Sleep Tracking on Electrooculography Devices using Artificial Intelligence and Data Processing Techniques

Mian Hamza

Department of Electrical & Computer Engineering

McGill University

Montréal, Québec, Canada

May 4, 2023

A thesis presented for the degree of Master of Science in Electrical Engineering

Abstract

The flexible PCB wearable device developed at our research lab calculates a single-channel EOG. We develop an infrastructure for our device, including an IoT structure to capture data and an iOS application. We develop an algorithm that can detect sleep stages using EOG data from our device. Previous attempts at classifying sleep always use data from double-channel EOG data. Initially, we used a labeled sleep dataset from the University of Wisconsin to train our neural network (NN). We then apply transfer learning to the sleep classifier with data extracted from our device. Overall, we were able to successfully create multiple classification models with data from the wearable device. Our Welch model with SMOTE obtained accuracies of 82.2 %, which is in line with the state of the art sleep stage classification.

Abrégé

Le dispositif portable PCB flexible développé dans notre laboratoire de recherche calcule un EOG à canal unique. Nous développons une infrastructure pour notre appareil, comprenant une structure IoT pour la capture de données et une application iOS. Nous développons un algorithme capable de détecter les phases de sommeil à l'aide des données EOG de notre appareil. Les tentatives précédentes de classification du sommeil utilisent toujours des données provenant de données EOG à double canal. Au départ, nous avons utilisé un ensemble de données sur le sommeil étiqueté de l'Université du Wisconsin pour entraîner notre réseau de neurones. Nous appliquons ensuite l'apprentissage par transfert au classificateur de sommeil avec des données extraites de notre appareil. Dans l'ensemble, nous avons réussi à créer plusieurs modèles de classification avec les données de l'appareil portable. Notre modèle de Welch avec SMOTE a obtenu des précisions de 82,2%, ce qui est conforme à la classification de pointe des stades de sommeil.

Acknowledgments

I am deeply indebted to Professors Sharmistha Bhadra and Zeljko Zilic for guiding me and helping me immensely throughout my research. I am also grateful to Shibham Debbarma for designing the electrooculography device.

I could not have undertaken this journey without my Parents, who supported me and encouraged me to pursue my passions. To my sisters Amna and Fatima and my brother-in-law Farhan, who made my time enjoyable throughout my Masters. Saim and Safa, who have made the last year memorable.

Lastly, I would like to thank my friends Jack and Shibham with whom I shared with the ebbs and flows of Graduate School.

Contents

List of Figures

List of Tables

List of Acronyms

AASM American Academy of Sleep Medicine.

AE autoencoder.

API application programming interface.

CNN convolutional neural network.

EOG electrooculography.

GRU gated recurrent unit.

ML machine learning.

NN neural network.

ReLU rectified linear unit.

RNN recurrent neural network.

SOTA state of the art.

Chapter 1

Introduction

Untreated sleep disorders can cause a variety of adverse health issues. This includes hypertension, stroke, cardiomyopathy (enlargement of heart muscle tissue), heart failure, diabetes, and heart attacks. Furthermore, due to poor sleep quality, impaired functioning can also occur. This can lead to more driving accidents, poor work/school performance, fatigue and weight gain [1].

The adverse effects of poor sleep are often overlooked; however, it is often the root cause of many problems people face. To address these problems, tracking sleep is an important step toward the diagnosis of sleep-related disorders.

It is important to address the gap of the large rate of undiagnosed sleep disorders; due to its potential for the betterment of lives. Tracking sleep helps many individuals understand the quality of their sleep and even obtain a prediagnosis of their sleep-related disorder. We

propose to develop an easy and inexpensive technology that can help people gauge their sleep. Tracking the different states of sleep allows us to determine how often patients are awake during the night, as well as the duration of each sleep state. The technology developed in this research can also be used as a screening technique to help medical professionals diagnose sleep disorders.

The premise of this thesis is to explore sleep models that use **electrooculography** (EOG) signals to classify sleep. EOG are signals that measure the movements of the eyes. The Integrated Microsystems Lab at McGill, developed a device that can read single-channel electrooculography signals from the eyes. It is a flexible Printable Circuit Board (PCB) that can be worn on the forehead. It will be referred to as the "Flex-EOG" device in this thesis. The purpose of the device was to help classify sleep and identify sleep-related disorders. This research enables us to create a low-cost, yet effective device that can successfully classify the sleep states.

The process of classification consists of creating a working sleep detection model from existing data. We then further train our model with data collected from our device. Then, the ultimate goal will be to update the models to work with the Flex-EOG device. This is a technique known as **Transfer Learning**. We need to use transfer learning to account for the variations in the circuit makeup of our device and that used in sleep studies. Therefore, our model needs to be calibrated to our device.

It is also noteworthy that an infrastructure for data collection is also developed using iOS devices. We also explore the feasibility of running sleep detection models on resource-constrained systems, such as mobile phones. Therefore, we research techniques to make our models resource friendly. We hope that the research conducted in this thesis can push forward the study of sleep.

Chapter 2 covers the background of EOG, what it is and the device that our lab has developed to record it.

Chapter 3 covers the basics of deep learning, advanced techniques used by state of the art models. It also covers techniques that compress the overall size of the model.

Chapter 4 covers a comprehensive overview of sleep detection. We discuss commercially available devices and patents for sleep stage detection devices. We also cover sleep detection research: deep learning models that were developed for sleep state detection.

Chapter 5 covers the experimental setup that we used to create our own sleep dataset. We run sleep trials using the Flex-EOG device.

Chapter 6 discusses the mobile app that is developed to collect sleep data.

Chapter 7 we design models and preprocessing techniques to detect sleep with EOG Data.

Chapter 8 covers the results and performance of the models designed in Chapter 7.

Chapter 9 is a Conclusion that covers an overview of the previous chapters and highlights possible avenues for further work.

Chapter 2

Background

In this section we will cover an overview of the different stages of sleep and their features. We also discuss information regarding EOG and the device created in the Integrated Microsystems lab.

2.1 Sleep Stages and Scoring

We will outline and differentiate the various types of sleep stages, and how they are scored in the literature. According to the American Academy of Sleep Medicine's (AASM) *"Manual for the Scoring of Sleep and Associated Events: Rules, Terminology, and Technical Specifications"*.

N1 and N2 are the initial stages of sleep where we are easing into our sleep states and our body is beginning to relax. N1 occurs as soon as we begin sleeping, and it is a very

short state of sleep, where we can easily wake up. N1 lasts around 10 minutes. N2 is a little longer, and occurs after N1 . Our bodies begin to relax during N2, and it is a transition state towards deeper sleep. Both N1 and N2 are light sleep stages and are characterized by **slow eye movements (SEM)**.

N3 is considered a deep sleep stage. It occurs after we have entered N1 and N2 sleep. During this sleep state is it harder to wake up, our breathing and heart rate decreases. Also, our muscles relax even further. During N3 our eyes are characterized by little to **no eye movement**.

N1-N3 are all characterized as non-rapid eye movement **NREM** sleep. Meaning that there is less eye movement in these sleep states in comparison to rapid eye movement (REM) sleep.

Rapid eye movement (REM) is the fourth and last state of sleep. It is characterized by random flickering movements of our eyes. They are "conjugate, irregularly, sharply peaked eye movements with an initial deflection lasting <500 msec". During this state we usually have dreams, and our heart rate and breathing increase, and our muscles tense up. REM sleep can usually last up to 1 hour.

Table 2.1 gives an overview of each sleep state. It is noteworthy that the duration of each sleep state varies drastically. Mostly we are in the N1, N2 stages of sleep, meaning that we are in light sleep most of the time. The differences in eye movements between the different sleep states allow us to classify them with EOG data alone.

Also, according to the same AASM manual, the differences between sleep states can be

sleep state	Eye Movement	Physiological Markers	Duration (% of Total Sleep)
N1	Slow Eye Movement	No distinct markers	5%
N2	Slow Eye Movement	Slow Pulse, Lower Body Temp., Relaxed Muscles	45%
N3	No Eye Movement	Slower breathing, Further relaxation of Muscles, Pulse, Body Temp.	25%
REM	Rapid Eye Movement	Irregular breathing, Tense Muscles	25%

Table 2.1: A summary of the characteristics of each different sleep state [2].

measured by brain wave activity using EEG signals [3].

2.2 Electrooculography

The electrooculogram\electrooculography (EOG) measures a potential difference between the cornea and the retina of an eye. This potential is related to eye movements, reflexes, and blinking. By measuring the change in the potential, we obtain an electrooculogram (EOG) signal. The magnitude of the EOG potential is correlated with the displacement of the eye from a neutral position during eye movement. Also, the amount of force that may have been used by the eye muscles during eye reflex and blinking activities [4].

The formal definition of EOG is: The electrooculography (EOG) is a voltage that is dependent on the movement of the eyes. It is calculated between electrodes placed near the eye at the inner and outer canthus. [5]. Usually in EOG measurements the recording

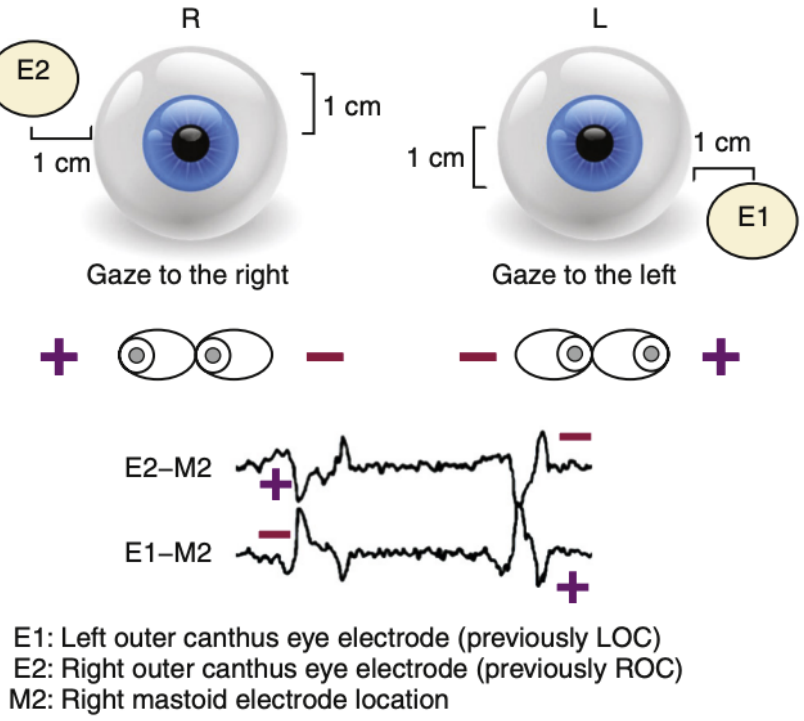


Figure 2.1: Electrode Placement for EOG recordings [6].

electrode placed on the left and right eyes are referred to as E1 and E2, respectively.

To measure the EOG, we use the difference between two electrodes, as shown in Figure 2.1. If the eye moves towards one electrode, it becomes relatively positive and the other eye becomes relatively negative. [6]. The EOG signals can be used to determine a wide variety of eye activities, whether it is blinking, winking, moving your eyes, and even sleeping.

For measuring sleep, EOG is used to detect movements of the eyes in different stages of sleep. When eyes move during sleep, they produce corresponding changes in the electrical field; this leads to a correlating potential change in the EOG electrodes.

2.2.1 The Flex-EOG device

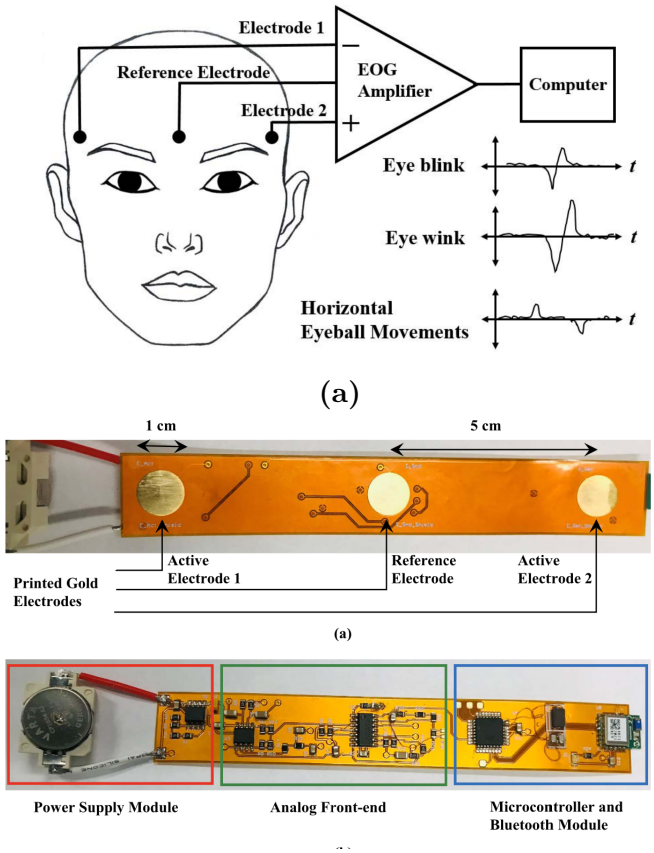


Figure 2.2: (a) The configuration of the Flex-EOG device, showing the placement on the forehead and the flow of data recordings. The graphs show how EOG signals looks like for different eye activities. (b) The three electrode configuration, the active electrodes measure the potential difference using the middle electrode as their reference. The other side houses the Analog/Bluetooth interface, used for to convert data into digital signals that can be sent over Bluetooth. [4].

We outline the function and design of the Flex-EOG device that was designed in the Integrated Microsystems Lab at McGill. It is used to collect electrooculography signals for

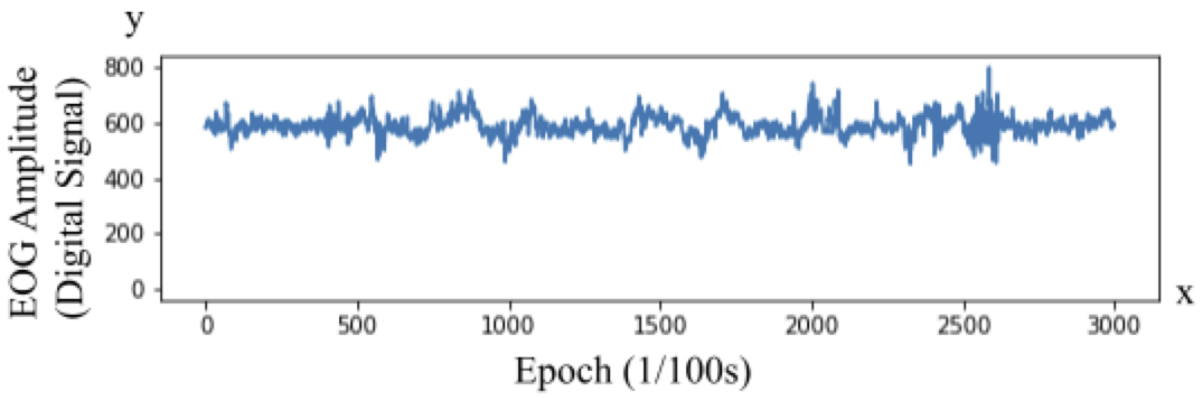


Figure 2.3: Sample of the raw Flex-EOG data.

our sleep studies. The functionality of the Flex-EOG device is shown in Figure 2.2. The eye biopotentials from the electrode are sent to the EOG amplifier circuit, after which the circuit communicates to any electronic device via a Bluetooth module. The Flex-EOG device outputs data as a single Channel EOG recording. The unit is a Digital Signal Amplitude, that has a range of 0-1023 depending on how strong the signal is.

In later chapters, we will cover the literature review of machine learning (ML) techniques and sleep detection models. Finally, in Chapter 7, we present our sleep detection model. The next chapter covers the mobile phone infrastructure that was set up to extract Flex-EOG data.

Chapter 3

Deep Learning & Associated Techniques

This chapter covers an overview of deep learning techniques used in classification, as well as the state of the art (SOTA) models that have been developed. The purpose of this chapter is to give an overview of the techniques used in this thesis. The techniques mentioned in this chapter are used in classification of sleep stages.

3.1 Historical Overview

Deep learning is a subset of Artificial Intelligence that uses neural networks (NNs) to process information. The idea of neural networks has been around since the 1960's, where a model was proposed that took inspiration from the took inspiration from the function of a biological

Neuron. Even today, the architecture of modern NNs is synonymous in function with neuron firing. There are two main architecture types that involve neural networks: CNN, RNN.

3.1.1 Convolutional Neural Networks

The architectures for convolutional neural networks (CNNs) are biologically inspired by human cognition. The structure of the process is derived from the way brain cells process vision. Fundamental concepts were first laid out by Hubel and Weisel in 1962. The structure began to resemble modern feedforward CNNs, in Neocognitron. Fukushima developed Neocognitron in 1980, with the explicit goal of mimicking human pattern recognition. The model created by Fukushima has many similarities with the modern CNN: it has similar layering and organization, it is hierarchical in nature, it can be used in unsupervised learning, and it is position-invariant meaning that the position of the input does not affect it. Although Neocognitron laid the groundwork, it wasn't until Yann LeCun introduced the backpropagation algorithm that modern day CNNs became more practical. Backpropagation allows CNNs to be trained in a simple, yet effective way. In 1989 LeCun et al. implemented a CNN that could handwritten zip code numbers that were provided by the US Postal Service. Backpropogation is the process of feedback in a neural network. We utilize the difference between our predicted output and our target output, and "backpropagate" that to the start of the network for our next iteration. Over multiple training cycles this allows our model to train effectively. This paved the way for future

techniques by LeCun, such as CNNs that use gradient descent. The new architecture, LeNet, would continue to be used by banks to read cheques. CNNs would not be widely used until 2012. The interest came from the creation of AlexNet. The publication of AlexNet in the paper: ImageNet Classification with deep convolutional neural networks, showed the classification prowess of CNNs by beating all previous records of object recognition in the ImageNet dataset [7, 8].

3.1.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are an architecture that is time dependent. Unlike CNNs, which are time independent, RNN can learn patterns that have a temporal aspect. This can be a powerful tool for temporally dependent data like EOG data in sleep detection models. The first semblance of recurrent neural networks were conceptualized by John Hopfield in 1982. Known as Hopfield Networks, it is not entirely like modern day RNN in that it does not process sequences of patterns. This is shown in Figure 3.1. It is a bidirectional complete graph. The bidirectional complete graph processes inputs from each neural unit in a different direction. The Hopfield Network in Figure 3.1 has 6 Neurons. The weight w_{ij} in the Network is the weight between the connections (i.e. synapse) of neurons i and j [9].

In 1986, David Rumelhart gave the blueprint for modern RNN. In his paper “Learning

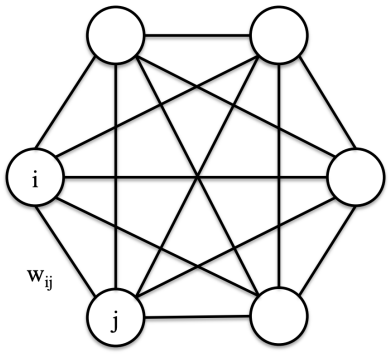


Figure 3.1: Hopfield Network, an early concept of a neural network [9].

representations by back-propagating errors", Rumelhart describes the use of back-propagation for networks that are neurone-like units. Rumelhart states that the goal of RNN is to design a self-organizing neural network, which "will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain" [10].

LSTM Networks, which is a derivative of basic RNN, were invented in 1997 by Hochreiter and Schmidhuber. The LSTM is an RNN with multiple selective memories. They are one of the most effective neural networks to date, and have set multiple records in the field of speech recognition, machine translation, and language modeling. They are able to process entire sequences of data, like speech, which makes them effective for data that is structured over time/sequentially. LSTMs can also be used in Electronic Health Records to detect anomalies in people's health, and for early diagnosis of diseases.

Lastly, another modern RNN configuration with a lower computational cost is "Attention".

Attention is inspired by how humans pay Attention, and the attention based networks have the ability to focus on the finer details of sequences, as well as process the macro patterns. They can process sequences longer than those of LSTMs. Along with an Encoder-Decoder structure, Attention can be an alternative to complex recurrent networks. Attention was first popularized in 2017, by Vaswani et al. in the paper "Attention Is All You Need" [11].

3.1.3 Autoencoders

An autoencoder (AE) is a type of neural network architecture derived from the standard feedforward network, that attempts to copy it input to its output. It is a network that is "trained to attempt to copy its input to its output" [12]. autoencoders (AEs) have been part of neural networks for decades. H. Bourlard & Y. Kamp, came out with a paper in 1988 titled "Autoassociation by multilayer perceptrons and singular value decomposition", this was the initial conception of an AE like structure created from multilayer perceptrons -which are used in traditional neural networks. In 1987 Yann LeCun wrote his PhD thesis titled "Modeles connexionistes de l'apprentissage" at the university of Paris VI; it was a technique that would learn through an auto-associative memory, i.e., memory mapping. Autoencoders were first envisioned as a means of dimensionality reduction and feature learning [13].

3.2 Components & Methodologies

The following section covers a succinct overview of RNN, CNN, AE and associated techniques.

3.2.1 Convolutional Neural Networks

Convolutional neural networks are able to process data that are grid-like in nature. For example, an image can be thought of as a 2-D grid of data, and a time-series data is a 1-D grid. Like in many different applications in Engineering, a **convolution** is used. Convolutions are popular in frequency domain analyses, like the Fourier transform. Before passing the data to a neural layer, the data go through layers of convolutions & related operations such as **pooling and batch normalization**.

Convolution Operation:

A convolution in machine learning is a linear operation where a kernel will be passed across the data. The kernel is usually a multidimensional array that is adapted from the learning algorithm and usually has the same dimension as the input data. Equation 3.1 shows the convolution equation that is used mainly in ML. The convolution is usually an integral for discrete-time calculations, but the values input into the computers are discretized, so the

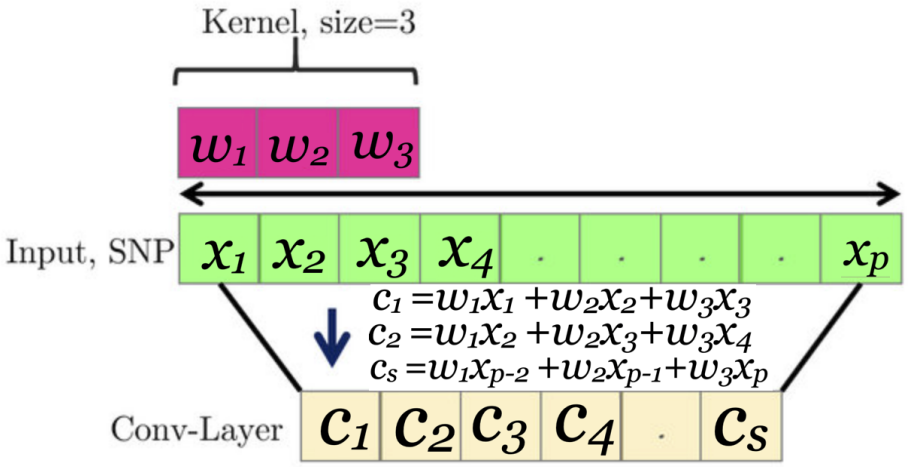


Figure 3.2: Example of a 1-D Convolution [14].

equation uses a summation instead [12].

$$s(t) = (x * w)(t) = \sum_{n=-\infty}^{\infty} x(a)w(t-a) \tag{3.1}$$

Figure 3.2 shows an example of a 1D convolution iteration. The output at a particular point will be the sum of the multiplication of the entire kernel with the corresponding data that it is transposed on. The kernel will slide across the whole dataset.

Pooling Operation:

Pooling is an operation that is used in conjunction with Convolution. The two main types of pooling are: max and average pooling. In both types of pooling, we run a kernel through our data. In max-pooling, we extract the maximum value of the data that the kernel is covering.

In average pooling, we average the pixels over the kernel. It is used to reduce the size of input data and to summarize an image or an array of pixels. By pooling, we extract vital information from the data and make the architecture invariant to the translations of the input [12]. Despite the utility of pooling, there is a debate about the usefulness of pooling in the architecture and whether pooling overcomplicates the architecture.

Batch Normalization

When training NNs, the distribution of input from each layer changes during training due to parameters of the previous layers changing. "This slows down training by requiring lower learning rates and careful parameter initialization", making it difficult to train models with saturating nonlinearities. [15]. Batch Normalization (BN) tackles this issue by normalizing the batches of output from network layers, with a standard deviation of 1, at little to no cost. It is a useful approach to perform reparametrization, and is a trainable layer that will fine-tune as training progresses over multiple epochs. It can be applied to multiple intermediate layers within a neural network.

3.2.2 Recurrent Neural Networks

Recurrent neural networks are a collection of networks that process sequential data. They are especially useful for time-dependent data. Therefore, recurrent neural networks (RNNs) are adapted to sequential data and can be trained on long sequences of data that would not

be practical on other networks.

Due to parameter sharing, most RNNs can process sequences of differing lengths. Recurrent neural networks can be built in many ways, but the underlying theme is that the function describing the neural node must contain recurrence.

Recurrence Equations:

$$s^{(t)} = f(s^{(t-1)}; x^{(t)}, \theta) \tag{3.2}$$

Equation shows an elementary RNN equation that is based on recurrence, where θ represents the parameters in the equation, and $x(t)$ -which is not necessary for a recurrent equation - represents an external signal that drives the system. The recurrence equation usually consists of hidden states $h^{(t)}$. As the network is trained, $h^{(t)}$ is used by the network to keep a rough summary of the "task-relevant aspects of the past sequence of inputs up to t" [12].

RNN Structure

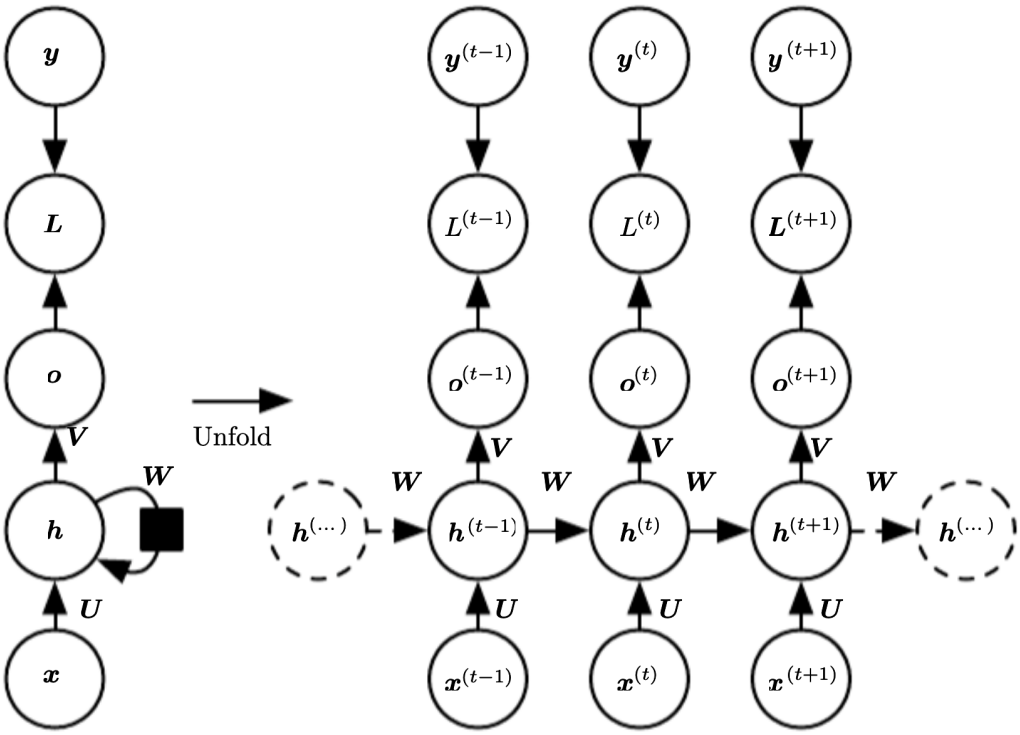
There are three basic types of RNNs:

1. Figure 3.3 a), Networks produce output at each time step, & hidden units have recurrent connections between them.

    - Hidden units can only be in the middle of an RNN; hence, they have to be connected to each other via recurrence.

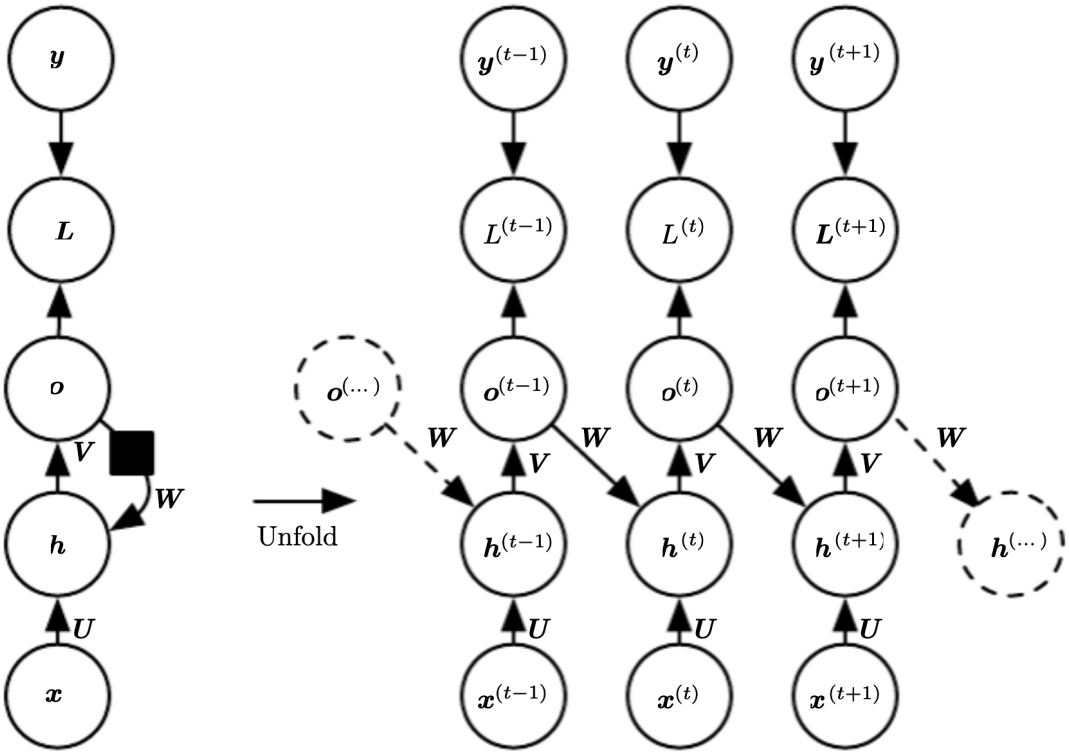
2. Figure 3.3 b), Networks that produce an output at each time step and have recurrent connections where the output at step $t-1$ affects hidden units at t

3. Figure 3.3 c), Networks parse the entire sequence and produce a single output.



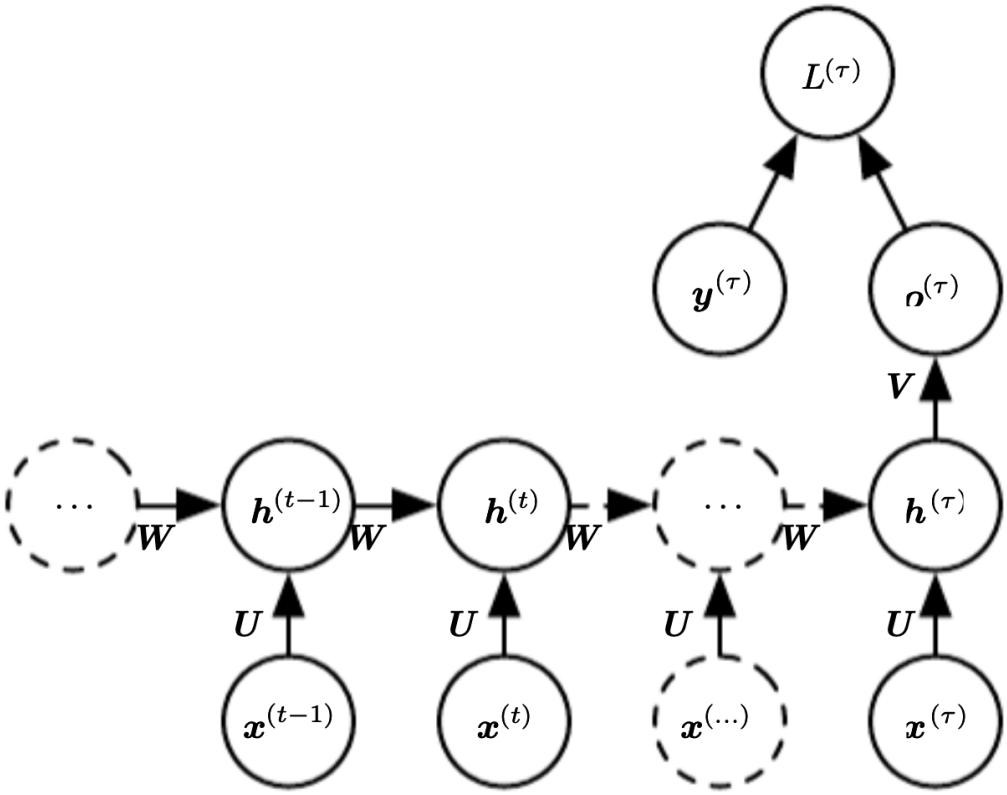
(a) $y = x$

Every input sequence x is fed into their corresponding hidden unit, h. The output o from each hidden unit is fed into the loss function, L. We compare each output o to its corresponding target y, which is also fed into the loss function.



(b) $y = 3 \sin x$

Similar to **Part a)** this RNN has a similar structure. For each sequence x we feed into the hidden units, and compare its output o to the target. However, one main distinction of this RNN structure is that the hidden units h, for each layer, are determined by feedback. The value of each sequence's hidden layer h's is determined by the output o of the previous layer.



(c) $y = 5/x$

This RNN has a single output o at the end of the Network. Each sequence's outputs from their corresponding hidden layer are also fed into the next sequences' hidden layers. Therefore, each intermediary output from each sequence's hidden layer is dependent on the previous layers' output. The output of the last hidden layer, is our final output. This is compared with the target using the loss function.

Figure 3.3: Three simple RNN structures [12].

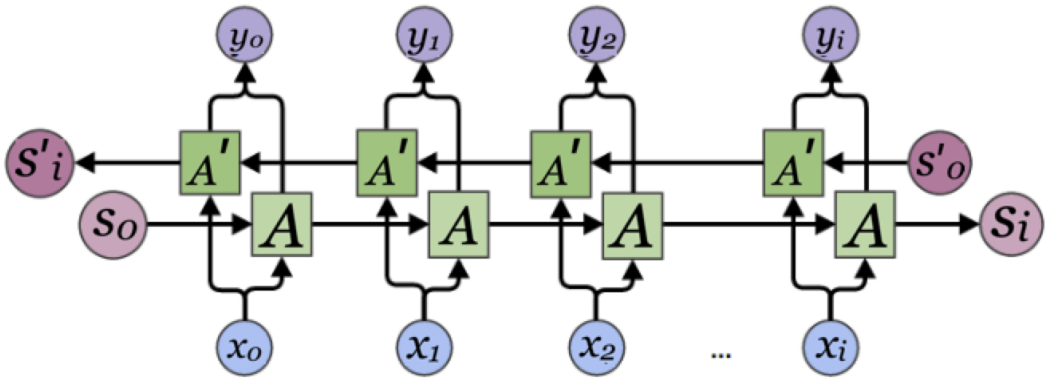


Figure 3.4: Structure of a Bidirectional RNN [16].

Another notable structure of RNN is the **Bidirectional** RNN Structure (BiRNN). As shown in Figure 3.4, the Bidirectional RNN structure has two structures:

1. Forward Time Direction

2. Reverse Time Direction

What the Figure 3.4, shows are basically two RNN structures, a forward (i.e. left) and a backward (i.e. right) structure that is zipped together. The inputs x_i, where i is a number denoting the number of nodes, are passed to A and A. The outputs of both A and A' are accumulated to form an output y_i for each node. We should also note that the A' node is not in the same position as A, rather it is the reverse position, i.e. the last node of A' will combine with the first node of A to form output y_0 [16].

BiRNNs have forward and backward recursions that are "zipped" together. They are an offshoot of basic RNNs. They are used in applications where sequences are closely linked

together. To understand these types of sequences, we must understand their surrounding sequences. For example, in speech detection where our interpretation of current sounds can depend on not only past sounds, but also future sounds. Bidirectional RNNs tackle this problem by also starting from the end of the sequence and training in reverse [12].

3.2.3 Autoencoders

A basic autoencoder has two different parts, the encoder function h and the decoder function r. The encoder function takes as input x, so the equation for the encoder is $h = f(x)$. The decoder function aims to create a reconstruction of the input, takes in the output of the encoder h, the decoder function is: $r = g(h)$. For autoencoders to be useful, the mapping of the input data to the encoder must not be 1-1 or 1-many. This means that the coded data h must have a dimension less than the input x, this is known as undercomplete autoencoders. With this autoencoder structure, the model is required to learn key features of the training data. These are known as undercomplete autoencoders, where we squeeze the model to smaller sizes to force it to select the most important features to learn [17].

Modern AE structures are known as stochastic autoencoders. Here, the encoder and decoder functions are conditional probability distribution functions, rather than deterministic mappings.

The equations for the encoder and the decoder are 3.3 & 3.4 respectively:

$$p_{encoder}(h|x) = p_{model}(h|x) \tag{3.3}$$

$$p_{decoder}(x|h) = p_{model}(x|h) \tag{3.4}$$

3.3 State-of-the-art Architectures

This section talks about state of the art architectures and associated techniques that can be employed.

3.3.1 ResNet

The ResNet architecture is useful in convolutional neural networks. As networks get deeper, the accuracy of the model saturates and then begins to degrade rapidly. He et al. in [18] came up with a solution in their residual neural network (ResNet) architecture. A ResNet is based on residual learning. In the degradation problem, it is suggested that the optimization routines have problems in approximating identity mappings by multiple nonlinear layers. In residual learning the weights, if the identity mappings have reached an optimal state, the optimization routines can drive the weights of the multiple nonlinear layers towards zero. This causes the solutions to approach the identity mappings.

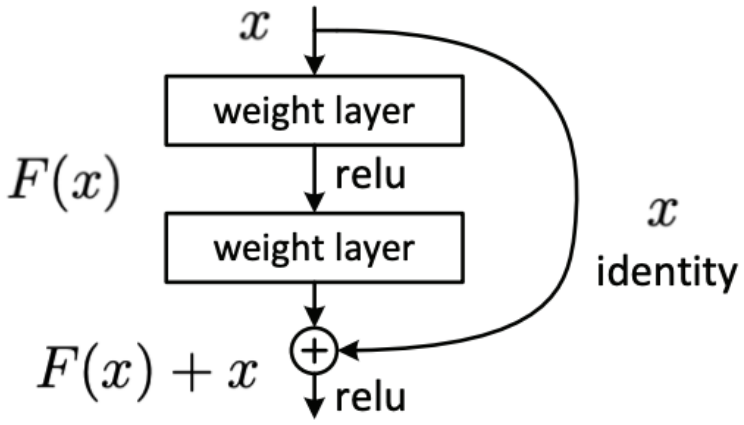


Figure 3.5: Residual Block: We see that inputs from a previous point x are summed with $F(x)$ to create $F(x) + x$, where x is applied to some weight layers. [18].

A ResNet consists of residual blocks. Each residual block consists of two things: a convolution block and an identity loop.

In Figure 3.5, the weight layer is the convolutional module, and the identity loop will be the shortcut connection that performs an identity mapping. A shortcut connection is one that skips one or more layers. Their outputs are added to the stacked layers. The computation does not add additional parameters or computation complexity.

The objective of the residual block is to allow gradients a "backdoor" to back-propagate through the network. This optimizes the learning process by avoiding dead gradients, which allows for deeper models. This is because small differences between images have to be backpropagated through the network as a gradient. The gradient will begin to shrink as it back-propagates through the network, and can eventually vanish. This leaves us with a useless gradient.

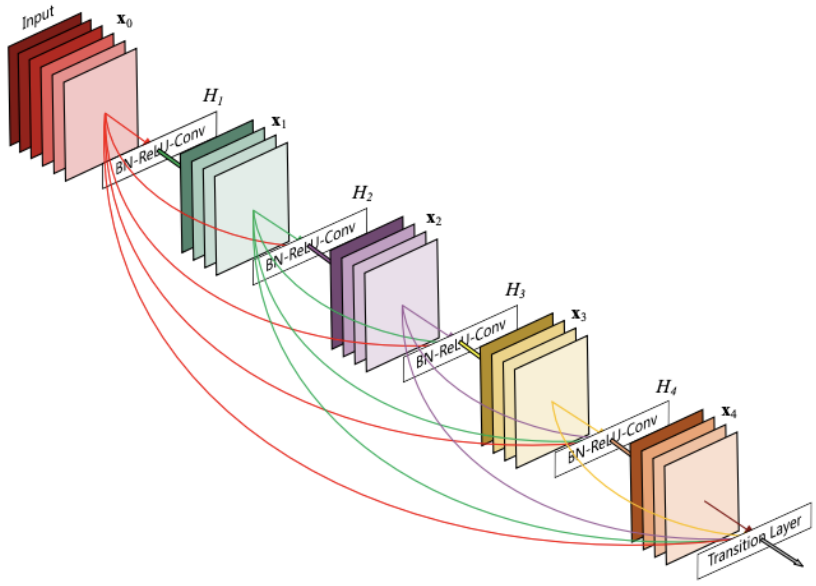


Figure 3.6: ML model with dense connections: We see that all outputs from previous dense blocks are passed as input to transition layers, H_n. The H_n layers process, and concatenate these values and pass them to the subsequent block, x_n. [19].

Residual blocks mitigate this issue by allowing for a shorter path back to the start of the network [12, 18].

The structure of ResNet is not important; the main benefit of the architecture is the idea of using shortcuts to connect across different convolutional layers. The shortcuts are easy to implement and can easily be applied to multiple CNN architectures.

3.3.2 DenseNet

DenseNet won the best paper at CVPR 2017 (Conference on Computer Vision and Pattern Recognition) and was noted to have better accuracy in ResNet. Since DenseNet was created after ResNet, some concepts borrowed and similarities between the two architectures are observed.

Just like the ResNet architecture, the actual DenseNet model is not important, the main contribution of the DenseNet architecture is the Dense Connectivity Structure. Similar to ResNet, the DenseNet architecture uses shortcut connections, just like in residual blocks. However, unlike the Residual Block, each dense block has connections from every previous feature map in the network, as shown in Figure 3.6. Also, unlike ResNet, all these previous connections are concatenated and passed forward through the network. In ResNet we sum weights from previous connections instead of concatenating them. The importance of identity connections in ResNet inspired the authors of DenseNet to create an extended idea that improved gradient flow through the network. The benefits of DenseNet include improved feature reuse, reduced redundancy, and thinner layers, which leads to improved parameter efficiency. The authors also discuss a regularizing effect from the architecture, which performs better on smaller datasets.

Growth Hyperparameter

Due to the accumulation of prior feature maps at each layer, the network will grow gradually as each previous block adds their feature maps. The authors define a growth rate hyperparameter, k. k sets the rate at which new information is added to the network and is also used to set the layer width, i.e. the number of filters.

Dense Blocks

DenseNet consists of "Dense" blocks which have a width k, and three operations: BN, ReLU, and 3x3 convolution. Dense blocks also have bottleneck layers in order to reduce dimensionality, and consists of a 1x1 convolution layer of width $4k$ that is applied before the activation function. The block then has a regular 3x3 convolution.

Transition Layers

There are transition layers between dense blocks. They downsample feature maps by a factor of: $0 < \theta \leq 1$ compression. This means that for every x feature maps the number will be reduced to θx. The reduction of feature maps is necessary to prevent the model from growing at an alarming rate, causing infeasible training and computational costs.

Transition layers consist of a 1x1 convolution layer that outputs θx feature maps, where θ is the compression factor given x input feature maps. A 2x2 average pooling layer of stride 2 follows. The pooling operation reduces the depth, height, and width of the input [12, 19].

3.3.3 Baidu DeepSpeech 2

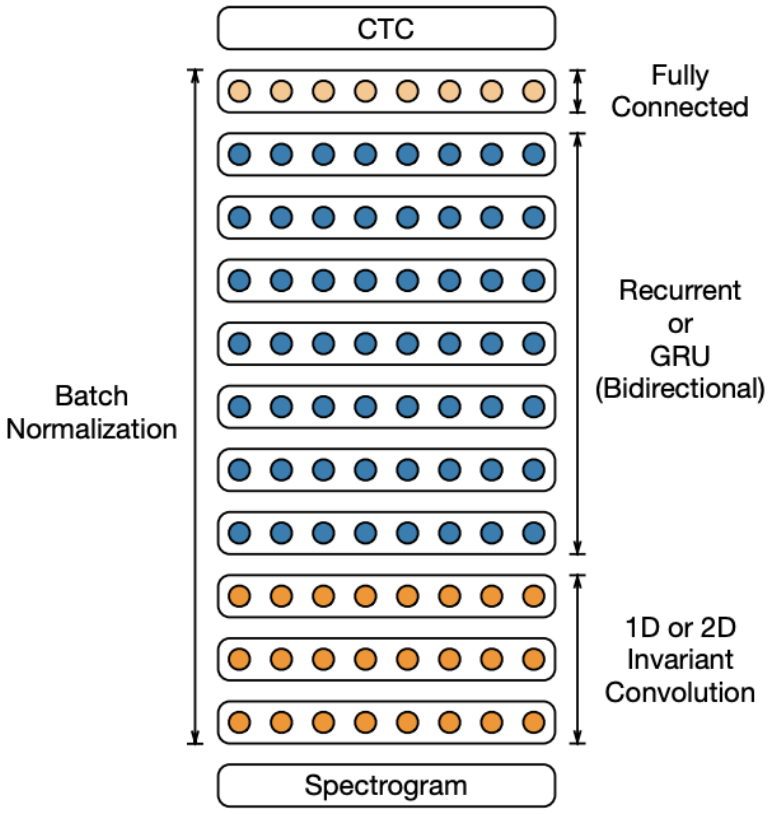


Figure 3.7: The model architecture of Baidu DeepSpeech 2 [20].

Baidu Deepspeech 2 is a speech detection model that pioneers the use of spectrograms in image detection. Spectrograms represent the band of frequencies of a signal over time. They are discussed in more detail in sections: 3.5.1, 7.3.4. Since then, spectrograms have become a cornerstone for end-to-end speech recognition and Natural Language Processing (NLP). However, unlike previous ResNet and DenseNet, DeepSpeech 2 uses a combination

of CNNs and RNNs in its model.

As shown in Figure 3.7, the model (which starts from the bottom to the top) takes as input a spectrogram. Then it is followed by multiple convolution layers. This portion helps process the data and to reduce the data into a matrix of key features. That matrix is fed into a Bidirectional RNN that has between 1 and 7 layers.

The function of the BiRNN is to learn sequential patterns in the features that were extracted from the previous CNN layers. It is important to note that batch normalization is applied throughout the model. Since the RNN layers will have many computations, batch normalization ensures that the training phase is stable. This is done by normalizing outputs of our model to ensure that we do not shrink our weights to extremely small values or that some of our weights increase beyond a reasonable limit. By reducing the gap between the large and small values, we ensure that they have an equal effect on future output. This is known formally as "reducing internal covariate shift" [15].

After the BiRNN layers, the model contains a fully connected network; These are standard network nodes that are typically included in every machine learning model. Finally, the model has a CTC layer. This is known as "connectionist temporal classification", and is a type of neural network model that is used heavily in speech and text recognition.

The key takeaways from this model are the following: Using CNNs and RNNs together we are able to extract information from spectrograms. The CNNs process and extract key features, BiRNN Layers then learn the sequences present in the spectrogram.

3.4 Transfer Learning

Transfer learning is when we exploit what we have already learned in one setting to improve "generalization in another setting." For example, a ML model that is trained to identify bicycles can be utilized to improve a model that needs to identify motorcycles. [12].

Transfer learning is a technique that uses knowledge transfer techniques to train machine learning models. We can describe transfer learning in two simple steps: initially, we train our models using data similar to our target dataset. We can also obtain a pretrained model for this step, and then we train our model on our goal dataset.

This is done for a variety of reasons. Firstly, Our goal dataset is too small to train an effective model on its own, so we effectively borrow a model that has already been trained on plenty of high quality data. Second, it is to add data variance; often times we take data from one or two studies. There may be biases that arise by taking data from limited resources, and it can be too costly or difficult to obtain more data. We can use transfer learning to add diversity by obtaining a model that has been trained from different sources. Lastly, it saves allot of computing resources. Training machine learning models requires allot of energy and processing power. Thus, large organizations with access to computing resources can train large and powerful models. These models can then be utilized to train more specialized classifiers by smaller research groups. It benefits researchers who lack access to high-performance computing resources because they can reuse these models. By reusing models researchers are starting off with a high quality model as their base. This also allows

for faster training times since models are more likely to reach their best performance with less epochs.

Transfer learning has been shown to improve model performance. Side-by-side comparisons show that models reach higher accuracy and help models reach state of the art [21].

3.5 Techniques for Model Compression

There are a few key techniques for model compression. Several of the main model compression techniques were compiled and published in the paper Deep Compression in 2016 by Han et al. [22].

The Deep Compression paper highlights a few key techniques for model compression: pruning, and trained quantization. Huffman coding is also used, but it will not be discussed in the context of this thesis.

Pruning

Pruning a neural network is a simple way of reducing the number of weights that propagate through the neural network. Pruning was first proposed as a way to reduce model complexity and overfitting. Consequently, many research papers have been published (LeCun et al., 1989 [23]; Hanson & Pratt, 1989 [24]; Hassibi et al., 1993 [25];).

In the Deep Compression paper, Han et al. propose an approach to pruning that does not cause the degradation of model accuracy. The approach follows a three step process: train

a model through normal training, prune all the connections with weights below a certain threshold, and retrain the pruned network to fit the data.

Step 1 allows the model to form connections and weights. It is important to have a strong baseline model to start with. Then we can selectively choose the most relevant weights that have the most power on our classification.

In step 2 by pruning we reduce model complexity because weights that do not contribute much to the final prediction of the network are removed.

Finally, by retraining the network with the remaining weights, we are fitting them more aptly to the final result. Since we already pruned the previous weights, we need to tweak the remaining network to adapt to the different structure. This step ensures that accuracy remains the same.

Trained Quantization and Weight Sharing

Trained quantization can further reduce the model weight by reducing the number of bits that represent each weight. For example, say a weight is represented as: 2.0849, which would be represented as a float of 32 bits. We could instead represent the same weight as 2 which can be represented as a two bit unsigned integer (uint). By applying quantization to all the weights across the network, a significant amount of space can be saved because neural networks can have millions of weights. For example, AlexNet, a well-known neural network classifier, has 61 million weights [7].

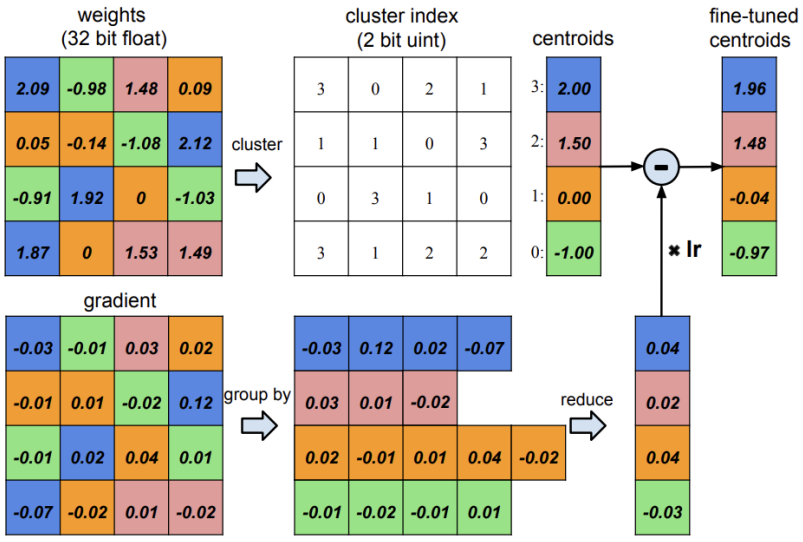


Figure 3.8: Weight sharing after quantization (top), and grouping the weights into centroids for training models (bottom) [22].

After weight quantization, weight sharing can further compress the model. Weight sharing occurs if we cluster similar weights together. We can use popular clustering algorithms such as k-means [26]. This way we can reduce the amount of weights we use altogether and calculate the gradients by using a single weight per cluster. This is illustrated in Figure 3.8 .

Advances in Deep Compression

Since the inception of Deep Compression as a compression pipeline, other implementations have focused on altering the underlying algorithms behind Deep Compression. Many different pruning algorithms have been developed, i.e. token pruning, a greedy approach. There have also been different clustering techniques that have been implemented, other than k-means.

Calculating Compression

Data Compression is calculated by the following metric. $z = \frac{S_u}{S_c}$, and $m = 1 - \frac{S_c}{S_u}$

Where z is the Data Compression Ratio, S_u is the uncompressed size of the data, and S_c is the size of the data after it has been compressed. m is Space Saving: the space saved as a result of compression. Space Saving is measured in proportion to the uncompressed size.

3.5.1 Other Compression Techniques

There are other compression techniques that are used in the model pipeline to address cost limitations in edge devices that have limited resources.

Gradient of Pixels

Boris Murmann talks about using a pixel gradient rather than a grid of pixels for image classification. This allows us to reach a high classification rate but with "low bit depths" [27].

The Histogram of Gradient, a classical computer vision technique, highlights the gradient orientation in localized sections of the image. HOG is similar to other computer vision techniques, such as the Canny Edge detector, SIFT. Unlike other features, HOG measures the magnitude and orientation of the gradient. For this reason, it is a good candidate for image classification. Given this information, the HOG gives the computer enough information about the nature of the image but also reduces the image size in the process.

Methodology: Below is a brief summary for HOG, for more information, please refer

to [28]

HOG is a simple process that has a few steps:

1. Calculate the Gradient of the Images:
    - Can be done with Sobel Operator, and this is done in the x, y directions.
2. Find the Magnitude and direction of the gradients:
    - $g = \sqrt{g_x^2 + g_y^2}$ and $\theta = arctan(\frac{g_y}{g_x})$, where g_x and g_y are the gradients calculated in the first step in the x & y direction respectively.
3. Create a histogram of the gradients in cells of 8 times 8:
    - This can be done in multiple ways, one technique is to have bin sizes of 20 (9 total bins). Each bin will have the added magnitude of the pixel, which corresponds to the pixel whose angle fits in the bin; here, we divide the image into 8 times 8 blocks, but the blocks can be of different sizes.
4. Normalize the gradients in a 16 times 16 cell:
    - We combine four, 8 times 8 cells into a larger cell, and we then normalize the vectors

Spectrograms

A spectrogram is a graph that can visualize the change in frequency over time. It is fairly popular in natural language processing to detect different speech patterns.

Baidu developed a technique that can split up a spectrogram of a voice recording into different words [20].

Recently, spectrograms have seen an emergence in medical AI. When conducting research, it was discovered that spectrograms can also be very useful for data compression in machine learning models. Since time-series data is stored at a rate of 32 bits per recording, high rates of data storage can add up.

In medical devices such as the EOG and even the EEG bands, the device measurement rate is around 100-200Hz. So, for a 30 second interval, there can be up to 6000 measurements. However, a spectrogram offers a way out, by allowing us to compress multiple epochs of data into a single image.

Convert to Grayscale

Figure 3.9: Size difference between Regular image and grayscale [29].

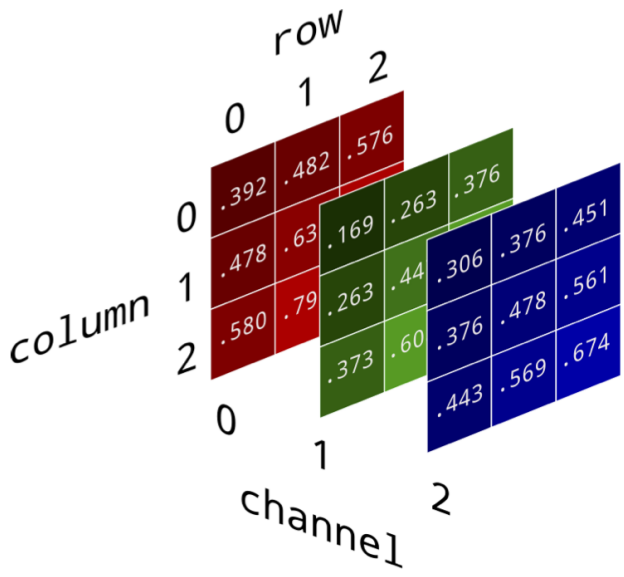


Figure 3.10: The three channels of a color image [30].

By converting an image to grayscale, we are saving a few key things. Space taken up by the image, and the amount of processing power required to train and run the machine learning model. Please note that the amount of storage the image takes up may not necessarily decrease and is dependent on the file storage format. As shown in Figure 3.9, we can decrease the space taken up by our image by removing the color channels. Figure 3.10 shows the three color channels seen in RGB images. We can compress the size of our data by a compression factor of three.

Along with size compression, our model based on machine learning also has a lower computational load. This is due to the fact that CNNs we now use is one dimensional as opposed to two dimensional. The CNNs we train our images on have to compute convolutions on single-channel grayscale images. This is in contrast to the CNNs that run on color images. They are 2D CNNs that have to run convolutions on three color channels. RNNs also have a third of the information to process from grayscale images. Our overall computational cost will benefit from converting our model to grayscale.

Further Compression of a Spectrogram

There are two techniques that can be applied on top of a spectrogram to further compress the data. These techniques are used in speech recognition.

Filter banks (FBANKS) are time-frequency representations computed by applying a set of filters to the spectrogram of a speech signal. Each filter is triangular and has a response of 1 at the center frequency. The filters are designed to be equally spaced in the Mel frequency domain. It is possible to pass from the linear frequency domain to the Mel one.

Mel-frequency cepstral coefficients (MFCCs) are applied on top of FBANKS. They are computed by applying a discrete Cosine transform (DCT) on the top of the FBANKs. DCT is a transformation that decorrelates features and is used to further compress them [31].

Chapter 4

Sleep Detection

There have been multiple studies in the past to determine sleep states using biosignals such as EOG and EEG. Recently, many different devices such as the Google Nest Hub, and wearable devices like the Apple watch, have implemented sleep tracking.

4.1 Commercially available devices

There are many different types of sleep detection devices. Some track your biopotential signals, such as your brain waves and eye movement. In this category, we see devices like the Muse, which gives sleep states from EEG signals. We will give a brief overview of sleep stage detection devices that use different techniques for sleep tracking.

4.1.1 Muse EEG Device

According to the AASM manual on sleep stages, differences between sleep stages can be measured by brain wave activity using EEG signals. For example, EEG signals from N1 sleep have low-amplitude mixed frequency activity that ranges from 4-7 Hz. A summary of the different brainwave activities is shown in Table 4.1. By treating N1, N2 as one sleep state: light sleep, we can see that each state of sleep has different types of brainwave activity. Thus, like EOG, EEG can also be used as a standalone signal to determine the different sleep states of a person. Section 4.2.2 goes over a research paper that has successfully been able to train an ML model to recognize sleep states from the different brainwave patterns mentioned in Table 4.1.

sleep stage	EEG Brainwave activity
N1	Low amplitude, predominantly 4-7 Hz
N2	Low amplitude, predominantly 4-7 Hz
N3	Slow wave activity, Waves of frequency 0.5Hz-2 Hz and peak-to-peak amplitude >75 nano Volts, measured over the frontal regions
REM	sawtooth waves, often serrated, 2-6Hz

Table 4.1: Shows the characteristics of Brainwave activity that occur during different states of sleep.

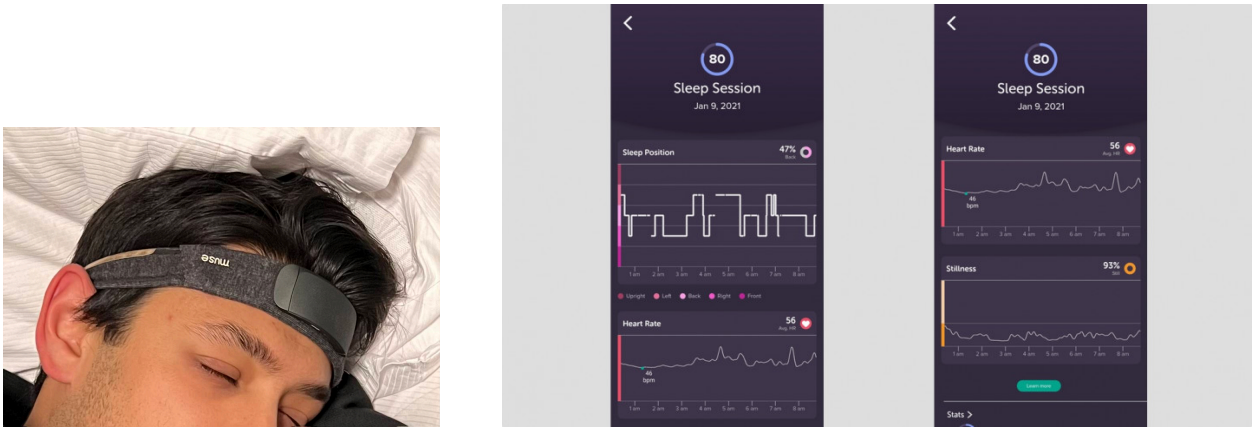


Figure 4.1: Muse Headband functionality and interface [32].

As listed above, the different stages of sleep have different EEG signals. Muse and other EEG sleep trackers use these signals to track your sleep. Depending on the device, different algorithms can be used, such as deep learning and statistical machine learning [33].

Figure 4.1 displays an overview of the functionality of the Muse Headband. Similar to other EEG devices, it is worn on the forehead and wraps around the back of the head. There is also a mobile app that gives different user metrics, such as heart rate, sleep states, and position. However, note that Muse has not published any academic research papers to support its algorithms.

4.1.2 Google Nest Hub

Other devices track body movement, position, and sleep sounds. There are many commercially available devices in this category; The Google Nest Hub uses a radar sensor to track your movements. This millimeter-wave sensor emits a radio wave and uses the reflected signal to figure out if someone has moved; their velocity and distance. The data from the radar sensor are trained on a machine learning model to track sleep [34]. Figure 4.4 shows the accuracy of the Radar. The black part of the graph shows a reflected signal. 4.4 d) shows that it is capable of detecting changes in chest size when breathing.

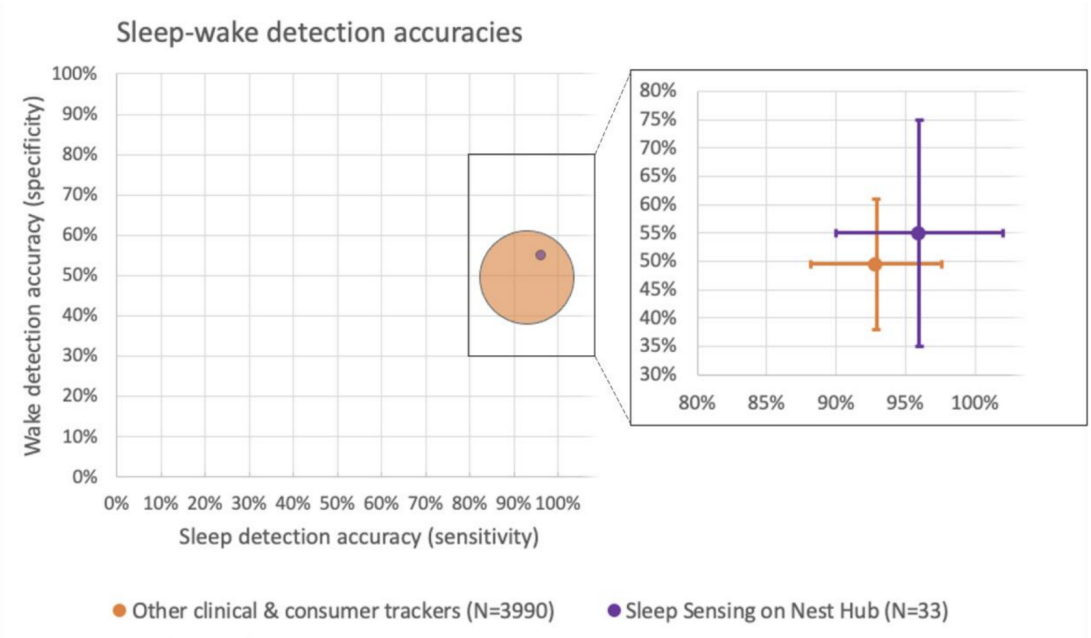


Figure 4.2: Comparison of Google Nest Performance with other Commercial Devices. Google Nest Hub has a sleep detection accuracy around 96%. Other Commercial devices have accuracies of around 93%. Wake detection is also better on the Google Nest Hub [34].

	Algorithm performance
Sensitivity	0.96±0.06
Specificity	0.55±0.20
Accuracy	0.87±0.06
PPV	0.88±0.07
NPV	0.86±0.17

Figure 4.3: Nest Hub Performance for sleep tracking [34].

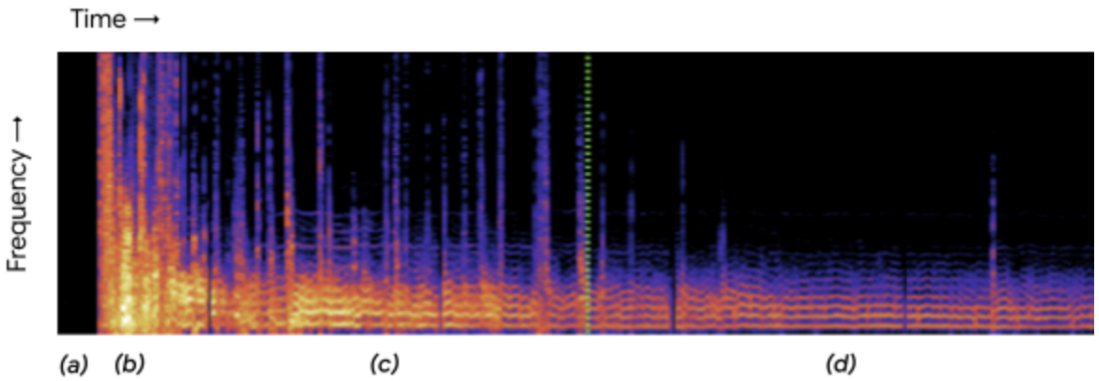


Figure 4.4: A spectrogram showing movement that is picked up by the Google Radar Sensor. (a) an empty room (no variation in the reflected signal shown by the black parts). (b) large pose changes. A large amount of the signal is reflected, a large range of frequencies are measured. (c) brief limb movements. Less variance of frequencies than in b). (d) small chest and torso displacements from respiration while at rest. Only the lower range of frequencies are picked up, but unlike an empty room, movements are still picked up, i.e. the spectrogram is not completely black in this portion [34].

Using the radar sensor, over a million hours of radar data with sleep recordings, Google is able to build a sensor that beats other commercial sleep trackers. Google creates a ML model capable of processing the 3D tensor data from the radar. Figure 4.3 shows their performance, it has an accuracy of 87%, which is in line with alot of state of the art accuracies. The place where the Nest really performs is its Sensitivity of 96%, which means that the model is good at identifying the state of sleep of the participants; however, the model has a low specificity of 55%, this means that there are many false negatives. Having false negatives in sleep stage classification is very common.

As seen in Figure 4.2, the Google Nest performs better than other commercial sleep classifiers.

4.1.3 Apple Sleep Device

Another similar device that tracks movement has recently been patented by Apple. Shown in Figure 4.5, it is a sheet that works by tracking user movement.

It is a "layered sensor having multiple laterally adjacent substrates in a single layer". The sensor is piezoelectric, meaning that it generates electric signals in response to pressure. It is placed underneath the user like a bed sheet, and can track their movement since the piezoelectric sensor will be sensing changes in pressure throughout the sheet.

Although Apple has not yet released a sleep detection model, the idea is similar to the Google Nest Hub. Both function by tracking movement to detect sleep.

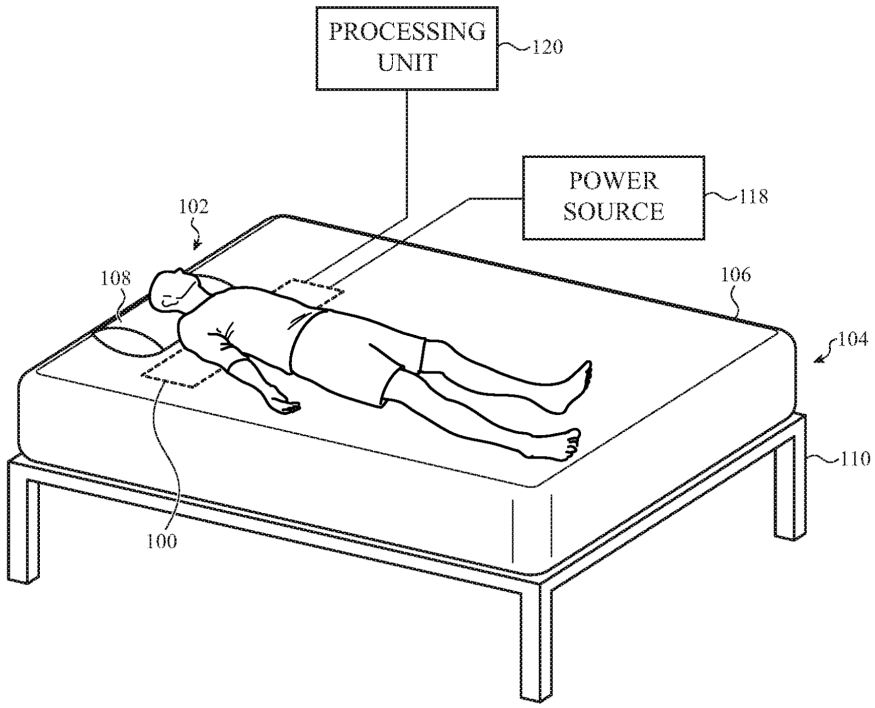


Figure 4.5: Patented Design for sleep tracking using a Piezoelectric sensor [35].

The wide variety of different sleep tracking devices shows the diversity of different techniques that can be used to track sleep [35].

4.1.4 Respiratory Trackers

There are also implementations of sleep tracking that use respiration patterns. These trackers can be noninvasive. One tracker uses motion sensors to measure the "anterior-posterior diameter of the chest wall during breathing" [36, 37]. They can often be used to identify abnormalities in breathing and can lead to a prediagnosis of sleep-related disorders. Yang et al. [38] uses Respiratory Variables for three-class sleep stage detection. Using a simple

threshold classifier, [38] is able to distinguish between REM, NREM, and awake sleep states, with an accuracy of 74%.

Overall, tracking respiration for sleep is an interesting domain that has great potential for exploration.

4.2 Sleep Detection Models

For this research we focus on classifying sleep using biopotential signals, such as electroencephalography (EEG) and electrooculography (EOG). We will outline two sleep detection models that use EOG, and EEG signals. We chose these models because the signals used in them are similar to the signals produced by the Flex-EOG device.

4.2.1 EOGNET

EOGNET implements sleep tracking using single-channel EOG signals [39]. Similar to the research in this thesis. Researchers use deep learning techniques such as CNNs to classify sleep stages.

As seen in Figure 4.6, the model uses a residual block along with a two-step training process. The first part of the model, with two scale CNNs is used to extract different features from the models, they minimize the cross-entropy loss between the true and predicted scores. By doing so, the weights of the first portion of the model i.e. the feature learning part, are optimized.

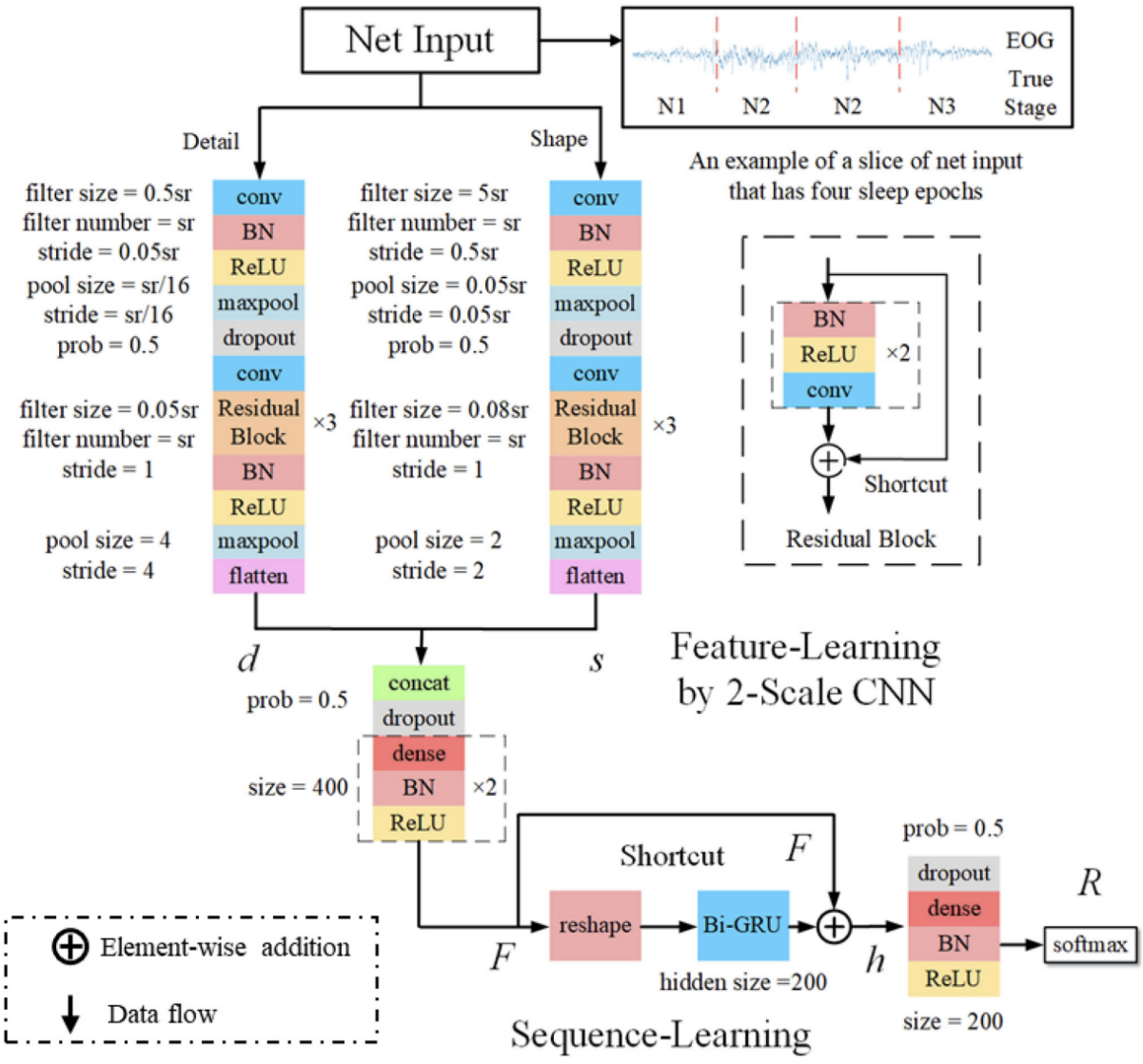


Figure 4.6: EOGNET: Proposed model used to classify sleep using single-channel EOG data.

The second part of the model takes the extracted features and trains them through a RNN based model, this portion takes advantage of the sequential nature of the EOG signals.

In the flow of the model, the first part is trained and optimized for feature extraction.

Then, the sequentially based model is trained with the optimized features from the first step.

Data Augmentation

Researchers translate existing data by 15 s and add Gaussian white noise to the remaining 15 s. This makes a total of 30 s epoch. This is done to address **class imbalance problem**, since the network weights will bias toward data with more labels. This happens in sleep stages since REM, N1 sleep only occur for a short period of time, so we have fewer recordings of them overall.

Results

Figure 4.7 shows the Confusion Matrix; overall it can be seen that for the five-class classification, N1 sleep suffers from poor results, and in the four-class classification deep sleep has the poorest results.

The model can attain a promising classification accuracy with 81.2% and 76.3% in dataset one and two for the five-class task and 85% and 82.1% in dataset one and two for the four-class task, respectively. **Note: The four class task refers to combining N1, N2 sleep Labels into light sleep, as seen in figure 4.7** For our research, our data set will consist of four-class accuracy. The model can accurately recognize minority classes since the F1 score and k are high. According to the results, single-channel EOG has equal performance to models trained with more common inputs, such as EEG. Therefore EOG signals, are capable of being used as primary sources of data input to train sleep stage detection models. [39].

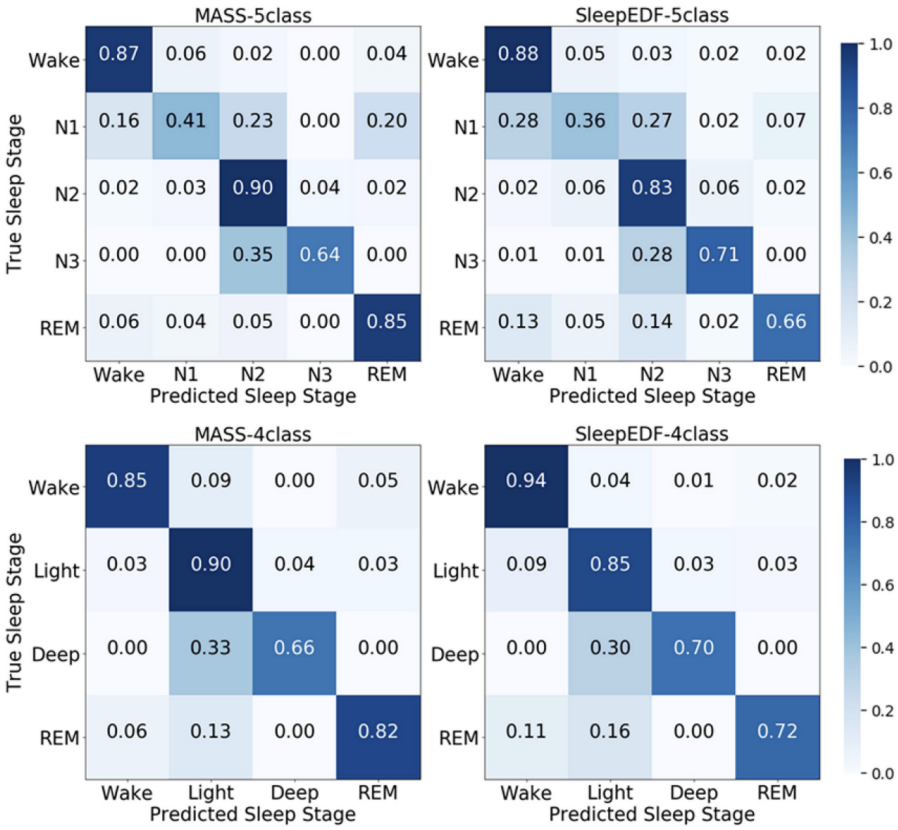


Figure 4.7: Confusion matrix of results; two datasets, and either five class or four class.

4.2.2 Orthogonal Convolutional Neural Networks for Automatic Sleep Stage Classification based on Single-Channel EEG

This model takes a single-channel electroencephalogram (EEG) from two different datasets and passes them through a Hilbert Huang Transform (HHT). They "use HHT to convert the 1D signal into a 2D time–frequency representation (TFR) that characterizes the time–frequency distribution of the instantaneous amplitude of the EEG signal" [40]. This representation is similar to the spectrogram approach technique used in this thesis.

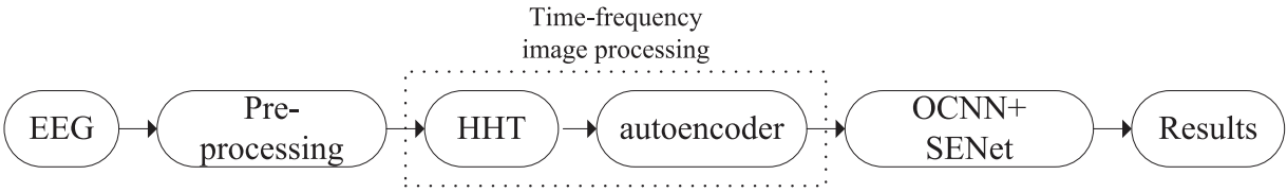


Figure 4.8: The pipeline of the sleep detection model.

As seen in Figure 4.8, the model uses an HHT along with an AE for dimensionality reduction [40]. As mentioned in Section 3.2.3, the autoencoders also play a role in feature extraction.

Figure 4.9 shows the orthogonal convolutional neural network (OCNN) which uses Orthogonal filters in a CNN, as well as the squeeze-and-excitation network (SENet). See [41], which covers the architecture behind the Squeeze-and-Excitation block, computational unit. A basic description of the benefits of SENet blocks from [41] is given below:

"In the earlier layers, the SE Block excites informative features, in a class agnostic manner." Low-level representations of features are strengthened as a result, since their important parts are "excited". Hence, they will stand out despite any surrounding noise. During later layers, SE blocks become class specific, each input will have a distinct response according to their class.

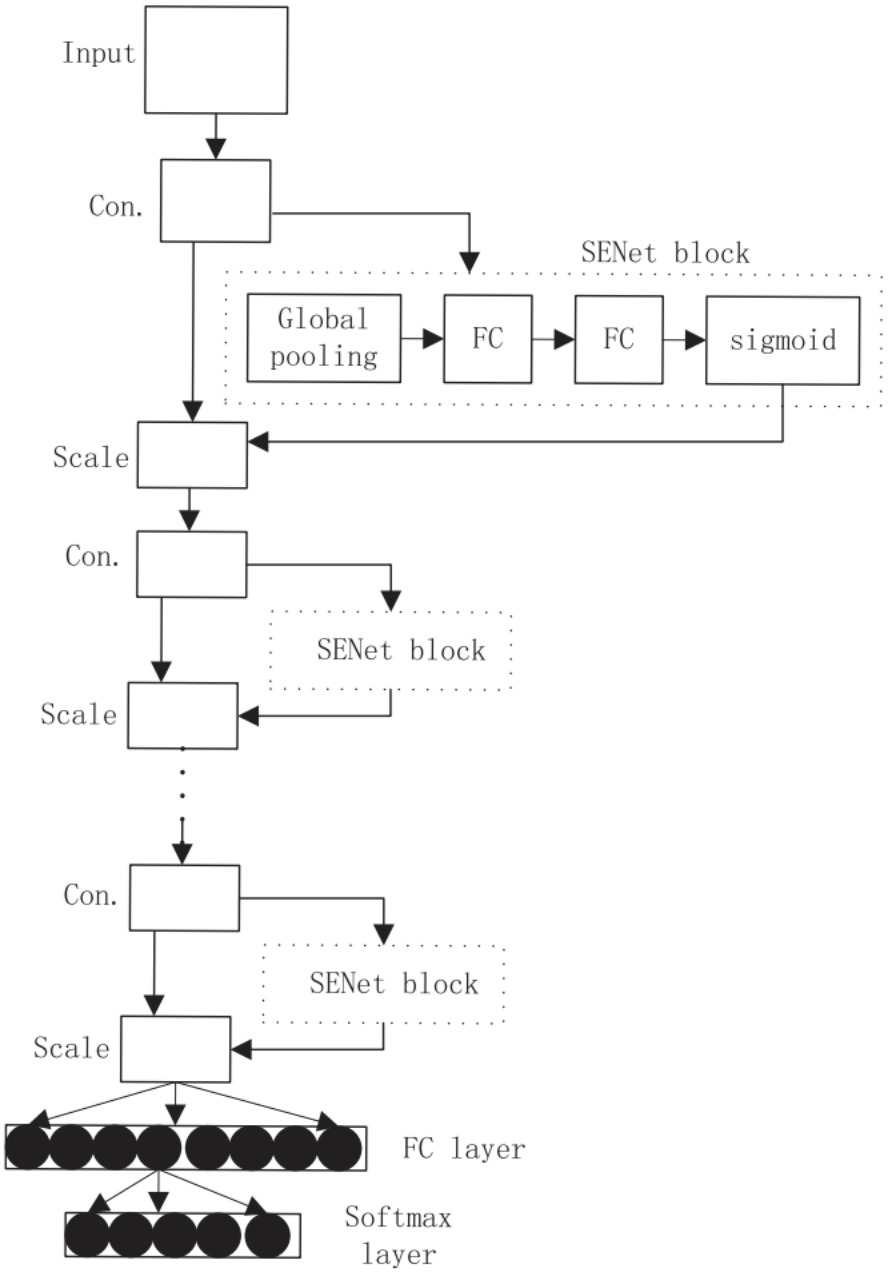


Figure 4.9: The OCNN + SENet model.

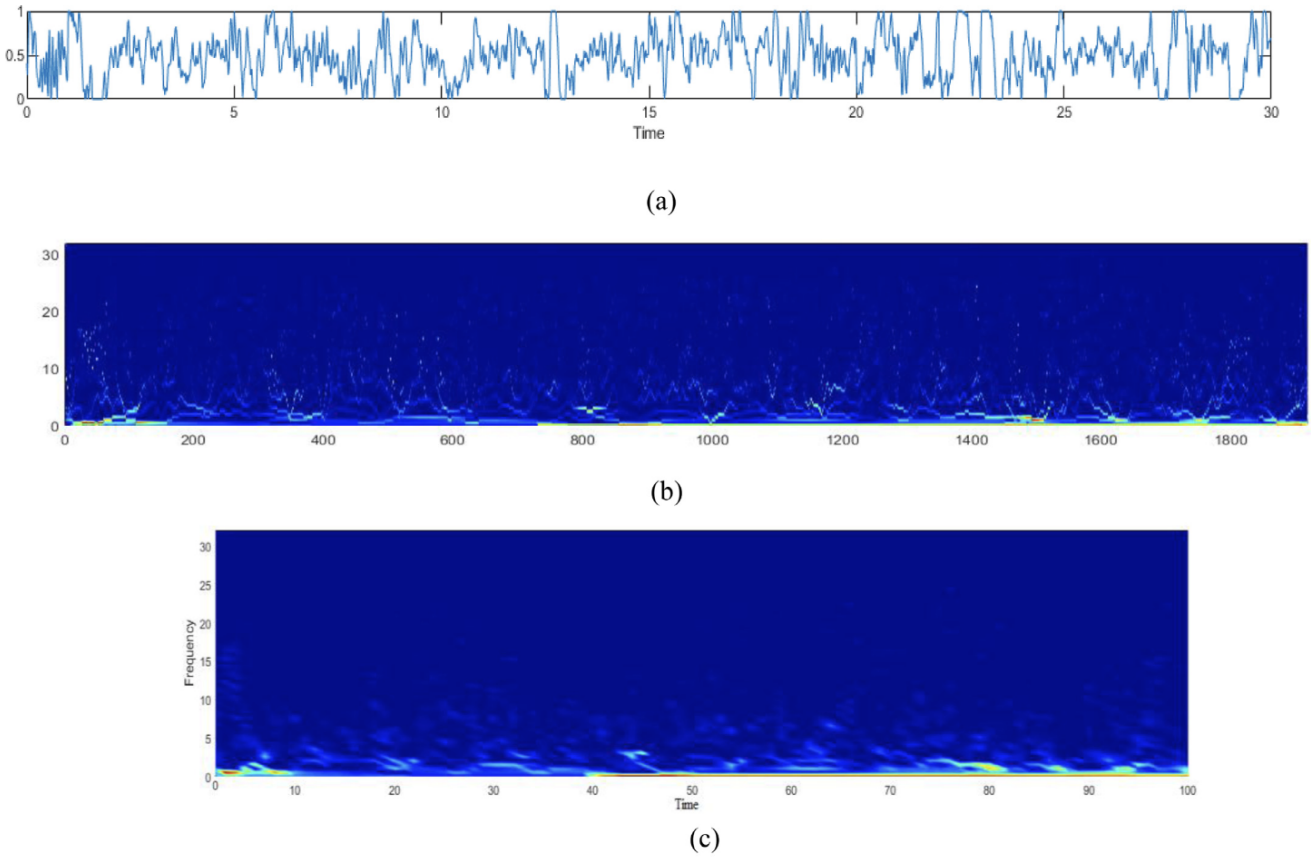


Figure 4.10: a) Time series: 30-s epoch EEG. (b) time–frequency image, (c) dimension reduction: output.

Methodology

The spectrogram, in Figure 4.10 b, is obtained from the conversion of the frequency domain of the time series EEG data. As shown in Figure 4.8, the structure of the pipeline is established. From the pipeline, we can see that the spectrogram is passed into a AE network for feature extraction. The autoencoder portion of the pipeline is trained first, before the spectrogram is inputted into the OCNN. Subsequently, the OCNN is trained with the "dimension reduced"

output from the AE, this is shown in Figure 4.10 c).

Results

Confusion matrix obtained from the two dataset (%).

Dataset	TAC	KP	F1-score for each class				
			W	S1	S2	SWS	REM
UCD	88.4	0.82	90.1	80.7	95.6	92.4	85.8
MIT-BIH	87.6	0.80	89	79.6	96.5	93.8	83.3

Figure 4.11: Confusion Matrix for OCNN sleep EEG.

Figure 4.11 shows the results of the confusion matrix, from the two datasets. Overall, decent accuracy is achieved, with accuracies greater than 80% for all sleep states (except S1). Given the novelty of this problem, current accuracies are expected. Especially in the medical field, where accuracies are in similar rangers. One example is Heart Arrhythmia detection; Stanford ML Group obtained an F1 score of 83.7 [42]. So, if we extrapolate to sleep detection, these scores are around SOTA.

However, it is noteworthy that the accuracy of the S1 score is considerably lower than in the remaining sleep states. The paper states that

Due to S1 being a transition state from wake to other sleep stages, it often exhibits patterns of other sleep states. Therefore, it can be difficult to classify. Initially, the S1 waveform resembles wake, and the latter portion resembles S2 sleep. Therefore, it is difficult for the model to identify S1 as a distinct sleep stage [40].

Although the above statement is true, the datasets also suffer from a class imbalance problem. That the authors missed. The problem is caused by the data set that contains disproportionate labels for each class. A class imbalance problem is caused in sleep detection because S1 is a transition state. Each overnight recording might contain a couple dozen epochs of S1 and hundreds of epochs of other classes. This will cause the model to gravitate towards the classification of abundant labels during training, and often leads to poor accuracy for rare labels. This problem is addressed in EOGNET [39]; it can be mitigated by using data augmentation 4.2.1.

Chapter 5

Experimental Setup

5.1 Experimental Setup

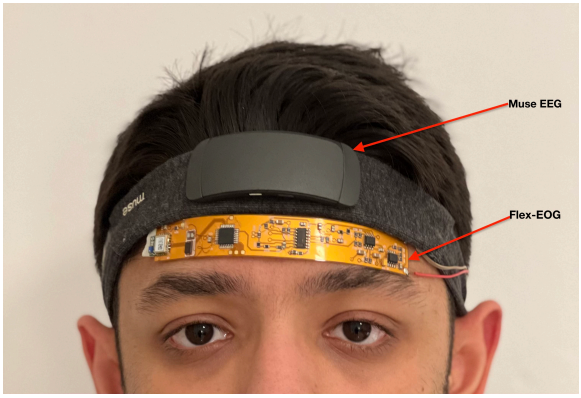


Figure 5.1: Placement of Muse and Flex-EOG bands on face.

We will outline the setup that we used to record sleep data from our Flex-EOG device. We take overnight EOG recordings. Participants wear the Flex-EOG device while sleeping.

As a ground truth, we use two separate sleep stage detection devices in order to track the participants' sleep stages.

A Muse electroencephalography (EEG)(www.choosemuse.com), and The Google Nest Hub (2nd Gen) [43]. Both Muse and Google have developed sleep scoring algorithms, which allow us to determine the sleep stages of the participants. As shown in Figure 5.1, two devices are worn at the same time when sleeping. The Muse EEG band is worn above the Flex-EOG band. The Flex-EOG band should be positioned right above the eyes to retrieve corneal biopotentials. The Flex-EOG device will be worn simultaneously to collect EOG data. The Google Nest Hub will be the third device that is set up on a bedside table next to the user.

5.1.1 Muse EEG Device

The Muse EEG device is a low-cost portable EEG device. It has been used for numerous research purposes in multiple medical contexts, including human visual attention, stroke diagnosis, and research on event-related brain potentials (ERP) [44–46].

The Muse iOS App has built-in sleep tracking with four different sleep stages: Awake, light sleep **(Equivalent to N1, N2)**, deep sleep **(N3)**, and REM. These sleep stages are measured using EEG signals from brain waves. To account for the different labels, we use Muse sleep labels. The Wisconsin Sleep Study labels are converted, using the guidelines given above, that is, N1, N2 becomes light sleep and N3 becomes deep sleep.

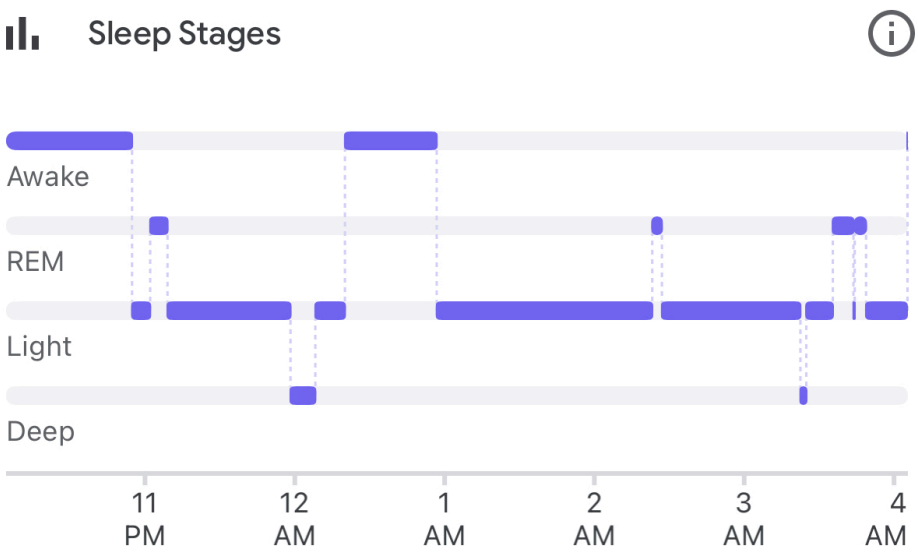


Figure 5.2: Sleep stages recordings from Google Nest Hub. It shows the different stages, and when the user entered each stage during their sleep.

5.1.2 Google Nest Hub

The Google Nest Hub uses a radar sensor to track your movements. This millimeter-wave sensor emits a radio wave and uses the reflected signal to figure out if someone has moved; their velocity and distance. With this data they are able to figure out the user's sleep states. The radar sensor data is trained on a machine learning model for sleep tracking. Google trained its Nest model on more than a million hours of sleep data taken from their clinical studies [34].

Like the Muse device, the Nest Hub creates sleep stage labels in the same format. Figure 5.2 shows the different stages of sleep of a participant taken overnight using a Nest Hub.

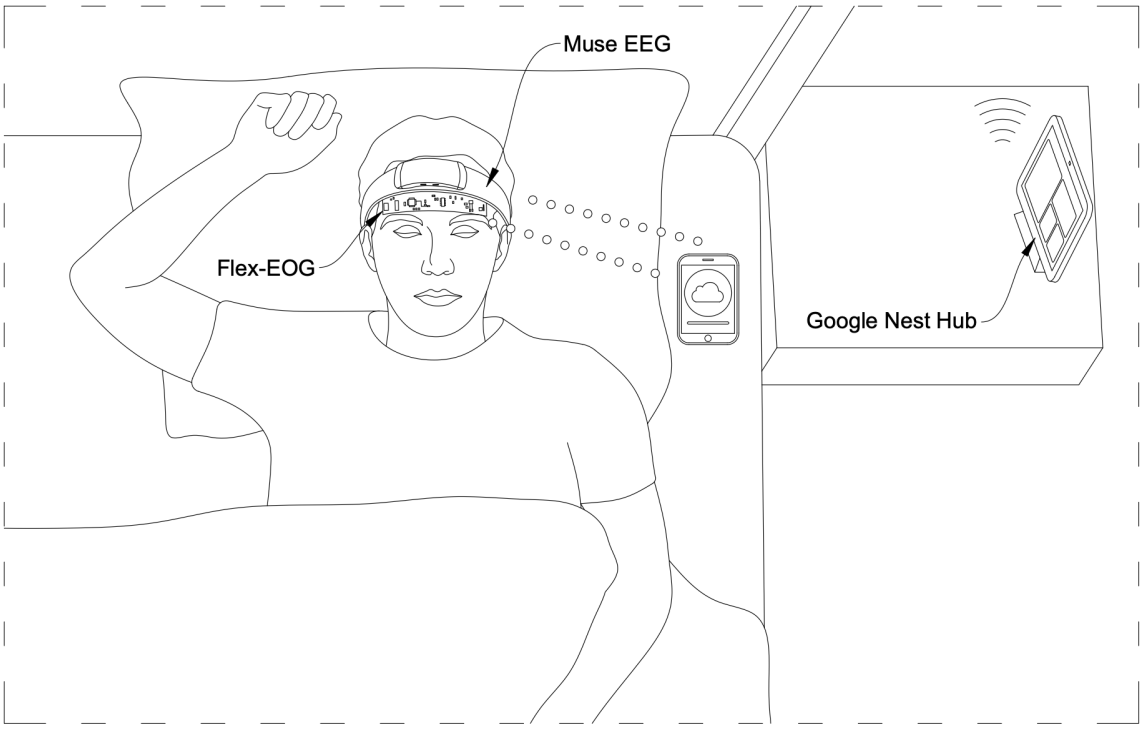


Figure 5.3: Sleep setup used to gather data.

Figure 5.3 shows the overall setup that is used to collect sleep data from the Flex-EOG device, as well as sleep stage labels from the MUSE and Google Nest Hub devices. The Google Nest Hub screen has to be facing the participant. It must be placed at the same level as the bed and within 1.5 meters. This is because the radar sensor is located at the front of the device. The iOS mobile device can be located anywhere near the participant, as long as it does not obstruct the Google Nest Hub. The person who manages the sleep study must ensure that all 4 devices are running properly. The MUSE EEG Band and Flex-EOG must be fully charged. The iOS device must be connected to a wall outlet to ensure that the charge does not deplete during the duration of the study.

5.2 Creating the Flex-EOG Dataset

We will describe the techniques used to create a dataset from recorded sleep data. The dataset will be used to build a machine learning model that can classify sleep states.

Algorithm 1 Extracting the Flex-EOG Data.

```
1: Extract Flex-EOG data from cloud as CSV
2: for Every timestamp in Flex-EOG data do
3:     Add concurrent sleep label from MUSE EEG
4:     Add concurrent sleep label from Google Nest Hub
5: end for
```

Algorithm 2 Creating the Flex-EOG Dataset.

```
1: for Each epoch of Extracted Flex-EOG data do
2:     Randomly choose a number N between 0, 1
3:     if N = 0 then
4:         Append Google Nest sleep Label
5:     else N = 1 then
6:         Append MUSE EEG sleep Label
7:     end if
8: end for
```

In order to create the Flex-EOG dataset, we must combine data from three separate sources. As shown in Figure 5.3 and discussed in the Experimental setup, the three sources are: Flex-EOG, MUSE EEG, and the Google Nest Hub.

This is detailed in Algorithm 1. We have to extract the Flex-EOG data from the cloud. Then, for each timestamp, we put the concurrent sleep stage that was detected by the Muse and Nest Hub device. Both Muse and Nest Hub have their own iOS apps that store sleep recordings. Therefore, we obtain Muse and Nest Hub sleep recordings from the iOS device.

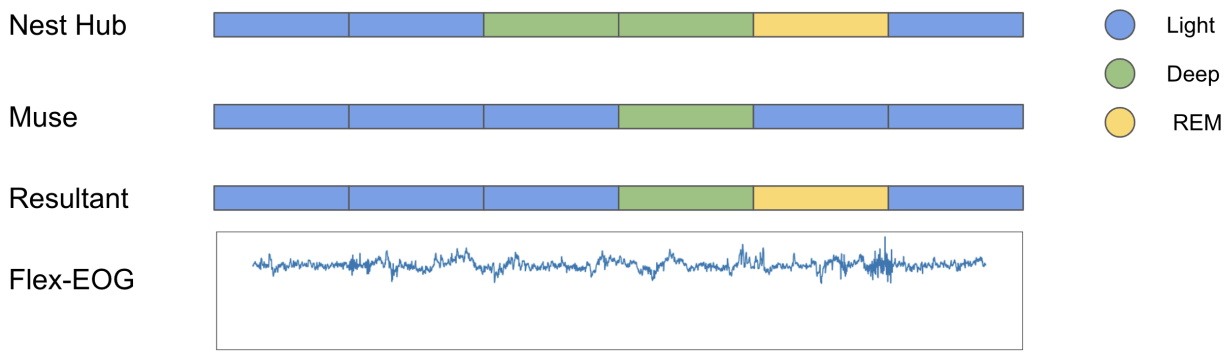


Figure 5.4: Technique used to create Flex-EOG dataset. We randomly select our sleep stage from our ground truths: The Muse and Nest Hub. We pair our selected label concurrently with our Flex-EOG data.

Once we have created a combined file with our three data sources, we further preprocess it to create our Flex-EOG dataset. As shown in Figure 5.4 and Algorithm 2, we obtain two separate ground-truth labels from Muse, Nest Hub. Then, for each epoch, we randomly choose our sleep stage label from one source to form our resultant sleep stage label. Using data from two different sources, our sleep stage labels are less prone to error. We then combine our resultant sleep stage label with our Flex-EOG data to create a dataset that will be used to update our model. Overall, the device was worn for 16 overnight recordings. For an average of 8 hours of sleep per day, this is equal to 128 hours. Data was collected from 5 different individuals, with between 2-4 nights of recordings from each person. The setup across different sessions was consistent with Figure 5.3. However, given that this was an experimental device, it would give us much less data recording. The Bluetooth chip would often have difficulty sending data. The device was also debugged and the battery was replaced. However, this did not make a difference in the function of the EOG device.

At the end of the day, we were able to collect around 35 hours of data, before preprocessing. It is a sufficient amount of data to train machine learning models.

5.3 Wisconsin Sleep Study Dataset

We will discuss the Wisconsin Sleep Cohort (WSC) and its role in the development of the sleep detection model. The Wisconsin Sleep Cohort (WSC) is a study of sleep apnea. It uses in-laboratory sleep studies conducted overnight with a sample of 1,500 Wisconsin State employees. The WSC is used to train the initial deep learning model and populate the data. Since we are in the early stages of the Flex-EOG project, we have not conducted clinical trials. Since smaller datasets are prone to overfitting, we populate our model with the WSC dataset. Also, we are able to train a higher quality model if we train with more high quality data.

The Wisconsin Dataset takes Polysomnography (PSG) recordings of participants while they are asleep overnight.

PSG is a combination of "EEG scalp electrodes, EOG, EMG of the chin and legs, ECG, snore microphone, airflow from Dymedix nasal-oral thermistor, Pro-Tech nasal pressure transducer, breathing effort from Pro-Tech zRIP inductance plethysmography summation systems, and oxygen hemoglobin from the Ohmeda 3900 oximeter using a 3-second averaging rate."

The WSC study currently contains 2570 overnight recordings that currently use the Comet Lab-Based system, Grass Technologies, to record the PSG. Then, each recording is sleep scored by a technician at an interval of 30 seconds [47, 48].

From the WSC dataset, we extract:

1. EOG values
    - E1 (E_{left}), E2 (E_{right}) which is obtained from the left and right eyes.
2. Sleep stage labels
3. Epoch number
4. Seconds elapsed

We will go into more detail about the Wisconsin Sleep Study dataset in **Chapter 7.2**, where we discuss **Model Design and Methodology**.

We propose a sleep stage classification algorithm to work on the Flex-EOG device. It is a deep convolutional neural network that has been trained on data extracted from the University of Wisconsin Sleep Study (WSC). Then the model is updated with the dataset that we created from our Flex-EOG device via transfer learning.

Chapter 6

Mobile Integration and Experimentation

This section will detail the development of the iOS application to interface with the Flex-EOG device and the cloud.

The iOS application packages the EOG data into a dataset that contains multiple variables necessary to process the data. All the added variables correspond to the same measurements found in the sleep datasets. We extract three variables: EOG value, seconds elapsed, and epoch number.

Once the iOS App processes and packages the Flex-EOG data into a datapoint, it sends the data to the cloud. The cloud will load the data into a csv formatted dataset.

6.0.1 Mobile App

The mobile app was developed for an iPhone using SWIFT and XCode. Both Swift and XCode are development tools for iOS devices. The mobile application architecture consists of two components: Bluetooth, and cloud Firebase.

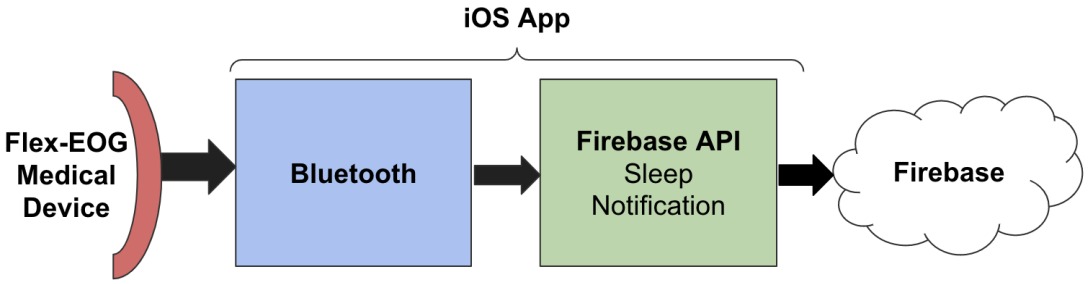


Figure 6.1: Structure of Sleep App.

Figure 6.1 shows the structure of the iOS application. As seen in 6.1, Flex-EOG will communicate with the Bluetooth module on the iOS device and send the raw EOG data. The app will then package the data in a **Notification Object**, which we will now refer to as a **Sleep Notification**. The Sleep Notification will then be sent to the cloud using **Firebase API** for SWIFT & iOS.

Core Bluetooth

Apple Developer comes with a built-in **Core Bluetooth** library that takes care of all Bluetooth functionality. **Core Bluetooth** is integrated into the sleep application. It detects and connects to the sleep device by using its universally unique identifier (UUID).

Once connected, the device will begin to feed the EOG data to the app. The application has to process and receive the data and send it forward to the cloud.

Firebase

The Firebase interface will receive the Bluetooth data collected that will be sent to the Firebase Cloud console. Firebase uses a data collection format known as NOSQL, a document-oriented database in which data is structured as **documents** and organized as **collections**. Documents can contain **subcollections** and nested objects. Firebase has developed its own API to work with SWIFT and iOS. In the application, the data sent from the Flex-EOG device via Bluetooth is parsed and packaged in an object; as stated above, we will refer to the object as **Sleep Notification**.

The Sleep Notification contains:

1. EOG Data:
    - This is the data sent from the EOG device.
2. Timestamp:
    - The time at which the object is created and received by Firebase, and it is a default feature of Firebase. This value is not equal to the time at which the value was created by the Flex-EOG, and due to Internet speed and the difficulty in sending data incoming at 100Hz, there is a lag between the Timestamp and

created time.

3. Epoch:

    - This records the epoch value of the data, since there are multiple writes per second, and the timestamp is not sufficient in order to organize the data, and we require this since Firebase stores data randomly and not in the order in which it was sent.

4. Seconds Elapsed:

    - To work with medical data, we need to determine which second has elapsed since the beginning of the experiment, and we use the second measurement to preprocess the Flex-EOG data and remove any readings after 100Hz.

5. Time Interval Since 1970:

    - This gives us the number of seconds and milliseconds elapsed since January 1, 1970. We need this since the Timestamp variable has a lag, so this is used to check the time the datapoint was created by the Flex-EOG, and **seconds elapsed** can not be used since we receive 100-200 data points per second; it does not give us an accurate time of when each point was created.

To ensure consistency across datasets, we also cropped our Flex-EOG data to 100Hz. The Wisconsin Sleep data set that we used to initially train our model has a frequency of 100 Hz for the EOG data. So, we parse our Flex-EOG data and remove values beyond the initial 100 measurements for each second. This is a necessary step, since the Flex-EOG device has a frequency that varies from 115-130 Hz.

The iOS app packages all the data and sends it to the cloud. In the Sleep Apnea iOS app, the object is created when receiving the Bluetooth data, and then the function **addDocument** is called. This Firebase function will then write the document to the specific collection in the cloud.

Since there will be multiple users, the database hierarchy is organized as shown in Figure 6.2: As shown in 6.2, the data is structured into documents that contain the names of the

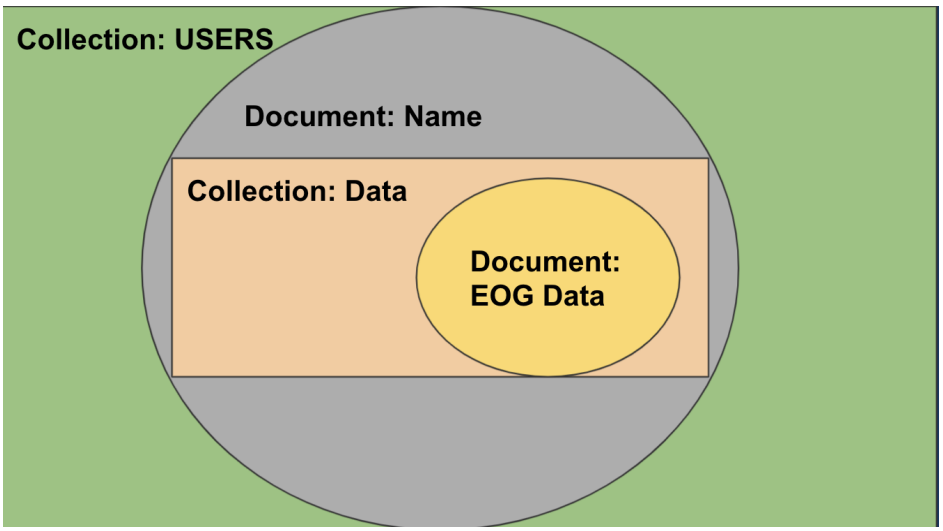


Figure 6.2: NOSQL Hierarchy of Sleep Apnea App.

users, and the documents of each user have their own collection **collections:Data**. The

collections:Data contains documents that write Sleep Notifications to Firebase. The main collection that stores all the data is **Collection: USERS**, all other subcollections and documents are inherited from USERS.

The iOS app is necessary in this scenario. Not only do we need to collect data, but we also need to be able to record variables such as time and epoch number. With time and epoch, we can determine the current sleep stage of the user. This is done by checking the Muse and Nest devices for that time. These two measurements are standard in medical datasets and form the basis for the preprocessing and labeling of sleep stages. We are able to create 30-second intervals of data.

Chapter 7

Model Design and Methodology

7.1 Introduction and Overview

There are multiple ML techniques that can be utilized for data classification. In this case, we have a time series dataset that can classify the different stages of sleep. These techniques include the use of convolutional neural networks, and recurrent neural networks. As stated in the previous section, CNNs, which are known for their image classification utility, can be used to classify input data from the device that is not converted into an image. For such tasks, 1-Dimensional (1-D) Convolutions are used in the model. RNNs are useful due to their time-dependent nature; they are able to extract domain knowledge when running over a time-based sequence. When combining 1D CNNs and RNNs together, the model is able to build a strong classifier that is able to traverse the search space and learn time-

dependent patterns. Data preprocessing is required to further mitigate mitigate the effects of noisy data and to highlight patterns. The preprocessing types used are the wavelet and Welch transforms. Raw input data is also fed into the model. The Welch transform can reduce noisy data by taking the power spectral density of the data at different frequencies; it highlights important areas in data analysis that are used for classification.

Lastly, a final classification technique involves the use of images. There are two cases: the preprocessed and raw data are converted into an image based graph, and raw data is converted into a spectrogram.

These images are then fed into classification models such as LeNet. The spectrogram-based model takes in the raw input data, and the spectrogram, if analyzed correctly, can be very good at differentiating different types of signals. With techniques mentioned in the previous sections; by using speech detection inspired models, we are able to extract key information from the spectrogram that can improve the performance of the model. The reason for choosing multiple models was to evaluate and contrast the performance of different models, in order to select a suitable model for the sleep detection device. It is also in our best interest to explore which models can be suitably applied to a wide variety of applications, especially speech recognition-based learning techniques applied to a spectrogram, since this is a novel field with room for exploration.

The models are evaluated based on the following criteria, in order of priority: F-1 score, accuracy and novelty of the architecture.

7.2 Dataset Used

The choice of dataset was determined according to the specification of the wearable device. The requirement was to obtain data that measured the electrooculography (EOG) of the patient during a sleep study. Throughout the duration of the sleep study, the data must be labeled according to the different states of sleep. Data was obtained from the National Research Resource Resource (NSRR) by filing a request for access and undergoing data ethics training. The chosen study was *The Wisconsin Sleep Cohort Study*, "an ongoing longitudinal study of the causes, consequences, and natural history of sleep disorders, particularly sleep apnea". It uses overnight sleep studies (patient studies at the University of Wisconsin - Madison) conducted with a baseline sample of 1,500 participants, assessed at 4-year intervals [47, 48]. The WSC study measurements comprises of three main portions:

1. Patient Polysomnography (PSG):
    - PSG records bodily functions such as brain waves (EEG), oxygen level in your blood (SPO_2), heart rate (ECG), as well as eye movements (EOG) and leg movements (EMG) during the study [49].
2. Sleep stages:
    - These are manually scored by a technician, and sleep Scoring follows the rules of Rechtschaffen and Kales [3]
3. Apnea and Hypopnea Events:

- These are manually scored by a technician, and guidelines for scoring different events are listed in the WSC manual.

The WSC study was chosen over other NSRR studies because its main objective was to study sleep and apnea events; therefore, it provided a comprehensive apnea score, unlike some studies that did not include apnea scores. The apnea scoring can be used for future work and further investigation. The study was also longitudinal and contained a large amount (30 GB) of data, covering more than 2,570 overnight recordings, which could be used to train the models. Lastly, this study has been referenced in more than 100 academic research papers, allowing access to a pool of relevant analyses that can influence the direction of research.

7.3 Preprocessing

We will describe the different types of preprocessing that we used on our data. We also discuss the key features of each preprocessing used. We feed our models with four different types of input: raw data, Welch, wavelet, and spectrogram.

7.3.1 Raw Data

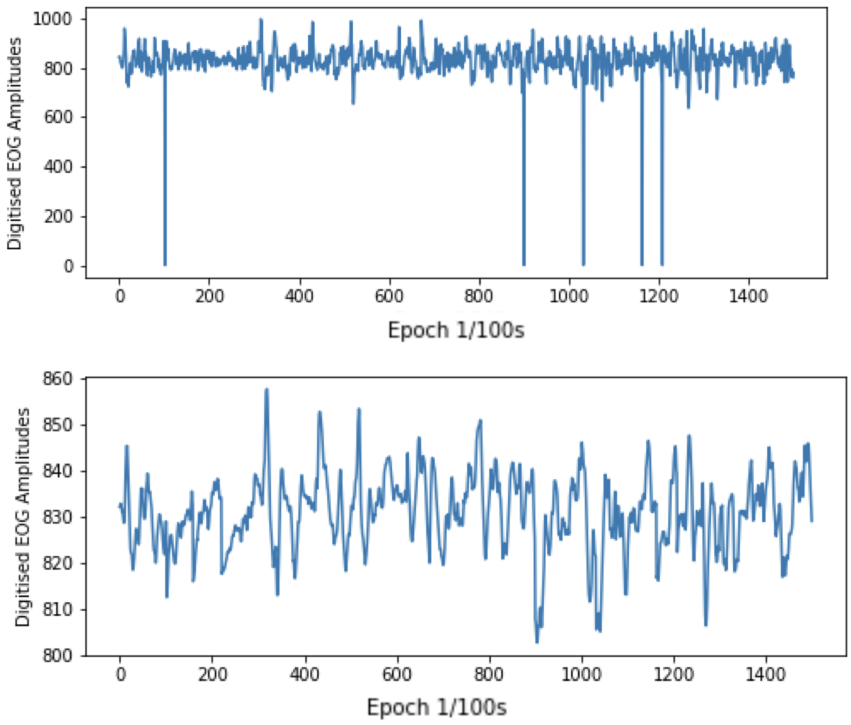


Figure 7.1: Before and after computing the weighted moving average of Flex-EOG data. The random spikes from the data are removed. The range of the data is shortened significantly.

Raw data is unprocessed EOG data. We feed this data directly into our model. Each epoch has 3000 datapoints, which corresponds to 30 seconds of recordings. There is no data compression for the raw data. For the dataset extracted from the Flex-EOG device, there is some noise that needs to be removed before feeding our data into the model. The device itself will output some random spikes. We remove this using a technique known as the exponentially weighted moving average (EWMA) [50]. Our future data is determined by

prior datapoints; this allows the data to stay within a range and lessens the size of the spiked data. As the moving average advances from the previous data points, it has an exponentially lesser effect on the current value.

7.3.2 Welch

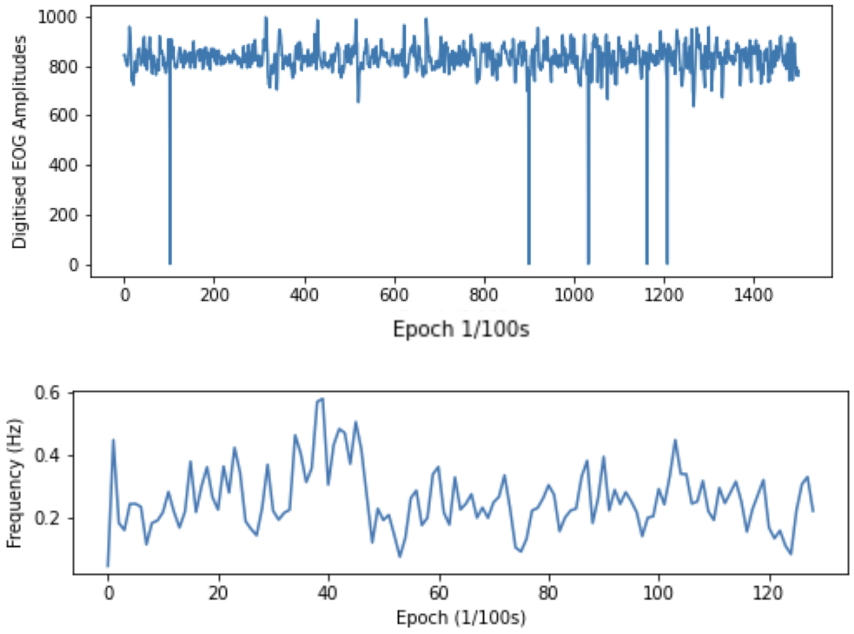


Figure 7.2: Before and after computing the Welch transform on the Flex-EOG data. Like the weighted moving average, the Welch transform also has removed random spikes. It is much more continuous.

The Welch transform is a really powerful technique that estimates the power spectral density of the data. It is calculated by taking the periodogram of the data over successive blocks of time and averaging it [51]. For each epoch, the Welch transform tells us which

frequencies contribute the most to our EOG signal. Welch not only extracts succinct information and key features from our data, but it also compresses the size of our dataset from 3000 datapoints per epoch to 129 datapoints per epoch:

$$z = \frac{S_u}{S_c} = \frac{3000}{129} = 23.25 \tag{7.1}$$

$$m = 1 - \frac{S_c}{S_u} = 1 - \frac{129}{3000} = 99.23\% \tag{7.2}$$

As shown in Equation 7.1, we have a data compression ratio z, of 23.25. This equals a 99.23% of space savings m [52].

In conclusion, we used the Welch transform to reduce noise and extract key patterns from the data. We also use the Welch transform for data compression to make our data lighter.

7.3.3 Wavelet

The wavelet transform can be thought of as a compromise between the frequency and time domains. We compute the wavelet transform by convolving a wavelet with the signal. It contains frequency information that is localized in the time domain. The wavelet used for Flex-EOG data is a Gaussian derivative wavelet of the second order. We use a continuous wavelet transform as opposed to a discrete transform. We noticed through trial and error that the continuous wavelet transform is better at noise reduction and extracting key features.

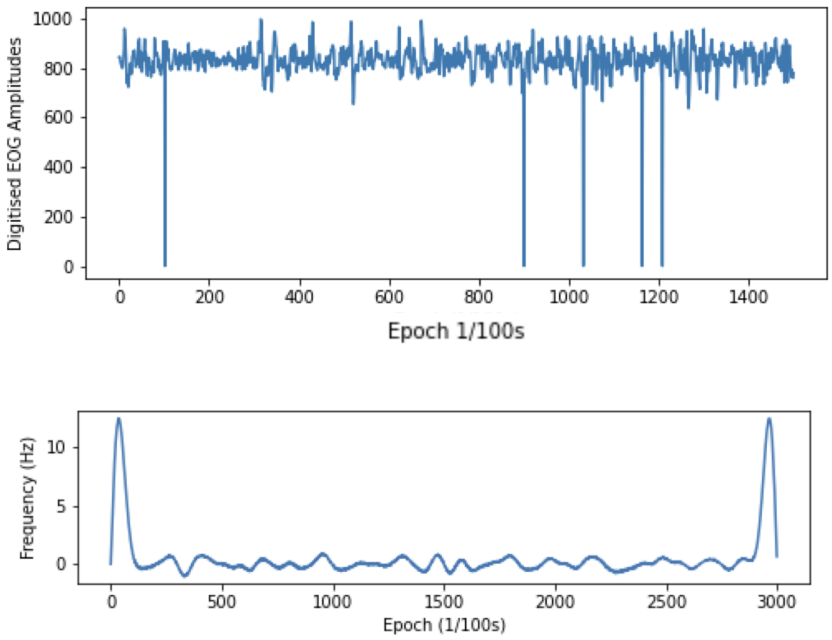


Figure 7.3: Before and after computing the wavelet transform on the Flex-EOG data. The wavelet transform changes our data into frequency vs time. We see that most noise is removed from our data, as the frequency transform only highlights key features.

This leads our models to train stronger classifiers.

One thing to note is that the discrete transform significantly reduces the size of the dataset by half, giving it a data compression ratio of two; however, the Flex-EOG dataset has too many spikes of noise. The discrete wavelet is not able to extract meaningful information as a result.

Therefore, we use the continuous wavelet transform, which has a compression ratio of one. That is, there is no compression.

7.3.4 Spectrogram

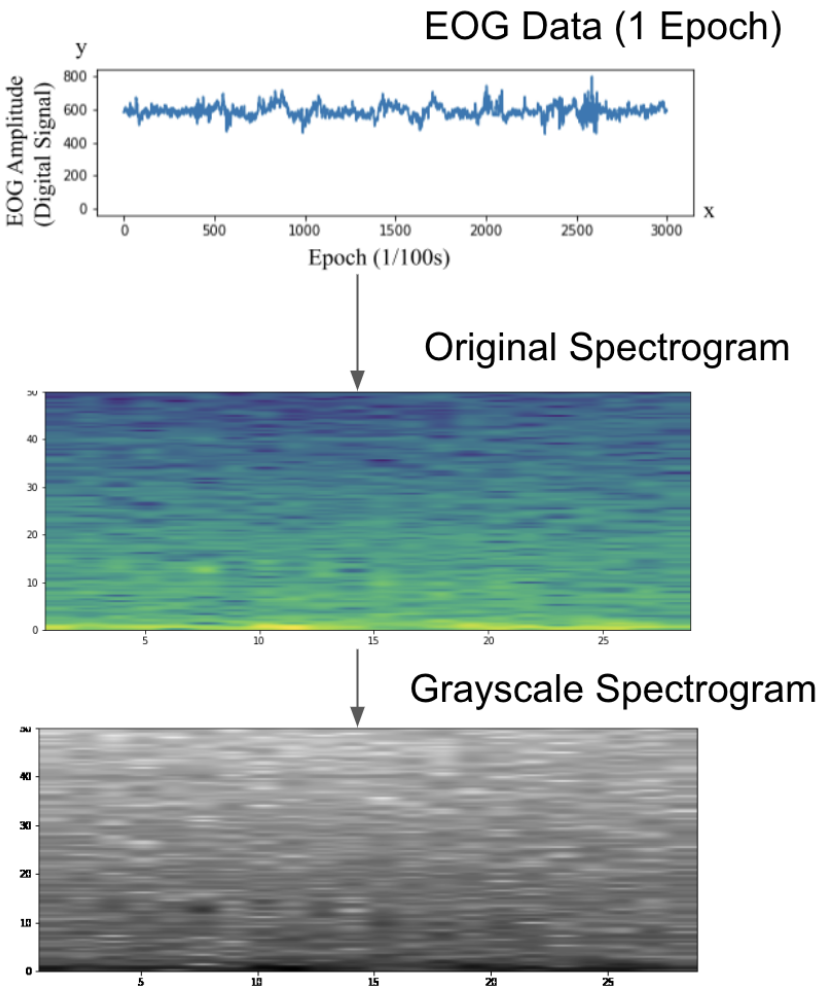


Figure 7.4: Steps taken to create spectrogram from EOG data. We convert each epoch of our original data into a spectrogram, which is a Graph of our Frequency signal over time. Then we convert our spectrogram to grayscale, since we don't require color information to detect sleep stages. This also compresses our model.

A spectrogram is a graph that visually shows the frequencies that a signal exhibits over time. It is used in many applications, including speech detection and medical AI. Even Google Nest Hub uses a spectrogram for noise detection during sleep; this includes snoring and coughing.

We create a ML model that trains on a spectrogram dataset. The dataset is made from EOG data from the sleep datasets. The steps we take to create the dataset are shown in Figure 7.4. Each 30 second epoch is converted to a spectrogram. We also convert the spectrogram dataset to grayscale. This is done because color does not contribute any additional information to the spectrogram. The grayscale image is a single channel, as opposed to the 3 channels required for color spectrograms. This allows us to theoretically compress our data by a factor of at least 3. To see such compression, there must be a data storage format that can take advantage of the grayscale spectrogram's smaller size.

7.4 Data Augmentation

We will discuss why it is important to utilize data augmentation and outline the different augmentations used on our data. Data augmentation is a necessary step in the classification of sleep detection. This is required to ensure that the model does not suffer from a class imbalance problem. A class imbalance problem occurs when the dataset has much more values for a particular class. Sleep detection always suffers from the Class Imbalance problem because most of sleep is light sleep. REM sleep and deep sleep are rarer during a human

sleep cycle, so naturally we have fewer of those data recorded in a sleep study. So, if we train the model, it is likely to overfit. That is, it will be adapted to detect light sleep, but it will not be able to determine REM, deep sleep. Our model will suffer from *"The Paradox of Accuracy"*, which means that while the classification accuracy of our model may be higher, it is likely to very accurately predict our majority class correctly. This causes our accuracy to shoot up, while minority classes like REM and deep sleep are poorly classified.

Another benefit of data augmentation is that it provides more data for our model to train with.

For time-series data, we used three types of data augmentation: Fourier transform phase shift, additive noise, and Synthetic Minority Over-sampling Technique (SMOTE). All techniques are proven techniques in the field of EEG and have been applied to similar tasks.

7.4.1 Fourier Transform

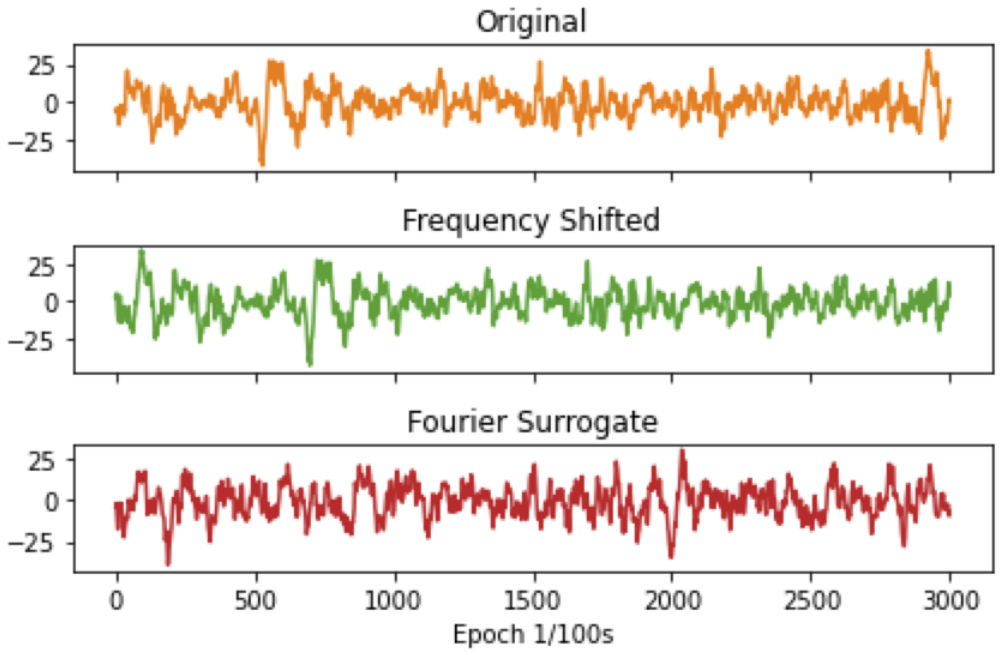


Figure 7.5: A comparison between the original, frequency shifted, and the surrogate data.

This technique involves a multistep process: Take the Fourier transform (FFT) of the signal, shift the phase of the FFT signal randomly from $[0, 2\pi]$. The frequency-shifted data is shown as the Green Graph in 7.5. Then, take the inverse Fourier transform of the shifted signal, input the phase-shifted inverse signal into a Fourier surrogate function, and our output will be the Fourier surrogate function; this is shown as the red graph in Figure 7.5.

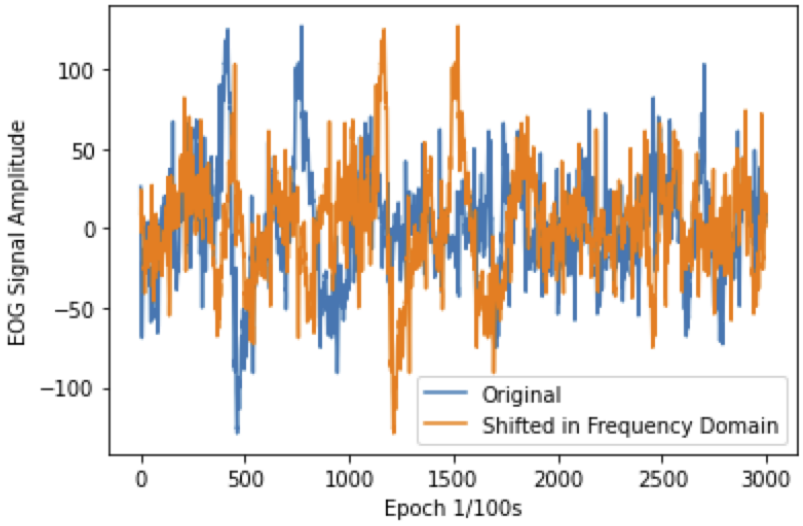


Figure 7.6: An example of a Random Fourier Transform shift on the WSC Dataset.

$$\tilde{x}(\omega) = \int x(t) e^{i\omega t}\, dt \quad \Longleftrightarrow \quad x(t) = \int \tilde{x}(\omega) e^{-i\omega t}\, \frac{d\omega}{2\pi} \tag{7.3}$$

Equation 7.3, shows the equation for a continuous Fourier transform. When we phase shift by θ, we shift our time coordinates from t to $t - \frac{\theta}{\omega}$. By shifting the phase in the Fourier transform and then taking the inverse Fourier transform, we will return a signal that is the same as the original data, but is shifted in the time domain. An example of this applied to the EOG data is shown in Figure 7.5.

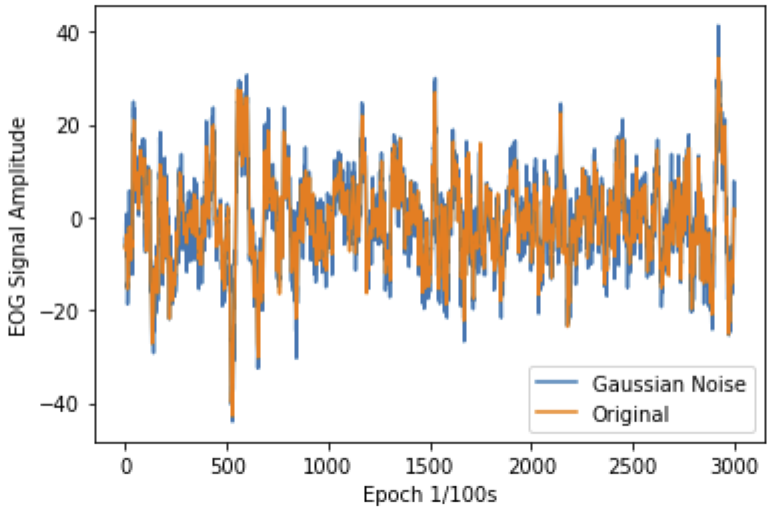


Figure 7.7: Graph showing a Gaussian Noise added signal superimposed on the Original Signal.

7.4.2 Additive Noise

Adding noise to the data is a simple, yet effective way to augment the data. The noise chosen to add to the Flex-EOG dataset is **GWN** (Gaussian white noise), with a mean set to zero. By setting the mean to zero, we ensure that over the length of our signal the Gaussian noise cancels itself out. It has an overall sum of zero. This ensures that the integrity of our data is maintained and that it is a positive addition to our classification model. The signal-to-noise ratio (SNR) is set between 8 and 12 decibels. This SNR is selected by reading literature that uses the same Gaussian noise in EEG data. Figure 7.7 shows a signal with added Gaussian noise. It is apparent from the Figure that the Gaussian noise just alters the amplitude of the signal at each particular epoch. That is why we can see that the blue signal of the added

GWN just barely exceeds the original signal in orange at some epochs.

7.4.3 SMOTE

SMOTE uses a k nearest neighbors approach to draw new datapoints between two randomly selected neighbors. SMOTE creates a new datapoint that is in-between both neighboring points of the same class. Thus, it creates a data point that shares the characteristics of its neighbors [53]. It is easy to apply and offers a quick solution to balance datasets.

7.4.4 Spectrogram Augmentation

For our Spectrogram Data, we employ a technique known as SpecAugment. Since a spectrogram is different from a normal image, regular image augmentation techniques do not apply. Regular techniques consist of blurring, rotating, and inverting. Due to the nature of a Spectrogram, which stores Frequency data, we must come up with techniques that do not modify or alter the scale and calibration of our data. SpecAugment takes a spectrogram and masks certain frequencies or times. This technique is known as frequency/time masking. SpecAugment was developed by Google Brain and is known to have a strong performance when it comes to increasing classifier accuracies. It is better than other augmentation techniques for spectrograms.

It is done by adding an occlusion/cropping of our spectrogram at certain bands. Figure 7.8 shows frequency and time masking. SpecAugment can be applied directly to the image,

without having to modify the original time-series data. It is done by zeroing the pixel values for the frequency/time band. Therefore, it is very cost-effective [54].

Figure 7.8 shows the SpecAugment applied to the Flex-EOG data. The time/frequency masked portion is shown in red, in reality it is a black line. However, for demonstration purposes, the color has been changed to red for clarity. We created both a frequency and time masked spectrogram. Also, for our dataset, we create data spectrograms that have both frequency and time masking on the same image. By doing this, we increase the different combinations of images that we can add to our dataset. Theoretically, we can have an infinite set of combinations of frequency- and time-masked spectrograms. This allows us to populate our training set and strengthen our model.

7.4.5 Applying Data Augmentation to our Data

Data augmentation is utilized in the pipeline of our model classification. We apply our augmentation only to our training dataset and not on our test set. This is done to ensure that our classifier can function on raw data, and augmented data only help the classifier perform that task. For our classes, we augment the data from our classes depending on how sparse the data is for each particular class. We try to create a balanced dataset with equal classes.

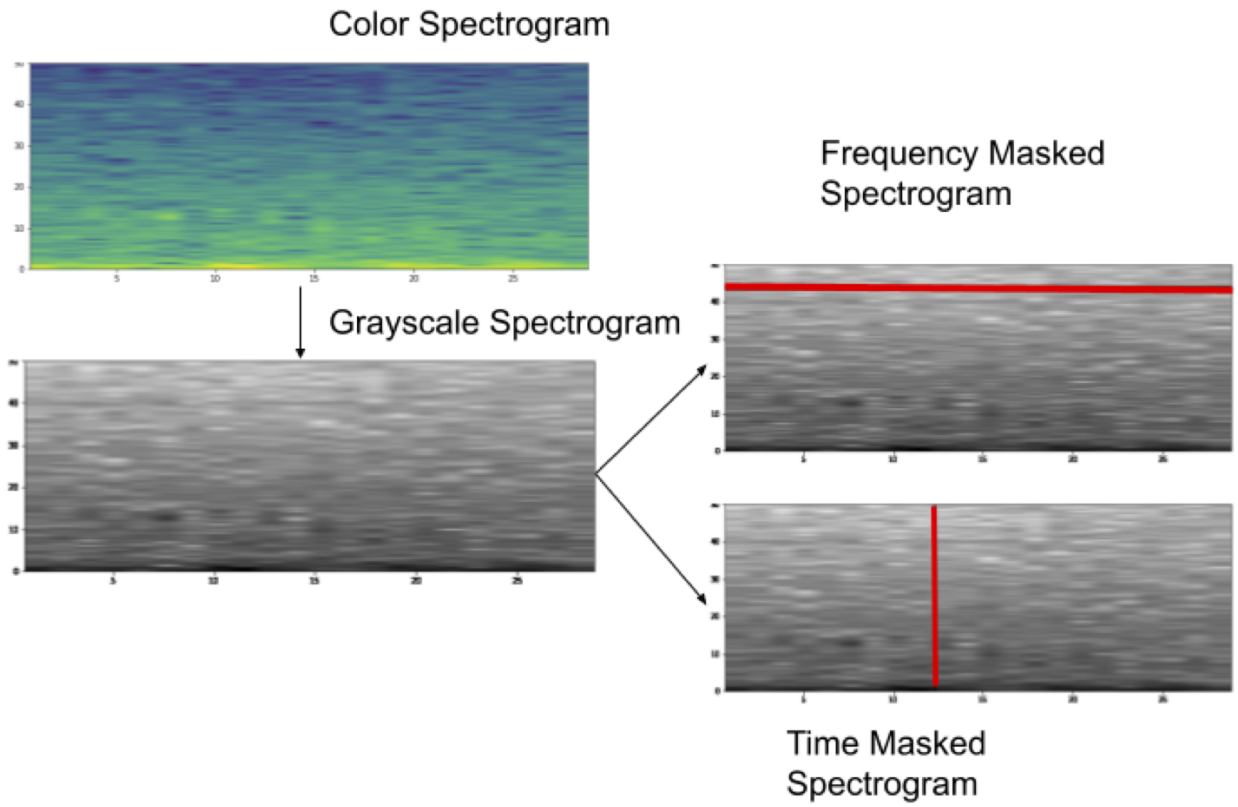


Figure 7.8: Applying SpecAugment to our sleep data. We take one epoch of a grayscale spectrogram and apply lines to them. The lines (shown in red) are meant to remove information from the spectrogram. The vertical axis is time ,asked, and the horizontal axis is frequency masked. By doing so we are feeding our model the spectrogram in a way that is not identical to its parent, but contains enough data and variation for augmenting our training set.

7.5 Architecture Choice

We will use the previously discussed literature on neural networks to help determine the design of our machine learning model. The choice of architecture is based on optimizing the accuracy of the model. Given the challenges, the input data presents. Being single-

lead EOG data means that it is sparse and prone to noise. The Integrated Microsystems lab communicated that preprocessing, such as low-pass filters, was necessary to extract meaningful data. The fact that the data of different sleep stages can be quite similar means that the classifier is more prone to making errors such as false positives and negatives. Lastly, since the classifier must integrate and function with a mobile device, the model must be lightweight and have CoreML functionality. However, it should be noted that not all the models developed will be geared for a mobile setting. Rather, the main objective of this paper will be to develop a successful sleep classification model and then select the best-fit candidate for the mobile device. This is because **the research** wants to prioritize the exploration of novel architectures with high accuracy. Afterward, model compression techniques can be used to reduce the size of the model. The model compression techniques can reduce the size of deep neural networks into a mobile friendly size and will be discussed later in this section. Given the uncertainty of the classifiers and the constraints set by the device, multiple different models were chosen to explore and find suitable candidates. The selected models were deep CNN-RNN, image detection on spectrogram. The architectures take into account recent advances in machine learning research and implement new concepts such as dropout, skipped connections, speech processing, transfer learning, etc. These modifications allow basic models to achieve high accuracy.

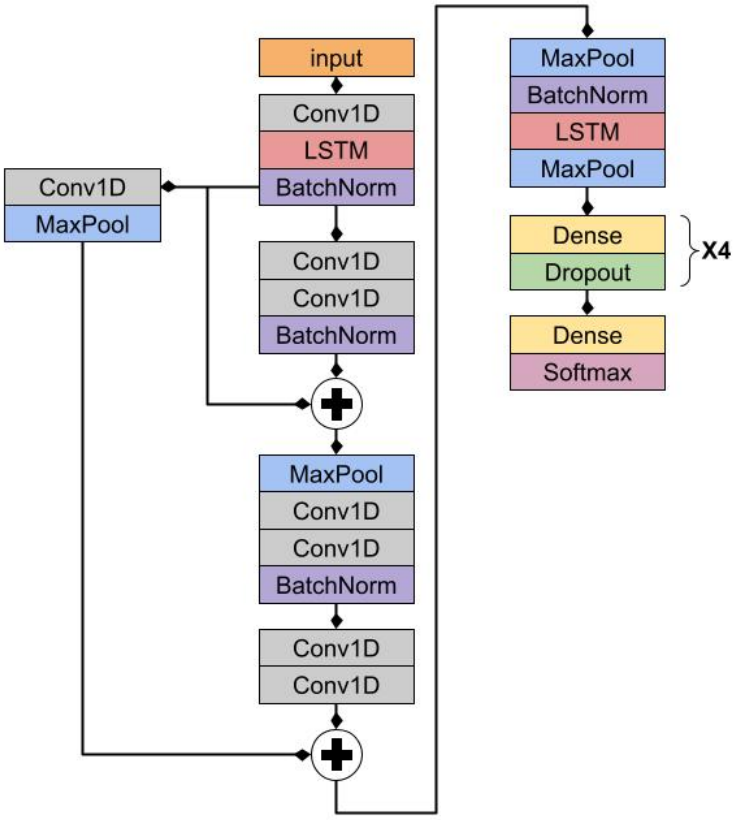


Figure 7.9: Deep CNN-RNN Network Structure.

7.5.1 Deep CNN - RNN

The Deep CNN-RNN architecture was selected because similar models have been applied for similar applications here [42]. This model involves the use of CNNs that form "very deep" layers, that is, greater than 16 layers total [55]. The deep CNN allows the model to traverse the search space to a greater extent, and hence perform better at classification. Skipped connections allow the model to find patterns at a macro and micro scale; that is,

the model will learn patterns in the forest and the trees. Lastly, the RNN layers will add a time-dependent component to the model, this will enable the model to learn patterns over time. As we add more data to the model via the iOS App, RNNs will help the model continue to improve, as it will have more sequential data to train on. The structure of the model is shown in Figure 7.9. The model is inspired by the ResNet architectures [18]. The model has multiple skipped connections that feed earlier output to the model. It allows us to create a deeper model that can be optimized for higher accuracies. We have also chosen to use long short-term memory (LSTM) layers at the beginning and end; these are a form of RNNs. They are used to learning sequential patterns in the data. The excessive use of Batch Normalization is theorized to help the data calibrate across devices by constantly recentering the data and alleviating any variation due to differences in magnitude between the two data sources: WSC and Flex-EOG [15].

7.5.2 Spectrogram Learning

Spectrogram-based learning poses a challenge since it is not used regularly, but also possesses a lot of potential in ML. As explained in Chapter 7.3.4, the spectrogram shows the intensity of specific frequencies over time. It can be thought of as a heat map, where stronger frequencies are represented by altering color and brightness. Therefore, time series-based data can be compactly represented in an image-based format.

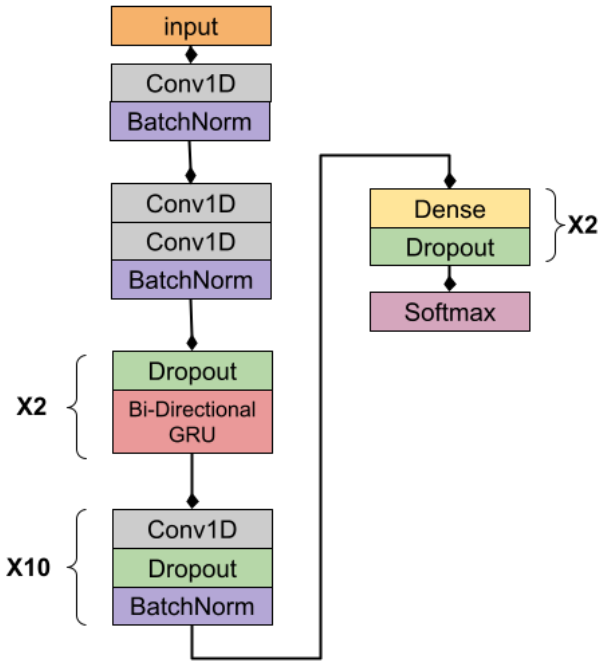


Figure 7.10: Spectrogram model network structure.

Learning from a spectrogram requires image detection models such as LeNet, AlexNet [12]. However, the strongest spectrogram-based models come from speech detection algorithms. Many NLP models utilize spectrograms to detect words and phrases. After experimenting with different types of models including LeNet, AlexNet inspired CNNs: we found that the models inspired by speech detection obtained the highest accuracy.

As shown in Figure 7.10, our spectrogram model borrows features from speech detection models such as Baidu DeepSpeech 2 [20]. The spectrogram model uses gated recurrent units (GRUs). It is a RNN based unit that is similar in function to LSTMs, both are "gated". However, GRUs are more computationally efficient due to their less complex structure. We use a bidirectional version of GRUs. This means that our GRUs have two structures. One that operates in the forward time and one operates backward in reverse time. A bidirectional RNN structure is shown in Figure 3.4 and is discussed in more detail in Chapter 3.2.2. Using the bidirectional GRUs, we increase our computational cost, as we have to process the data in both time directions. However, it is better than using a Bidirectional LSTM, which would be even more computationally expensive.

The remainder of our spectrogram model is very deep NN that uses CNNs to process the sequential information passed from BiGRUs. We also incorporate Batch Normalization to ensure that our model does not create extremely large and computationally expensive weights, as this leads to higher computational costs and poor model accuracy [15]. We also use a healthy amount of dropout to ensure that the model has regular variation. This is done to ensure that the gated units and CNNs do not overfit [56].

Since our spectrogram model works well with BiGRUs -see Chapter 8, we can theorize that within each sleep state our eye movements have multiple patterns. Similar to when we talk, there are multiple ways of saying the same phrase.

For example, the following phrases all have the same meaning: "She opened the door",

"The door was opened by her", and "The door was unlocked by that girl". So, similarly, it is possible to say that sleep states, such as REM sleep, have multiple sequences. They all have the same "meaning", i.e, that the person is in REM. **But, each sequence is different from each other.**

Chapter 8

Results

We will discuss the performance of our models to find the model that best fits our desired criterion; high accuracy, performance across classes, and mobile friendly (i.e. lightweight). The models are evaluated based on the following criteria, in order of priority: accuracy, F1 score, precision, recall, and feasibility in mobile apps.

8.0.1 WSC Dataset

Preprocessing	Accuracy	F-1 Score	Precision	Recall
Raw	0.772	0.7497	0.7532	0.7717
Wavelet	0.804	0.8028	0.8025	0.8043
Welch	0.916	0.9129	0.9131	0.9158
Spectrogram	0.867	0.8602	0.8651	0.8668

Table 8.1: Results of the models in the **WSC** Dataset.

As we can see above, our most promising model uses the Welch transform. However, our spectrogram and wavelet models are also strong. These three models have state of the art accuracy for sleep classification using EOG signals.

The Welch model has an accuracy of 91.6%, this is the strongest accuracy we achieved across all of our preprocessing models. As stated before, this could be due to the high compression rate of the Welch transform. It helps our model by extracting the key features of the data. The Welch transform is also a good option for resource-constrained systems, due to its ability to lower the memory cost of our dataset.

One proof that our preprocessing helps is looking at the accuracy of our model with raw data. It is 77% and does not even exceed 80%. This shows us that preprocessing is necessary for EOG data. Biosignals often exhibit noise, random spikes, and random variations. Preprocessing eliminates the unnecessary data variability that is prevalent in

biosignal datasets. It is also key to make our model perform faster, since less data have to be processed. The Welch transform takes it a step further by extracting only 129 of the 3000 datapoints present in each epoch. It is very succinct, and this helps our model to distinguish between different sleep states.

Given the poor performance of our Raw model on the WSC dataset, we choose not to apply it to our Flex-EOG data for transfer learning. This is so because the poor performance would also apply to the Flex-EOG data.

8.0.2 Flex-EOG Dataset

Preprocessing	Augmentation	Accuracy	F-1 Score	Precision	Recall
Wavelet	SMOTE	0.708	0.6478	0.6205	0.7077
	Fourier Transform	0.735	0.7309	0.787	0.735
	Gaussian Noise	0.701	0.698	0.708	0.701
Welch	SMOTE	0.822	0.8214	0.8217	0.8243
	Fourier Transform	0.761	0.759	0.771	0.761
	Gaussian Noise	0.744	0.745	0.753	0.744

Table 8.2: Results of models on **Flex-EOG** Dataset.

As shown in Table 8.2, the **Welch** transform is the best performer. It consistently gives us better metrics regardless of the type of augmentation used. Regarding the type of augmentation, **SMOTE** and **Fourier transform** both create strong models. The strong F-1 score, precision, and recall values show that augmentation works by allowing us to create balanced datasets that are not biased towards larger classes.

Augmentation	Accuracy	F-1 Score	Precision	Recall
SMOTE	0.752	0.754	0.786	0.752
Fourier	0.744	0.740	0.753	0.744
Gaussian Noise	0.744	0.738	0.754	0.744

Table 8.3: Results of spectrogram models on **Flex-EOG** Dataset.

Spectrogram models are surprisingly strong performers. Compared to other types of preprocessing, such as Welch, the spectrogram is not the best performer. However, spectrograms are a very experimental field in machine learning, so obtaining state of the art results is very promising. SpecAugment is one of the reasons for the strong performance [54]. By being able to randomly populate our dataset, we are able to overcome our bottleneck of data variability. Spectrogram models are an interesting topic to explore for future work. Spectrogram models can be further improved, and the use of spectrogram images is a good way to transmit datasets across IoT devices. Many classical ML problems, such as CIFAR-10, are all image-based classification issues. With that said,

further development to strengthen spectrogram models and associated techniques can prove to be fruitful in future classification problems.

8.0.3 Positive effects of augmentation

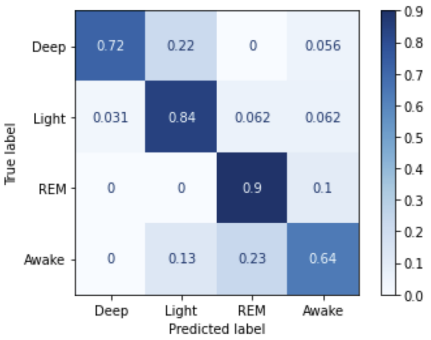
(a) Without augmentation.

(b) With augmentation.

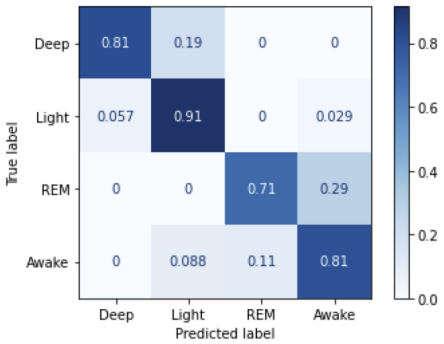
Figure 8.1: Comparing the effects of augmentation to our model performance.

Augmentation has a strong effect on the performance of our model. As shown in Figure 8.1, the model has a higher sensitivity to weaker classes after applying augmentation. In this case, we applied SMOTE augmentation to our spectrogram model. Light sleep classification had notably better performance after applying augmentation. The deep sleep classification accuracy also improved. The accuracy of REM decreased, but the model became more balanced. Overall, applying data augmentation is a necessary step in unbalanced datasets, where models are prone to bias. This is the case in sleep detection, where there are severely unbalanced data sets.

8.0.4 Comparison of the two strongest models



(a) Spectrogram model with SMOTE.



(b) Welch model with SMOTE.

Figure 8.2: Two best performers.

Model	Augmentation	Accuracy	F-1 Score	Precision	Recall	DCR
Welch	SMOTE	0.822	0.8214	0.8217	0.8243	23.25
Spectrogram	SMOTE	0.752	0.754	0.786	0.752	3.0

Table 8.4: Comparison of the metrics of our two best models, where DCR stands for data compression ratio.

Both models use SMOTE as their data augmentation choice; from this we can deduce that for sleep detection SMOTE is a stronger performer than GWN and Fourier.

By analyzing the two confusion matrices in Figure 8.2 and the statistics in Table 8.4, it is clear that the Welch model with SMOTE is superior. It has higher accuracy, F-1 score, precision, and recall.

The Welch model has a high data compression ratio for raw data; however, we cannot fully compare the data compression rates of the spectrogram image with the raw data. The spectrogram image compression will vary depending on the format of the image and the mobile device. We also note that the spectrogram can compress multiple epochs of data into one single image. This is an interesting discovery that can be explored in future work. At this point in time, the Welch model is superior for mobile apps. Not only is it a lightweight option, but spectrogram image processing for mobile applications is still not optimized for resource-constrained devices such as mobile. In summary, the Welch model with SMOTE is our strongest model. It has superior performance and is the best performer for resource-constrained applications [57].

8.1 Explanation of Results

The Welch model is the best performer. The reason behind the performance of the Welch lies in its data compression ratio. Since it has the highest data compression ratio of 23.25, this means that our initial 3000 data points are expressed in only 129 datapoints after using the Welch transform. Not only that, but by using the Welch transform we are reducing our likelihood of noise interfering with each model. Since the Welch model is the power spectral density, it can be argued that it is the transform with the greatest reduction of noise. Since we estimate the density of frequencies, relevant EOG recordings will more likely

be represented in the Welch signal. The noise will be scattered and filtered out. Although the wavelet transform also uses the frequency domain, it is not able to express our signal as a density nor does it compress the signal size. Compared to Welch and wavelet, the raw data will give the most noise. This is why raw data is the worst performer. Since we are feeding the neural network with so few datapoints, it is forced to learn the "key features" of each epoch. Furthermore, our model is less likely to be influenced by random noise.

Although all data augmentation types were strong, SMOTE was extremely effective for Welch models. The reason for the strong performance of SMOTE on the Welch is due to the nature of our Welch data. Since we have less noise and only 129 datapoints per epoch, we are controlling for external factors that can hinder the performance of SMOTE. Using too much or too noisy data decreases the effectiveness of a nearest-neighbor approach. These factors can interfere with our cluster boundaries, there can be data of different sleep states whose signals are too close to each other leading to poor clustering. Distorted clusters and noise can lead to lower-quality synthetic data [58, 59].

Given the nature of our sleep data from EOG, we know that sleep states often have overlapping signals: Light and deep sleep are often misclassified as one another; this can be seen even in our 2 Best models in Figure 8.2. Our models often predict light sleep as deep sleep and vice versa. We also know that our Raw EOG data is extremely noisy, which is the reason it had the worst performance. It is also noteworthy that the wavelet had the lowest precision and recall when compared to the spectrogram and Welch. This is shown in

Preprocessing	Avg. F1 Score	Avg. Precision	Avg. Recall
Wavelet	0.692	0.705	0.715
Welch	0.775	0.782	0.776
Spectrogram	0.744	0.764	0.746

Table 8.5: The average F1 score, precision, and recall for each of the models with different preprocessing types. We can see that the wavelet was the worst performer. Welch was the best performer, closely followed by the spectrogram.

Table 8.5. The poor precision and recall for the wavelet means that it had allot of false positive and less true positive predictions compared to the other two models. The wavelet transform does not process our data to create sufficient boundaries and distinctions between different sleep states. It is apparent why SMOTE was a poor candidate for the wavelet, due to its weaker preprocessing we were left with noise and faulty decision boundaries. This in turn led to low-quality synthetic data being created and poor model performance. For the wavelet models, SMOTE was the weakest preprocessing. Since GWN and Fourier did not rely on cluster boundaries, they had better metrics.

The two other types of data augmentation used, Fourier and GWN, are less effective because they do not aim to create an approximation of our data. Rather, they aim to tweak existing datapoints. These techniques can work on data such as the wavelet transform, which are more prone to distorted clustering when using SMOTE.

Finally, the spectrogram model is a strong model with room for growth. Since it is a relatively new technique, it presents more difficulties in creating effective models. Since we are feeding our spectrogram model an image, we are presenting it with far more datapoints

than the Welch model. Our model can be prone to learning irrelevant details and bias towards phantom patterns, that is, learning patterns that are not, in fact, a reality [60]. The Google Nest Hub team were able to create an effective spectrogram model, but it was after completing more than a million hours of sleep recordings. We theorize that because Google had so much data the model was less prone to making false assumptions from the spectrograms. Since the phantom patterns will appear only in a few data points in a sea of data, it will not be able to bias the model. Thus, spectrogram models require more data to be able to successfully overcome their shortcomings. Finally, there is still room for new research in this field to improve its effectiveness.

8.2 Implications

There are a few key points to highlight from the results and the overall thesis. First, we have a proof of concept that transfer learning combined with data augmentation are effective techniques that can be applied together to train machine learning models. Especially in the context of embedded medical devices, we have seen that devices that measure biopotentials can have effective ML models made for them. From the research in this thesis, we have established a proof of concept that we are able to create strong ML models that work on a biopotential device. Through future research, we may be able to establish a trend that shows the effectiveness of transfer learning and data augmentation to train ML models on biopotential devices.

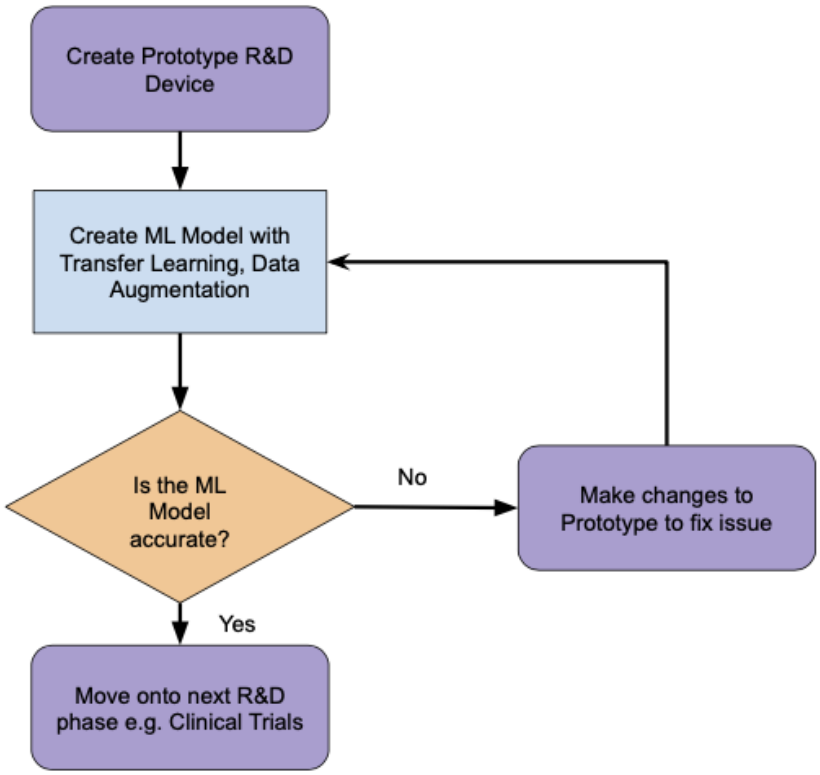


Figure 8.3: Flowchart depicting the steps researchers can take during the development of their products. By adding a step of creating an initial ML model we are able to go back and iteratively improve our device functionality before entering the next phase of research and development (R&D).

Second, this allows us to develop an iterative process in which researchers can quickly make iterative changes early in the development of their products. As depicted in Figure 8.3, researchers are able to go "back to the drawing board" more easily. As shown in Figure 8.3, as soon as we can gauge the performance of our prototypes using our ML model, we can make changes to our prototype if necessary. In the field of research, this has many implications.

We are able to test the viability and functionality of many experimental devices. This has a huge cost- and time-saving benefit since this is done before the clinical trials phase. Using this, we are able not only to create functional systems, but also to make iterative changes at a much more effective rate. This can help us to create better products. Thus, this is a pathway for researchers to ensure that their product is functional before moving forward with more costly development phases.

We strongly suggest that the researchers make multiple models with multiple augmentation and preprocessing techniques applied, as done in this thesis. By doing so, they are able to better understand the performance of their prototype device, rather than relying on the performance of a single model alone.

Chapter 9

Conclusion

We were able to successfully build a model that can classify sleep states. The key takeaway from the research conducted here is that, by using the tools available to us, we are able to build useful models despite a lack of clinical data.

In this case, we built an iOS-cloud infrastructure and used transfer learning of existing datasets. We also applied a wide variety of data augmentation techniques to increase the diversity and size of our dataset. From that we created a sleep classifier that has accuracies close to SOTA. The techniques used in this thesis are useful for experimental wearable devices that are in the R&D stage. In conclusion, we are able to create strong machine learning-based models for our EOG device. Our strong performance can be seen in the performance of our Welch model. Its accuracy of 82.2% is within the same range as the SOTA of 82.1% in EOGNET.

9.0.1 Future Work

There is much to explore in future work. In the case of the models, they can be made even deeper with more layers. They can also contain new types of data preprocessing. You can see some prospective preprocessing types in the literature review of this thesis; such as FBANKS, Mel-spectrogram.

Furthermore, we can investigate how to make the models more mobile friendly, by using the techniques mentioned: Deep Compression, HoG. Lastly, we can find suitable machine learning models by combining multiple types of model architecture and preprocessing to find the best fit for our goals.

Overall, there is still a lot of room for future exploration and development of the research explored in this thesis.

Bibliography

[1] Cleveland Clinic. Sleep apnea: Causes, symptoms, tests amp; treatments, July 2020.

[2] Aakash K Patel, Vamsi Reddy, and John F Araujo. Physiology, sleep stages. In *StatPearls [Internet]*. StatPearls Publishing, 2022.

[3] Richard B. Berry, Rita Brooks, Charlene Gamaldo, Susan M. Harding, Robin M. Lloyd, Stuart F. Quan, Matthew T. Troester, and Bradley V. Vaughn. AASM scoring manual updates for 2017 (version 2.4). *Journal of Clinical Sleep Medicine*, 13(05):665–666, May 2017. https://doi.org/10.5664/jcsm.6576.

[4] Shibam Debbarma and Sharmistha Bhadra. A lightweight flexible wireless electrooculogram monitoring system with printed gold electrodes. *IEEE Sensors Journal*, 21(18):20931–20942, September 2021. https://doi.org/10.1109/jsen.2021.3095423.

[5] Laura J. Frishman. Electrogenesis of the electroretinogram. In *Retina*, pages 177–201. Elsevier, 2013. https://doi.org/10.1016/b978-1-4557-0737-9.00007-2.

[6] Raman K. Malhotra and Alon Y. Avidan. Sleep stages and scoring technique. In *Atlas of Sleep Medicine*, pages 77–99. Elsevier, 2014. https://doi.org/10.1016/b978-1-4557-1267-0.00003-5.

[7] Roger Grosse. *A Closer Look at AlexNet*. University of Toronto, 2018.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[9] Zhe Yao, Vincent Gripon, and Michael G. Rabbat. A massively parallel associative memory based on sparse neural networks, 2013.

[10] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. https://doi.org/10.1038/323533a0.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran

Associates, Inc., 2017. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[13] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, dec 2010.

[14] Pérez-Enciso and Zingaretti. A Guide for Using Deep Learning for Complex Trait Genomic Prediction. *Genes*, 10(7):553, July 2019. https://doi.org/10.3390/genes10070553.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[16] Christopher Olah. *Understanding LSTM Networks*. Colah's Blog, 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[17] Amin Emad. *ECSE 552: Deep Learning*. McGill University, 2021.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[19] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[20] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Vaino Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 173–182. JMLR.org, 2016.

[21] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *Artificial Neural Networks and Machine Learning – ICANN 2018*, pages 270–279. Springer International Publishing, 2018.

[22] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

[23] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1990. https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

[24] Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1989. https://proceedings.neurips.cc/paper/1988/file/1c9ac0159c94d8d0cbedc973445af2da-Paper.pdf.

[25] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1993. https://proceedings.neurips.cc/paper/1992/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf.

[26] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley*

Symposium on Mathematical Statistics and Probability, volume 1, pages 281–297. University of California Press, 1967.

[27] Alex Omid-Zohoor, Christopher Young, David Ta, and Boris Murmann. Toward always-on mobile object detection: Energy versus performance tradeoffs for embedded HOG feature extraction. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(5):1102–1115, May 2018. https://doi.org/10.1109/tcsvt.2017.2653187.

[28] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. IEEE. https://doi.org/10.1109/cvpr.2005.177.

[29] Sherzod Bek. PyTorch RGB TO GRAY. Pytorch Forums, 2019.

[30] Brandon Rohrer. How to convert an rgb image to grayscale. https://e2eml.school/convert_rgb_to_grayscale.html.

[31] Mirco Ravanelli, Titouan Parcollet, Peter Plantinga, Aku Rouhe, Samuele Cornell, Loren Lugosch, Cem Subakan, Nauman Dawalatabad, Abdelwahab Heba, Jianyuan Zhong, Ju-Chieh Chou, Sung-Lin Yeh, Szu-Wei Fu, Chien-Feng Liao, Elena Rastorgueva, François Grondin, William Aris, Hwidong Na, Yan Gao, Renato De Mori, and Yoshua Bengio. Speechbrain: A general-purpose speech toolkit, 2021.

[32] InteraXon. Muse EEG Band, 2018.

[33] Stanisław Saganowski, Przemysław Kazienko, Maciej Dziezyc, Patrycja Jakimów, Joanna Komoszynska, Weronika Michalska, Anna Dutkowiak, A Polak, Adam Dziadek, and Michal Ujma. Review of consumer wearables in emotion, stress, meditation, sleep, and activity detection and analysis. *arXiv preprint arXiv:2005.00093*, 2020.

[34] Michael Dixon, Logan Schneider, Jeffrey Yu, Jonathan Hsu, Anupam Pathak, D Shin, Reena Singhal Lee, Mark Rajan Malhotra, Ken Mixter, Mike McConnell, James Taylor, and Shwetak Patel. Sleep-wake detection with a contactless, bedside radar sleep sensing system. Technical report, 2021.

[35] Henry Rimminen, Ali M Amin, Timothy L Weadon, Chuo Yindar, Zeng Zijing, and Erno Klaassen. On-bed differential piezoelectric sensor, Feb 2021.

[36] Atena Roshan Fekr, Majid Janidarmian, Katarzyna Radecka, and Zeljko Zilic. Respiration disorders classification with informative features for m-health applications. *IEEE Journal of Biomedical and Health Informatics*, 20(3):733–747, May 2016. `https://doi.org/10.1109/jbhi.2015.2458965`.

[37] AR Fekr, K Radecka, and Z Zilic. Tidal volume variability and respiration rate estimation using a wearable accelerometer sensor. *2014 4th International Conference on Wireless Mobile Communication and ...*, 2014. 32 cites: https://scholar.google.com/scholar?oi=bibs&hl=en&cites=17336154483766844419.

[38] Jialei Yang, James M. Keller, Mihail Popescu, and Marjorie Skubic. Sleep stage recognition using respiration signal. In *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, August 2016. https://doi.org/10.1109/embc.2016.7591322.

[39] J. Fan, C. Sun, M. Long, C. Chen, and W. Chen. EOGNET: A Novel Deep Learning Model for Sleep Stage Classification Based on Single-Channel EOG Signal. *Front Neurosci*, 15:573194, 2021.

[40] Junming Zhang, Ruxian Yao, Wengeng Ge, and Jinfeng Gao. Orthogonal convolutional neural networks for automatic sleep stage classification based on single-channel EEG. *Computer Methods and Programs in Biomedicine*, 183:105089, January 2020. https://doi.org/10.1016/j.cmpb.2019.105089.

[41] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.

[42] Awni Y. Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H. Tison, Codie Bourn, Mintu P. Turakhia, and Andrew Y. Ng. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature Medicine*, 25(1):65–69, January 2019. https://doi.org/10.1038/s41591-018-0268-3.

[43] Michael Dixon and Reena Singhal Lee. Contactless Sleep Sensing in Nest Hub. *Google AI Blog*, March 2021.

[44] Olave E. Krigolson, Chad C. Williams, Angela Norton, Cameron D. Hassall, and Francisco L. Colino. Choosing MUSE: Validation of a low-cost, portable EEG system for ERP research. *Frontiers in Neuroscience*, 11, March 2017. https://doi.org/10.3389/fnins.2017.00109.

[45] Cassandra M. Wilkinson, Jennifer I. Burrell, Jonathan W. P. Kuziek, Sibi Thirunavukkarasu, Brian H. Buck, and Kyle E. Mathewson. Application of the muse portable EEG system to aid in rapid diagnosis of stroke. June 2020. https://doi.org/10.1101/2020.06.01.20119586.

[46] Olave E. Krigolson, Chad C. Williams, and Francisco L. Colino. Using portable EEG to assess human visual attention. In *Lecture Notes in Computer Science*, pages 56–65. Springer International Publishing, 2017. https://doi.org/10.1007/978-3-319-58628-1_5.

[47] Guo-Qiang Zhang, Licong Cui, Remo Mueller, Shiqiang Tao, Matthew Kim, Michael Rueschman, Sara Mariani, Daniel Mobley, and Susan Redline. The national sleep research resource: towards a sleep data commons. *Journal of the American Medical Informatics Association*, 25(10):1351–1358, May 2018. https://doi.org/10.1093/jamia/ocy064.

[48] T. Young, M. Palta, J. Dempsey, P. E. Peppard, F. J. Nieto, and K. M. Hla. Burden of sleep apnea: rationale, design, and major findings of the Wisconsin Sleep Cohort study. *WMJ*, 108(5):246–249, Aug 2009.

[49] Kendall K. Morgan. What is polysomnography (psg)?

[50] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[51] P. Welch. The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, June 1967. `https://doi.org/10.1109/tau.1967.1161901`.

[52] David Salomon. *Data compression*. Springer, London, England, 4 edition, January 2007.

[53] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[54] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin Dogus Cubuk, and Quoc V. Le. Specaugment: A simple augmentation method for automatic speech recognition. In *INTERSPEECH*, 2019.

[55] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. `https://arxiv.org/abs/1409.1556`.

[56] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[57] Mian Hamza, Sharmistha Bhadra, and Zeljko Zilic. Sleep stage detection on a wearable headband using deep neural networks. In *Internet of Things*, pages 187–198. Springer International Publishing, 2022.

[58] N. Japkowicz. Class imbalance: Are we focusing on the right issue? *Notes from the ICML Workshop on Learning from Imbalanced Data Sets*, page 17 â 23, 2003. Cited by: 108.

[59] Vicente GarcÃa, Jose SÃ¡nchez, and Ramon Mollineda. An empirical study of the behavior of classifiers on imbalanced and overlapped data sets. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4756 LNCS:397 â 406, 2007. Cited by: 77.

[60] Emilija Perković. The phantom pattern problem: The mirage of big data,. *The American Statistician*, 76(1):86–87, January 2022.