_{\text{div}}(f_t; X_t) = D_{KL}(\hat{p}, p_{\text{prior}})$$

where $\hat{p} = E_{x_t \in X_t}[\delta(f_t(x_t))]$ is the mean output embedding of the whole target domain, and $p_{\text{prior}}$ is a custom uniform distribution with $K$ dimensions showing empirical label distribution.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>W</th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>REM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDF</td>
<td>8,238</td>
<td>2,795</td>
<td>17,725</td>
<td>5,676</td>
<td>7,678</td>
<td>42,112</td>
</tr>
<tr>
<td>S1</td>
<td>6,158</td>
<td>1,339</td>
<td>17,699</td>
<td>7,752</td>
<td>8,524</td>
<td>41,472</td>
</tr>
<tr>
<td>S2</td>
<td>11,244</td>
<td>1,607</td>
<td>17,746</td>
<td>5,436</td>
<td>6,463</td>
<td>42,496</td>
</tr>
</tbody>
</table>

Combining 3.3 and 3.4, the IM loss is:

$$L_{\text{IM}}(f_t; X_t) = L_{\text{ent}}(f_t; X_t) + L_{\text{div}}(f_t; X_t)$$

### Self-supervised Pseudo-labeling

While the minimization of both $L_{\text{ent}}$ and $L_{\text{div}}$ facilitates the adaptation of knowledge between unseen source domain and target domain by encouraging class diversity, it is not capable of eliminating the harmful effects of incorrect label assignment caused by noisy target prediction resulting from the domain shift. Aiming to mitigate this issue, we adopted a self-supervised pseudo-labeling strategy in order to ensure a better supervision of the target data encoding training as well as to obtain more accurate labels for the target data. Taking inspiration from [44] and DeepCluster [6], we calculate the cluster centroids induced by the source model for the target data according to the following procedure:

We begin by determining the centroid of each class in the target domain, as in weighted k-means clustering,

$$c_k^{(0)} = \frac{\sum_{x_t \in X_t} \delta_k(\hat{f}(x_t)) \hat{g}_t(x_t)}{\sum_{x_t \in X_t} \delta_k(\hat{f}(x_t))}$$

(3.6)
where $\hat{f}_t = \hat{g}_t \circ h_t$ indicates the previously learned target model. With the use of these centroids, it is possible to comprehensively analyze the distribution of different categories within the target domain. Using the nearest centroid classifier, we then predict the pseudo-labels,

$$\hat{y}^{(0)}_t = \arg \min_k D_f(\hat{g}_t(x_t), c^{(0)}_k)$$  \hspace{1cm} (3.7)

where $D_f$ measures the cosine distance between $\hat{g}_t(x_t)$ and $c^{(0)}_k$. At last, based on the generated pseudo-labels, we compute the final target centroids:

$$c^{(1)}_k = \frac{\sum_{x_t \in X_t} 1(\hat{y}^{(0)}_t = k) \hat{g}_t(x_t)}{\sum_{x_t \in X_t} 1(\hat{y}^{(0)}_t = k)}$$  \hspace{1cm} (3.8)

where $1(\hat{y}_t = k)$ is 1 if $\hat{y}_t$ and $k$ are equal, and 0 otherwise. We then compute the final pseudo-labels based on the new centroids:

$$\hat{y}^{(1)}_t = \arg \min_k D_f(\hat{g}_t(x_t), c^{(1)}_k)$$  \hspace{1cm} (3.9)

The pseudo-labels may then be updated multiple times using Equation 3.8 but the first update is sufficient according to [44]. Finally, the cross-entropy loss between the pseudo-labels and the labels predicted by the target model is calculated.

$$L_{pl}(f_t; X_t; \hat{Y}_t) = -\mathbb{E}_{(x_t, \hat{y}_t) \in X_t \times \hat{Y}_t} \sum_{k=1}^K 1(\hat{y}_t = k) \log \delta_k(f_t(x_t))$$  \hspace{1cm} (3.10)

To summarize, given the source model $f_s$ and the pseudo-labeling strategy explained above, our approach freezes the classifier from the source model $h_s = h_t$ and adapts the feature encoder $g_t$ with the full objective as:

$$\mathcal{L}(g_t) = \mathcal{L}_{ent}(g_t \circ h_t; X_t) + \mathcal{L}_{div}(g_t \circ h_t; X_t) + \beta \mathcal{L}_{pl}(g_t \circ h_t; X_t; \hat{Y}_t)$$  \hspace{1cm} (3.11)

where $\beta > 0$ is a balancing hyper-parameter.
Chapter 4

Experiments and Results

4.1 Datasets

For our experiments, we utilized three datasets: Sleep-EDF (referred to as EDF), SHHS-1 (S1), and SHHS-2 (S2), which represent distinct domains due to their varied EEG channels and sampling rates. For each dataset, we selected a single EEG channel to use in our experiments.

4.1.1 Sleep-EDF(EDF)

Based on the polysomnography (PSG) readings of 20 healthy subjects (10 men and 10 women), the Sleep-EDF-20 was derived[20]. Two studies were conducted with a total of two sets of subjects: age effect on sleep in healthy subjects (Sleep Cassette), which was conducted on healthy participants aged 25 to 101, and Temazepam effects on sleep (Sleep Telemetry). There were two scalp-EEG channels per PSG recording (Fpz-Cz, Pz-Oz) with a sampling rate of 100 Hz for each of the two datasets. In line with previous studies [15, 73, 58, 13], we adapted the Sleep Cassette data and used only the single Fpz-Cz channel for our experiments.

4.1.2 SHHS

The Sleep Heart Health Study (SHHS) [82, 62] is a multi-center cohort study examining the cardiovascular consequences of sleep-disordered breathing. A study is being conducted to determine whether sleep-related breathing is associated with an increased risk of coronary heart disease, stroke, all-cause mortality, and hypertension. There were a total of 6,441 men and women aged 40 years and older who participated in SHHS Visit 1 between 1995 and 1998. SHHS Visit 2 contains data on 3,295 of the S1 participants’ polysomnogram recordings. A parent cohort adjustment was made based on the outcome data. Both S1 and S2 datasets contain data from two EEG channels, specifically C4-A1 and C3-A2.

4.1.3 Data Preprocessing

Accordingly, we followed the setting of study [15] by selecting subjects from the S1 and S2 datasets in such a way that:

- Each dataset includes distinct patients.
- The S2 dataset’s subjects have a sampling rate of $250Hz$.
- Subjects were selected with an Apnea Hypopnea Index (AHI) $< 1$, which eliminates any bias towards sleep disorders and ensures consistent clinical status among all participants [17].

Table 4.1 provides a concise summary of the three datasets, outlining the number of subjects (Subjects) in each cross-domain, the selected EEG channel, the sampling rate, and the number of samples (Samples) in each cross-domain. In addition, the S1 and S2 datasets were downsampled to $100Hz$ so that the sequence length was equivalent to that of the EDF dataset.

In order to eliminate unknown stages, the data was analyzed, and then segmented into non-overlapping segments of 30 seconds ($T = 3,000$). Non-overlapping segments reduce data redundancy and simplify analysis. As a result of segmenting the data into non-overlapping 30-second epochs, each epoch can be treated as a distinct sample, allowing each epoch to be analyzed independently. In addition, it is a standard practice in sleep research to use 30-second epochs, allowing for the comparison of results from different studies.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Subjects</th>
<th>EEG Channel</th>
<th>Sampling rate</th>
<th>#Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDF</td>
<td>20</td>
<td>Fpz-Cz</td>
<td>100</td>
<td>42,112</td>
</tr>
<tr>
<td>S1</td>
<td>42</td>
<td>C4-A1</td>
<td>125</td>
<td>41,472</td>
</tr>
<tr>
<td>S2</td>
<td>44</td>
<td>C4-A1</td>
<td>250</td>
<td>42,496</td>
</tr>
</tbody>
</table>

Table 4.1: Description of Datasets

4.2 Implementation details

Following [15], we tested our methodology using PyTorch in the Google Colab platform on a Tesla T4 GPU. The datasets are randomly split into 60%, 20%, and 20% for training, validation, and testing. The validation part is used to tune the hyperparameters. To increase the robustness of the reported results, we conducted the training and validation process five times, using different random seed values for each iteration. Afterward, we calculated the average of the five results to obtain the final values. This method allowed us to reduce the influence of unintentional bias and present more reliable and precise outcomes. There will be a GitHub repository for the source code.

4.2.1 Feature Extractor Architecture

The EEG signals are first preprocessed into 30-second segments (i.e., epochs) that are then used to extract informative features. Following the design of the feature extractor proposed by
the feature extractor is composed of three convolutional blocks. The features from these three blocks are then fed to a fully connected layer, followed by a batch normalization layer and a dropout layer with probability of 0.5 to prevent the model from overfitting on the source domain. Using the following architecture, the model is trained:

- **Feature Extractor**: Three convolutional blocks followed by a linear block
  - Block1
    - Conv1d(1, 32, 25, 6)\(^3\) BatchNorm1d(32)\(^4\) ReLU(), MaxPool1d(2, 2, 1)\(^5\)
  - Block2
    - Conv1d(32, 64, 8, 1), BatchNorm1d(64), ReLU(), MaxPool1d(2, 2, 1)
  - Block3
    - Conv1d(64, 128, 8, 1), BatchNorm1d(128), ReLU(), MaxPool1d(2, 2, 1)
  - Block4
    - AdaptiveAvgPool1d
    - Linear(128, 128)\(^6\) BatchNorm1d(128), Dropout(0.5)

- **Classifier**: One linear layer
  - Linear(128, 5)

### 4.2.2 Network Hyper-parameters

Training the whole network is conducted through back-propagation, with a starting learning rate of 0.01 for training the source model and 0.001 for adapting the target domain. For the source domain training, we use mini-batch SGD, and for the target domain adaptation, mini-batch Adam, with momentum 0.9 and weight decay \(10^{-3}\). Additionally, we utilize the same learning rate scheduler \(\eta = \eta_0 (1 + 10 p_{\text{prog}})^{-0.75}\) as \([18]\), where \(p_{\text{prog}}\) is the training progress changing from 0 to 1. We set the batch size to 128 for all datasets. We determined the hyperparameter values using the validation set of the data and through grid search. Grid search is a machine learning technique used to find a model’s optimal hyperparameter combination. To perform grid search, a grid of possible hyperparameter values is created and then searched through these grids to determine the combination of hyperparameters that provides the best performance on a validation set. The weakening bound for data augmentation is set to 0.8 and the segment weakening percentage \(p\) to 10. Moreover, we applied \(\beta = 0.3\) and \(p_{\text{prior}} = [0.2, 0.05, 0.4, 0.15, 0.2]\) according to the duration of each sleep stage for all experiments. The pseudo-labels are updated epoch by epoch in the target domain and the maximum number of epochs is empirically determined to be 10.

---

\(^3\) Conv1d(in channels, out channels, kernel size, stride)
\(^4\) BatchNorm1d(num features)
\(^5\) MaxPool1d(kernel size, stride, padding)
\(^6\) Linear(in features, out features)
4.3 Experimental Settings

To assess the effectiveness of our model and baseline models, we utilized two metrics: classification accuracy ($ACC$) and macro-averaged F1-score ($MF1$). Below is a description of these two metrics:

$$ACC = \frac{\sum_{k=1}^{K} TP_k}{N}$$  \hspace{1cm} (4.1)

$$MF1 = \frac{1}{K} \sum_{k=1}^{K} \frac{2 \times Precision_k \times Recall_k}{Precision_k + Recall_k}$$  \hspace{1cm} (4.2)

$$Precision_k = \frac{TP_k}{FP_k + TP_k} \hspace{1cm} Recall_k = \frac{TP_k}{TP_k + FN_k}$$  \hspace{1cm} (4.3)

where $TP$ denotes True Positives, $FP$ denotes False Positives, and $FN$ denotes False Negatives. $K$ is the number of classes and $N$ is the number of samples respectively.

For further evaluating the performance of our method, we used a confusion matrix. Each row of the matrix represents instances in the true class, whereas each column represents instances in the predicted class, or vice versa.

4.4 Baselines

We compared our proposed model with a variety of baselines in order to evaluate it. First, three sleep staging methods were included in the Direct Transfer (DT) analysis. AttnSleep [13], SleepEEGNet [54], and DeepSleepNet [73] are the methods that have been tested. Furthermore, eight state-of-the-art discrepancy-based and adversarial-based domain adaptation baselines were adapted. Discrepancy-based methods include Deep CORAL [70], MDDA [64], and DSAN [87], while adversarial methods include DANN [19], ADDA [75], CDAN [49], DIRT-T [69], and ADAST [15]. Lastly, we compared our method with the source-free domain adaptation method, SHOT [44].

We also included the results of the Source-Only experiment, which represents our backbone network’s DT results. Moreover, in order to provide a fair comparison, we used our feature extractor for all eight UDA baselines and the SFUDA baseline. The baseline hyperparameters were tuned to maximize their performance. Below is a summary of these baselines.

- **DeepSleepNet** [73] uses CNNs to extract time-invariant sleep characteristics, and bidirectional-Long Short-Term Memory to learn the transition rules between sleep stages automatically.

- **SleepEEGNet** [54] utilizes a sequence-to-sequence deep learning algorithm with CNNs and bidirectional recurrent neural networks (BiRNNs).
4.5 Results

A comparison of various methods is presented in Table 4.2. A number of sleep stage classification and deep learning unsupervised domain adaptation methods, including DeepSleepNet[73],
SleepEEGNet [54], MDDA [64], DSAN [87], ADDA [75], CDAN [49], and DIRT-T [69], have been borrowed from [15]. In addition, we have reproduced and verified the reported performance of Attention Sleep [13], DANN [19], Deep CORAL [70], and ADAST [15] methods. Furthermore, this study developed and implemented source-only model, SHOT [44], and our methodology. The performance of direct transfer (DT) methodologies is generally lower than that of domain adaptation. DT experiments conducted on [73, 54, 13] indicate that domain shift causes significant degradation of performance and should be addressed separately. Moreover, in spite of the fact that the results of [13] are comparable or even better than conventional DA methods, the large number of parameters and modules used by this method shows that its potential has not been properly exploited. It should be highlighted that our Source-Only approach achieves relatively comparable results with UDA baselines, which indicates that the feature extractor has indeed been more generalized and is capable to handle cross-dataset variations effectively.

Table 4.2: Comparative Analysis of Various Baselines. Best results are in bold, and the second best are underlined.

<table>
<thead>
<tr>
<th>Baselines</th>
<th>Cross-Domain F1-score</th>
<th>UDA</th>
<th>Cross-Domain F1-score</th>
<th>SFUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepSleepNet [73]</td>
<td>49.19 ± 3.09</td>
<td>63.92 ± 2.35</td>
<td>SHOT [44]</td>
<td>56.30 ± 3.79</td>
</tr>
<tr>
<td>SleepEEGNet [54]</td>
<td>48.48 ± 2.74</td>
<td>63.24 ± 4.19</td>
<td>SHOT [44]</td>
<td>56.30 ± 3.79</td>
</tr>
<tr>
<td>AttnSleep [13]</td>
<td>67.37 ± 0.11</td>
<td>61.53 ± 0.82</td>
<td>SHOT [44]</td>
<td>56.30 ± 3.79</td>
</tr>
<tr>
<td>Source-Only (Ours)</td>
<td>62.45 ± 0.6</td>
<td>62.55 ± 1.64</td>
<td>SHOT [44]</td>
<td>56.30 ± 3.79</td>
</tr>
<tr>
<td>MDDA [64]</td>
<td>66.18 ± 0.73</td>
<td>61.49 ± 2.43</td>
<td>SHOT [44]</td>
<td>56.30 ± 3.79</td>
</tr>
<tr>
<td>DSAN [87]</td>
<td>59.22 ± 4.34</td>
<td>65.96 ± 4.57</td>
<td>SHOT [44]</td>
<td>56.30 ± 3.79</td>
</tr>
<tr>
<td>DANN [19]</td>
<td>74.97 ± 1.39</td>
<td>70.89 ± 1.39</td>
<td>SHOT [44]</td>
<td>56.30 ± 3.79</td>
</tr>
<tr>
<td>ADDA [75]</td>
<td>77.40 ± 0.54</td>
<td>75.95 ± 4.57</td>
<td>SHOT [44]</td>
<td>56.30 ± 3.79</td>
</tr>
<tr>
<td>CDAN [49]</td>
<td>69.32 ± 3.01</td>
<td>68.90 ± 3.69</td>
<td>SHOT [44]</td>
<td>56.30 ± 3.79</td>
</tr>
<tr>
<td>DIRT-T [69]</td>
<td>60.63 ± 1.30</td>
<td>55.55 ± 1.16</td>
<td>SHOT [44]</td>
<td>56.30 ± 3.79</td>
</tr>
<tr>
<td>ADAST [15]</td>
<td>74.97 ± 1.39</td>
<td>75.55 ± 1.64</td>
<td>SHOT [44]</td>
<td>56.30 ± 3.79</td>
</tr>
</tbody>
</table>

In order to address the domain shift problem for classification of sleep stages, domain adaptation becomes increasingly necessary due to the performance drop caused by domain shift in DT methods. According to the UDA baselines indicated in this study, DANN [19], DeepCORAL [70], ADDA [75], and MDDA [64] are three different global alignment methods that show less powerful results than DSAN [87], CDAN [49], DIRT-T [69], and ADAST [15]. This is due to the fact that [87, 49, 69, 15] all align class-conditional distributions, thus considering class distributions, which is crucial in our case given the class imbalance issues in sleep datasets.
4.6. Ablation Study

Further, we compare our method to SHOT [44], a source-free UDA approach. As expected, the results of [44] have decreased compared to other baselines. This is due to the fact that [44] treats all classes equally, which is not the case with sleep stage classification. Furthermore, [44] does not utilize source data or any statistics from it due to privacy concerns, which further limits its performance.

Despite being source-free, our proposed method outperforms all baselines in terms of average accuracy and F1-score, and obtains the best or second-best mean accuracy and F1-score in five of the cases due to the fact that our method takes class distribution into account, specifically imbalanced class distributions. In addition, it attempts to train a generalized source model, thereby facilitating more effective adaptation.

4.5.1 Confusion Matrix Visualization

To evaluate our model further in comparison to a source-only model and a source-free UDA method, confusion matrices were computed and visualized in Figure 4.1. It can be seen in Figure 4.1a and 4.1c that the source-only model (Figure 4.1a) has greatly benefited from the adaptation process. This model does not choose stage N1, as there is an imbalance in sleep datasets. Considering that N1 is a short sleep stage with only a few samples, the model ignores this class of data as it does not have a significant impact on accuracy and loss. Figure 4.1b illustrates the SHOT [44] model performance. According to this model, all classes are equal in size, which is not correct in sleep datasets. As a result, the model has performed poorly. For example, many samples were labeled incorrectly as N1, a rare class, and many of the samples belonging to stage N2 were labeled incorrectly, a more frequent stage that comprises more than one third of the entire dataset. The problem of imbalanced classes in sleep datasets has nevertheless been addressed by defining a weighted diversity loss in our methodology. Figure 4.1c depicts the confusion matrix of the proposed method applied to the target domain. It is evident from figure 4.1c, however, that even though our model has improved the situation significantly, the accuracy on the N1 stage still needs to be improved. It is due to the fact that this class has a much smaller number of samples than the entire dataset, and there is a great deal of overlap between this class and some others. By using data augmentation or oversampling, this issue may be less apparent. Nevertheless, due to the nature of this class, it is generally difficult to classify and separate it from other stages. As compared to the two matrices previously mentioned, the results demonstrate a matrix that is more similar to a diagonal matrix. The diagonal elements of a confusion matrix represent samples that were classified correctly, while other elements represent incorrectly classified samples.

4.6 Ablation Study

The objective of this subsection is to evaluate the effectiveness of the three modules in our framework, namely supervised contrastive pre-training, data augmentation, and distribution-aware diversity loss. The results are shown in Table 4.3.

Regardless of whether the other module was used or not, both supervised contrastive pre-training and weighted diversity loss improved performance. The average classification accuracy was increased from 57.40 to 67.01 by merely utilizing a weighted diversity loss. Although
the addition of the source pre-training module had no significant impact on the baseline model, it was clearly beneficial to the overall performance since it allowed the extracted features to be generalized. This approach can achieve the highest performance when all these components work together, and when they are combined, the source-only model and our approach can be effective.

4.6.1 Parameter Sensitivity Analysis for the Data Augmentation

As part of our investigation, we determined how different hyper-parameters within the data augmentation module affected target domain adaptation accuracy. The weakening percentage $p$ and weakening bound $\lambda$ were evaluated at a wide range of values. Figures 4.2 and 4.3 indicate
Table 4.3: Ablation Study Results

<table>
<thead>
<tr>
<th>Weighted $L_{adv}$</th>
<th>Source Pretraining</th>
<th>EDF $\rightarrow$ S1</th>
<th>EDF $\rightarrow$ S2</th>
<th>S1 $\rightarrow$ S2</th>
<th>S1 $\rightarrow$ EDF</th>
<th>S2 $\rightarrow$ EDF</th>
<th>S2 $\rightarrow$ S1</th>
<th>ACC</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td>×</td>
<td>47.32 ± 3.01</td>
<td>66.24 ± 1.73</td>
<td>61.39 ± 3.39</td>
<td>61.17 ± 1.96</td>
<td>55.73 ± 2.44</td>
<td>57.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>×</td>
<td>✓</td>
<td>55.53 ± 1.17</td>
<td>60.46 ± 1.50</td>
<td>60.26 ± 0.61</td>
<td>60.08 ± 1.31</td>
<td>59.91 ± 1.60</td>
<td>58.80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>×</td>
<td>65.82 ± 1.62</td>
<td>78.25 ± 0.64</td>
<td>67.10 ± 0.66</td>
<td>76.92 ± 0.12</td>
<td>67.53 ± 0.98</td>
<td>67.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>76.79 ± 1.67</td>
<td>70.23 ± 0.79</td>
<td>72.43 ± 1.55</td>
<td>77.53 ± 0.99</td>
<td>73.72 ± 1.65</td>
<td>74.95</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cross-Domain F1-score

<table>
<thead>
<tr>
<th>Weighted $L_{adv}$</th>
<th>Source Pretraining</th>
<th>EDF $\rightarrow$ S1</th>
<th>EDF $\rightarrow$ S2</th>
<th>S1 $\rightarrow$ S2</th>
<th>S1 $\rightarrow$ EDF</th>
<th>S2 $\rightarrow$ EDF</th>
<th>S2 $\rightarrow$ S1</th>
<th>MF1</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td>×</td>
<td>47.60 ± 4.79</td>
<td>45.94 ± 2.64</td>
<td>61.09 ± 2.02</td>
<td>56.60 ± 2.31</td>
<td>57.11 ± 1.91</td>
<td>52.37 ± 2.09</td>
<td>53.45</td>
<td></td>
</tr>
<tr>
<td>×</td>
<td>✓</td>
<td>48.30 ± 4.99</td>
<td>51.20 ± 2.28</td>
<td>55.61 ± 1.87</td>
<td>55.08 ± 0.45</td>
<td>55.56 ± 0.82</td>
<td>53.23 ± 1.38</td>
<td>53.16</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>×</td>
<td>56.85 ± 0.79</td>
<td>47.00 ± 3.44</td>
<td>67.16 ± 0.78</td>
<td>57.04 ± 0.69</td>
<td>66.14 ± 0.47</td>
<td>57.77 ± 0.49</td>
<td>58.66</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>65.44 ± 1.44</td>
<td>57.29 ± 1.08</td>
<td>69.38 ± 0.74</td>
<td>60.74 ± 1.21</td>
<td>67.44 ± 0.92</td>
<td>61.78 ± 0.79</td>
<td>63.78</td>
<td></td>
</tr>
</tbody>
</table>

that our approach performs relatively insensitively across various values, with a performance difference of less than two percent between different values.

Figure 4.2: Parameter Sensitivity Analysis for the Data Augmentation based on values of $p$

Figure 4.3: Parameter Sensitivity Analysis for the Data Augmentation based on values of $\lambda$
Chapter 5

Discussion and Conclusion

5.1 Discussion and Conclusion

Classification of sleep stages is the process of categorizing individual sleep stages according to physiological and behavioral characteristics. In general, there are five stages of sleep, each characterized by distinctive patterns of brain activity, muscle tone, and eye movement. With manual sleep stage scoring methods requiring a high level of time and labor intensity, automatic sleep stage classification is becoming more and more popular. There are, however, limitations regarding the domain shift and the availability of labeled datasets for classification modules designed specifically for this purpose.

Several factors may contribute to the differences in distribution of the train (source) and test (target) data in application, including differences in the hardware of clinics, the location of the sensors on the subject’s skull, and differences in the characteristics of subjects, such as the shape of their skulls [55]. This would result in a considerable domain shift that would negatively impact the performance of the classification model on the target data. Moreover, in order to train a deep neural network, a large amount of data is generally required. Accordingly, superior performance in determining sleep stages automatically is only obtainable with these models when the number of subjects is substantial, that is, when there are hundreds or thousands of subjects to analyze [61].

Unsupervised domain adaptation methods, or UDA methods in short, provide practical solutions for reducing the impacts of the above issues. Many studies have been conducted in the field of domain adaptation for sleep stage classification, most of which use adversarial-based techniques [15][55][86]. The idea behind this technique is to use a generative model and a discriminator to map data from the source domain to the target domain, such that the mapped data is indistinguishable from the real data in the target domain. As a result, the model is able to learn features that are domain-invariant, which would improve its ability to generalize to unseen, unlabeled data in the target domain. However, access to the raw EEG data from the source domain is necessary for these methods. This can give rise to privacy concerns, as EEG signals have a proprietary nature and can contain sensitive information.

The use of source-free UDA approaches is an effective way to mitigate privacy concerns while maintaining a generalized model. In this case, data from the source domain will not be used in the adaptation phase, and only a model pre-trained on the source domain will be
5.2 Applications

The project has a multitude of implications, both in academia and in industry. On the one hand, this project incorporates source-free UDA into the area of sleep stage classification, which provides the opportunity to further improve the methodology and apply it to large-scale datasets. On the other hand, low-cost wearable EEG recording devices have made it possible to better monitor patients and to conduct long-term monitoring more conveniently and accurately with such approaches. As they are able to utilize private datasets without directly accessing them, they are also able to utilize a greater amount of data for training, since the source domain data is not necessary to be stored on the device.

5.3 Limitations

In spite of the fact that our study provides an innovative approach to domain adaptation in sleep stage classification, it still has certain limitations.

In particular, our method uses a weighted diversity loss in order to take into account the class-imbalance nature of sleep datasets. However, the distribution preference is manually included in the loss. This distribution may work well when applied to healthy subjects but may have difficulty when applied to patients with specific medical conditions. It has been mentioned previously that the sleep patterns of certain groups of people differ from those of normal people. Thus, our model may not be the most suitable option in this instance.

5.4 Future Work

The current study would be enhanced by an exploration of the effectiveness of models that can capture temporal information, such as transition rules, in producing high-quality representa-
tions. This would be a valuable addition to the study’s previously mentioned limitations. In order to accomplish this, network architectures such as Convolutional Recurrent Neural Networks [10] and Transformers [77] can be utilized.

In addition, the multi-source domain adaptation approach may also be applied to this problem setting. Multi-source domain adaptation is a technique used in machine learning to train models that are capable of performing well on multiple source domains. Considering that each subject has unique characteristics and differences from the other, aggregating a set of models, each trained on one of the subjects, would enhance the generalizability and robustness of the approach.

The objective of our study was to solve the challenge of sleep stage classification by analyzing a single-channel EEG signal. Source-free domain adaptation can then be applied to a multi-channel EEG signal situation, or even to a multi-modal PSG sleep stage classification. A model can be more generalized by utilizing other formats of data, such as ECG, EOG, and multi-channel, in order to extract much more valuable information. As a result of combining multiple sources of data from various sensors or modalities, multi-modal PSG sleep stage classification has demonstrated superior accuracy and robustness to artifacts over single-modal PSG classification. The multi-modal classification of PSG can improve sleep stage classification accuracy by incorporating complementary information from a number of different modalities. As an example, combining EEG, EOG, and EMG signals can provide a more comprehensive picture of brain activity and muscle tone during sleep, which can help distinguish between different stages of sleep. Furthermore, single-modal PSG classification is susceptible to artifacts or noise in the signals, which can cause misclassification of sleep stages. As a result of integrating information from multiple modalities, multi-modal PSG classification can be more robust to artifacts and reduce the impact of individual artifacts.
Bibliography


Curriculum Vitae

Name: Yasmin Niknam

Post-Secondary Education and Degrees:
Bachelor of Science in Electrical Engineering
Minor in Computer Engineering
2016 - 2021
University of Tehran
Tehran, Iran

Honours and Awards:
Best Undergraduate Thesis Award from the University of Tehran
Ranked 8th out of 120 undergraduate students, University of Tehran
Ranked 111th among more than 200,000 participants in the Iranian National University Entrance Exam in 2016

Related Work Experience:
Machine Learning Researcher
Vector Institute
2021 - Present

Teaching Assistant
University of Western Ontario
2021 - 2022

Machine Learning Intern
Mitacs
2021 - 2022

Teaching Assistant
University of Tehran
2018 - 2021