Deep Learning with EEG Data

Bereket Tekeste

Thesis submitted for the degree of
Master in Applied Computer and Information Technology: AI
30 credits

Department of Computer Science
Faculty of Technology, Art and Design

OSLO METROPOLITAN UNIVERSITY

Spring 2023

Deep Learning with EEG Data

Bereket Tekeste

Deep Learning with EEG Data

http://www.oslomet.no/

Printed: Oslo Metropolitan University

Contents

List of Figures

List of Tables

Abstract

Electroencephalogram (EEG) data has shown great promise but requires sophisticated methods due to the complex spatial and temporal patterns found in such data, so this research was conducted with the objective to investigate the efficiency of different types of deep learning models that includes Convolutional Neural Networks (CNNs), Long Short-Term Memory(LSTMs), Hybrid CNNs and Siamese LSTMs in classifying EEG data associated with schizophrenia. What was demonstrated was that these models were able to capture the intricate patterns within EEG data exceptionally well leading to accurate predictions about the patient's condition, and results from evaluating different models indicate that the Hybrid (CNN+LSTM) architecture offers optimal suitability for this specific application because of improved outcomes. Important implications regarding the improvement of diagnosis and treatment for schizophrenia and other neurological disorders can be achieved through deep learning models as shown by this research.

Acknowledgement

My sincerest appreciation extends to my advisor, Mohamed Radwan whose dedication and support have made it possible for me to make progress on this thesis project. It has been an honor to receive your diligence, guidance, and words of encouragement throughout the process.

Furthermore, I am thankful to the faculty and staff at Oslo Metropolitan University's Computer Science Department for providing an educational environment filled with learning opportunities guaranteed for intellectual growth and success.

I want to express gratitude towards all those who had supported me over this period starting from my family: My parents who continue to cheer me up; To special mention of who goes out to my lovely wife Mahdi who never fails to understand me no matter how difficult things become you have been indispensable part during these times. Finally, yet importantly I owe lots of thanks to each person mentioned above as without them would not have made it this far.

Chapter 1

Introduction

Including natural language processing and voice recognition among others, DNNs have shown remarkable results in multiple areas including image categorization. DNNs can achieve accurate predictions by learning from massive, annotated datasets despite the challenge of creating and analyzing large datasets, but in cases of scarce or insufficient resources, data collection, and analysis might take considerable time. DNNs can learn from a few annotated samples with N-shot learning without requiring large datasets this is a promising approach.

The goal of this research is to examine if n-shot learning can be used with time series data which is a group of observations taken at regular intervals over time, and it will compare the performance of n-shot learning against traditional DNNs. Time series data is unique compared to images or text. It's used in all sorts of fields, like finance, meteorology, and healthcare. But because there are often connections between events over time variables and conditions that affect its efficacy such as similarity metrics and number of samples per class. This research aims at answering three fundamental questions:

- Under what circumstances can we accurately classify or forecast time-series using n-shot learning?
- What factors contribute to affecting the efficiency of n-shot-learning techniques for these cases?
- What compromises one must make if they want an improved performance through altered/extended versions of already developed models?

we will analyze the obtained results and assess their performance using recognized evaluation measures. This will enable us to address commonly raised questions in this field. Additionally, ascertaining both benefits and shortcomings of using n-shot learning for time-series data is crucial in making consequential judgments while ruminating future investigations' probable outcomes. To accurately measure the efficacy of margin-based techniques versus traditional ones concerning future studies' recommendations, several metric assessment tools shall be utilized for thorough performance analysis.

1.1 Background

Deep neural networks (DNNs) are foundational components of modern artificial Intelligence applications. These models leverage their multi-layered structure to identify intricate patterns and relationships by connecting numerous nodes in their internal structure. With that ability, they have proven highly effective in tasks such as image classification, natural language processing (NLP), and speech recognition [1].

The accuracy of a Deep Neural Network (DNN) model is largely determined by the size and quality of the dataset used. For training, datasets contain sample inputs related to the problem which are associated with labels representing the correct output for those inputs [2]. The DNN adjusts connections between nodes based on discrepancies between predicted outputs and correct labels, thereby improving its performance when applied to new data sets. In other words, it refines predictions made by recognizing patterns in existing data sets by means of constant updates to relevant parameters [3].

Collecting and labeling extensive datasets can be a challenging task to accomplish as it requires significant amounts of time, effort, and resources [4]. Even if the data is available, there may still be issues such as data scarcity or imbalance which could reduce the effectiveness of a Deep Neural Network (DNN). Furthermore, acquiring the needed amount might turn out to be expensive depending on the domain. The larger and more diverse a dataset is however, the higher level of accuracy DNNs can reach in their predictions [3].

The study by[5] delved into how class imbalances can cause decreased performance for Convolutional Neural Networks (CNN). These researchers analyzed several strategies geared towards correcting the data discrepancy including methods that operated at a data level such as oversampling or undersampling, algorithmic level tactics which they termed cost-sensitive learning, and likewise hybrid approaches formulated out of combining many different techniques.

To address the problem of limited or imbalanced datasets in deep neural networks (DNN) numerous approaches have been developed. Some of these include Transfer learning, N-shot learning, data augmentation, and active learning [6, 7], which are used for leveraging outside knowledge and data as well as generating synthetic training data in order to build more robust models that perform better on unfamiliar datasets. These techniques can be an effective means of augmenting existing resources in deep neural networks with related but less biased or smaller sources thus yielding accurate outcomes amid scarce information.

Mental illness can have a powerful effect on our minds, emotions, and behavior. Thankfully with the right diagnosis and proper treatment plans in place, psychological disorders can be managed for improved outcomes [8]. One of the more effective ways to measure brain activity is through Electroencephalography(EEG) which is used as an aid to diagnose epilepsy, sleep disturbances, and Attention deficit hyperactivity disorder(ADHD)[9].

In spite of this, analyzing this data can be challenging due to its varied nature between individuals over

time; furthermore, exterior features such as skull conductivity might affect reading accuracy [10] making it a complex process requiring neurological expertise. To that end, researchers are employing advanced machine learning models like Deep Neural Networks(DNNs), Siamese networks, and N-shot learning technologies in diagnosing schizophrenia through detailed analysis of EEG signals research showing great potential when dealing with illnesses affecting thousands around the world[11].

N-shot Learning doesn't require vast amounts of labeled datasets but rather a few samples from trained professionals who help train AI algorithms better understand mental health afflictions while still providing meaningful results amidst a lack of luster datasets thus giving us hope towards efficiently tackling future diseases also.

Recent advances in one-shot learning algorithms for EEG data classification and signal detection have brought about new prospects, thanks to their unique architectural design that leverages similarity comparison between inputs rather than computing absolute values for each input in isolation making Siamese neural network architectures extremely powerful even when dealing with unbalanced and small datasets. By incorporating techniques like convolutional neural networks (CNNs) or long short-term memory (LSTM), researchers have obtained a better understanding of how individual patients respond differently in diverse scenarios. Similarly, transfer learning and meta-learning strategies add additional layers of difficulty but have shown excellent promise in producing very precise results even with a lack of resources; there are also alternatives through Generative-Adversarial Networks or GANs that work well with these given application domains. The findings from these studies show that one-shot learning can be achieved through the accessible use of EEG data, and from identifying patients to classifying objects and even recognizing faces despite small datasets Siamese networks can do it all. To make sure the model can perform its best amid challenges like overfitting or computational complexity approaches such as transfer learning[12] and data augmentation [13] may be employed to yield desired results. Moreover, studies on EEG signal detection associated with epilepsy seizures show success through Siamese architectures [14] while Li Guan Li's research has revealed that event-related potentials can also be detected via these models[15]. This method owes much of its acclaimed power to contrastive loss functions which further deepen similarity among samples from similar labels; hence establishing it as useful in many domains where accuracy is key.

Consequently, the use of Siamese Network models has shown promising results with regard to the diagnosis and treatment of neurological disorders. One-shot learning utilizing EEG data was capable to attain impressive performance for the diagnosis and classification of depression [11] and an autism spectrum disorder(ASD) [16]; A few-shot learning to EEG classification achieved excellent performance [17]. Improvements in feature extraction facilitated by Spuler's novel loss function provided a more accurate way of diagnosing neurological diseases than what had ever been previously known [18]. Better understanding provides expectations for future healthcare improvements through simpler diagnostic procedures as well as much fewer human biases.

The thesis is structured as follows: This chapter 1 presents the motivation and introduction to the topics. Chapter 2 offers an in-depth background and state-of-the-art review, discussing the relevant theories and experimental considerations for machine learning and deep learning tools. Chapter 3 outlines the methodology and the dataset, describing the proposed approach and the steps taken to achieve the research objectives. Chapter 4 presents the results while Chapter 5 presents the summary and outlook of the thesis.

Chapter 2

Theory

> ***AI and machine learning are transforming every aspect of our lives, from healthcare to transportation to entertainment.***
>
> JENSEN HUANG, CEO OF NVIDIA

Machine learning is a branch of artificial intelligence that uses the process of training a model on data to identify hidden patterns that can be used to make predictions. The machine learning process consists of 3 main components as shown in Figure: the model, the optimizer, and the dataset. The model includes a function for inputting data which it processes according to adjustable parameters; this output is used by the optimizer during one or multiple optimization steps. The optimizer adjusts these parameters to improve performance while relying on real-world data as provided by the stringently pre-processed datasets for accurate representation in machine understanding. By using this technique, computers are able to imitate certain behaviors of systems based on their given properties after running through an extensive and iterative setup program [3].

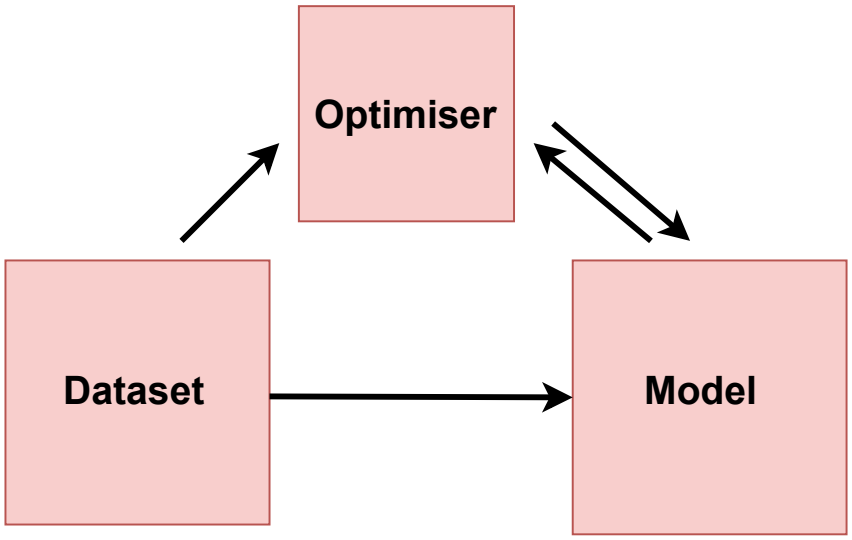

Figure 2.1: In machine learning, information is typically conveyed from the dataset to the optimizer only in learning techniques that involve feedback. Figure adapted from Ref.[19].

The optimization method used in machine learning includes different approaches such as unsupervised, reinforcement, and supervised. Unsupervised learning emphasizes deciphering patterns without reference to known, predetermined output values or labels [20]. Reinforcement and unsupervised learning approaches use feedback to guide the learning process and develop predictive models. Reinforcement learning utilizes evaluative feedback from a reward system to explore all possible inputs, actions, or outputs, which can be a slow and exhaustive process [21].

Supervised learning calls on data with considerations made about true output labels or values to converge on a solution faster. Deep Learning attempts to fit complex results by training multiple models one after the other in an effort to achieve a specific goal from the initial input signals received.

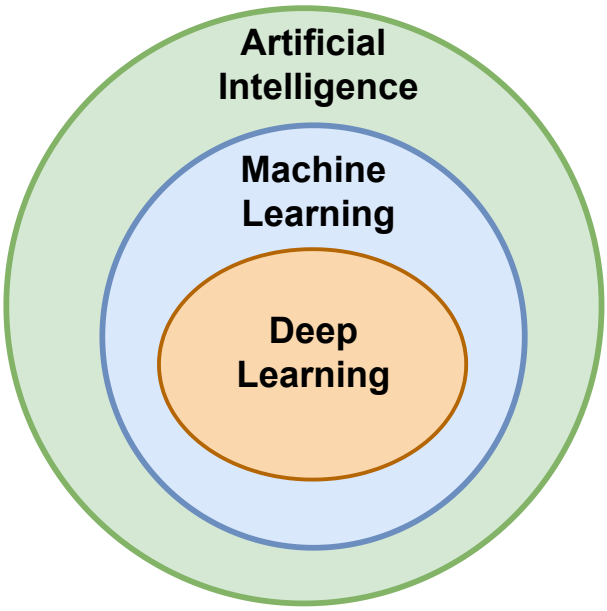

Figure 2.2: The hierarchical relationship between AI, ML, and deep learning. Deep learning is a subset of machine learning, which is a subset of AI.

Artificial intelligence (AI), machine learning (ML), and deep learning are dramatically impacting the world today [22]. Artificial Intelligence is a greater concept that encompasses machines that can act with human-like intelligence across multiple domains. Machine Learning is a related field focused on developing algorithms to enable computers to learn autonomously from training datasets without explicit programming [23]. Deep Learning involves using multilayer neural networks to extract features and insights of large amounts of data in order to achieve more advanced results than those achievable by other methods. All three fields are closely intertwined; AI is the uppermost branch, from which stems machine learning followed by deep learning at the lowest level in the hierarchy as shown in figure 2.2. Combinedly these disciplines are leading to pioneering advancements in continually advancing operations such as image recognition, natural language processing, and autonomous automobiles [3]. The relationship between AI, ML, and deep learning is that deep learning is a subset of machine learning, which in turn is a subset of AI [23]. Their combined symbiosis further influences how technology interacts daily with humans' lives now more than ever before.

2.1 Model

2.1.1 Artificial neural networks

The McCulloch-Pitts model of a neuron has been improved over numerous decades and several technical updates have been added to the simple model such as adding time delay, introducing adaptive weights, incorporating additional inputs and outputs beyond binary signals, alteration of parameters or input functions to actively learn from changing data and introducing memory structures for further analysis or categorization of data [24], and a host of other technical updates that make this simple model more robust in its capabilities as an artificial brain cell. The seminal mathematical model for neurons developed by Warren McCulloch and Walter Pitts back in 1943 paved the way for significant developments in artificial intelligence, if the stimulus exceeded an established threshold value with no inherent delay in processing data then the original McCulloch Pitts Neuron could have one of its two outcomes: either a '1' or '0'. Simplistic as they may have been at first glance, however, they evolved into sophisticated models that more closely mirrored actual neurons over time.

Innovation has led to the implementation of advanced techniques such as introducing various delay times between inputs and outputs along with including several feedback or input variables that lead to better learning. With the incorporation of innovative techniques by artificial intelligence such as introducing variable delays between input and output signals or equipping learning algorithms to modify themselves based on acquired datasets; along with memory architectures that enable the preservation and become possible for artificial intelligence to bring many modern conveniences into our lives.

These advancements have made it so AI-generated results can be produced authentically by efficiently identifying patterns across larger amount of data. Ultimately these innovations are responsible for us being able to experience all the benefits artificial intelligence provides in modern society today. The McCulloch-Pitts model has proven to be an indispensable tool in a wide range of applications, from pattern recognition tasks like image and voice recognition, to binary classification such as spam filtering and diagnosing diseases. It's also been used to construct digital circuits that can perform logical operations similar to AND, OR, and NOT functions. This model has made its way into studies regarding artificial neural networks which are integral components of AI research. Neuroscientists have endeavored to make use of the model too, gaining insights about how biological neurons work as well as their role in processing information within the brain [24]. It's flexibility makes it valuable for research in many fields inspiring further exploration on many fronts.

The Perceptron

Research into artificial neural networks started with the development of the McCulloch-Pitts neuron in 1943, which was a mathematical model that represented how simple biological neurons collect and process information. This research was carried forward by Frank Rosenblatt when he proposed the first modern artificially intelligent network called "Perceptron" in 1957[25]. Perceptron introduced the concept of associating weights to input values, allowing more sophisticated operations compared to

standard function machines. The output generated from this model is calculated using an activation function that might be step or sigmoid depending upon user preference, and based on its final value will determine whether 0 or 1 should be returned as a result. Primarily used for binary classification problems, Perceptron's learning algorithm works by evaluating errors through gradient descent and then it adjusts parameters/weights so as to minimize them in successive iterations thereby providing accurate results over time. By utilizing such kinds of algorithms aided by systematic weighting systems, artificial neural networks have found much success across multiple applications with the potential to transform the decision-making process forever.

The Perceptron model was developed in the context of binary classification tasks, where two classes are defined as 1 (positive) and -1 (negative) [25]. The activation function of the Perceptron, $\phi(Z)$, can be written using two if statements as follows:

$$\phi(Z) = \begin{cases} 1, & \text{if } Z \geq \theta \\ -1, & \text{if } Z < \theta \end{cases}$$

where Z is the net input and θ is the defined threshold. If the net input Z is greater than or equal to the threshold θ, the activation function outputs 1, indicating a positive class prediction. Otherwise, if Z is less than the threshold, the activation function outputs -1, indicating a negative class prediction.

The activation function of the Perceptron, $\phi(Z)$, takes the net input, z, and predicts a class of 1 if z is greater than a defined threshold θ, and -1 otherwise. This is implemented using a variant of the unit step function [25]. The net input of a function is expressed as the linear combination of the input features and the corresponding weights.

The net input of a Perceptron, denoted as z, is expressed as the linear combination of the input features x and their corresponding weights w [25].

$$z = w_1 x_1 + ... + w_m x_m \tag{2.1}$$

where:

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

The Perceptron is used when there are two separate classes labeled as "positive" and "negative". The mathematical formula for the activation function of the Perceptron model takes inputs, represented by the variable $phi(z)$, to calculate if the predicted class should be a 1 or -1. In this calculation, an evaluation value known as θ acts as a predetermined threshold that must be exceeded in order for a class result of 1 to be determined. If however, the proposed input does not exceed θ, then a class result of -1 will be calculated. For example, values greater than (or equal to) θ will produce an output signal of 1 while lower values below it will generate an output signal of –1. To sum it up, the main goal of this revised

version of unit step function is accurately predicting outcomes based on given classification criteria with either '+' or '-' values assigned accordingly.

$$\phi(z) = \begin{cases} 1, & \text{if } z \geq \theta \\ -1, & \text{otherwise} \end{cases} \tag{2.2}$$

By introducing a new constant term $w_0 x_0$ to z, the definition can be simplified to (Rosenblatt, 1957):

$$z = w_0 x_0 + w_1 x_1 + ... + w_m x_m = w^T x \tag{2.3}$$

Where $w_0 = -\theta$ and $x_0 = 1$, leading to:

$$\phi(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{2.4}$$

In this formulation, w_0 is typically referred to as the bias unit. A diagram depicting the model's function is shown in Figure 2.3.

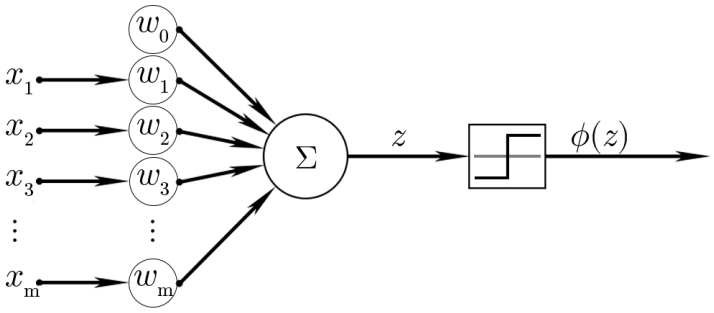

Figure 2.3: The perceptron model. Figure taken from Ref. [19].

Adaline (Adaptive Linear Neuron or Adaptive Linear Element)

Adaline (Adaptive Linear Neuron or Adaptive Linear Element) is an artificial neural network model first proposed in the early 1960s by Bernard Widrow and his student Ted Hoff [26]. It is a single-layer, feedforward architecture built on supervised learning principles that allow neurons to adapt their weights based on minimizing the discrepancy between predicted output and desirable result. This adaptation process follows what is known as the Widrow-Hoff rule which dictates that every neuron should be updated according to its contribution towards a given outcome. The Adaline model can therefore prove useful for both regression and classification problems across various domains. The novel paper introducing this concept described how components necessary for such networks could potentially be used to construct adaptive switching circuits, hence it provided substantial detail about how these systems are constructed from scratch along with important insights into how they should learn effectively.

Non-linear activation

Artificial neural networks incorporate nonlinear activation functions to introduce more complexity into the model. This is necessary when understanding interactions between input and output variables since linear connections cannot provide a complete representation of real-world problems. Typically, sigmoid, tanh (hyperbolic tangent),and Rectified Linear Unit (ReLU) are used as the activation function; however, many variations of ReLU can also be implemented such as Leaky ReLU, Parametric ReLU (PReLU), or Exponential Linear Unit (ELU). By employing these different types of non-linear activation functions within an ANN model it allows for deeper insight into complexities that could not have been revealed through only linear connections.

The **sigmoid function [27]** is a widely used non-linear activation function in artificial neural networks. It is defined as:

$$f(x) = \sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.5}$$

Given an input of x to a neuron, the sigmoid function is often used as it can map out any value of x between 0 and 1. This creates an S-shaped curve that demonstrates how even minor changes near the origin have large differences in the output. This advantage can be useful especially when dealing with probability and binary classification issues. Unfortunately, however, there are also some drawbacks to using this type of function such as the vanishing gradient issue - wherein inputs beyond certain limits lead to very small gradients hence slowing down learning performance and asymmetry at origin which also causes bias in models.

The **Hyperbolic Tangent (tanh) function** is a commonly used non-linear activation function that maps the inputs to the range [-1, 1]. It can be expressed as follow:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.6}$$

where x is the function's input. The function may be used to describe data negative and positive values since its output ranges from -1 to 1. Similar to the sigmoid function, the hyperbolic tangent function exhibits steeper gradients near 0 and is symmetric about the origin. By doing so, the vanishing gradient issue that might arise with the sigmoid function can be avoided and the model can learn more faster. When the input values are too high, the *tanh* function might still experience the expanding gradient issue.

The **Rectified Linear Unit (ReLU) function** is an often-used non-linear activation function that has a definition described as follows:

$$f(x) = max(0, x) \tag{2.7}$$

when the input value, x, is greater than zero then the output of the ReLU function carries that same value.

However, if x is less than zero the output will be set to zero. This simple yet computationally efficient form has made it a preferred choice for deep learning models within artificial neural networks and other related machine learning algorithms. A key advantage of using this type of function over functions such as sigmoid or tanh which have derivatives as fractions is that its derivative can range from 0 to 1 thereby allowing for faster gradient calculations during backpropagation by reducing computational complexity. The only potential issue with using ReLU could come in regards to neurons becoming “dead” after repeatedly receiving negative inputs forcing their outputs to remain zero due to their definitions. To counter this certain variants such as Leaky ReLU, Parametric ReLRU (PReLU), and Exponential Linear Units (ELUs) were developed which allow some small degree of variation while having positive gradient descent results even with negative input values.

The **Leaky Rectified Linear Unit (Leaky ReLU)** function is an alternative to the traditional Rectified Linear Unit (ReLU) that reduces the chance of "dead neurons" occurring when negative inputs are used. This improved version involves a slight slope for any input under 0, preventing neurons from entering into a dormant state. The Leaky ReLU takes on the form of:

$$f(x) = \begin{cases} ax & \text{when } x < 0, \\ x & \text{when } x \geq 0. \end{cases}$$

For Leaky ReLU it is important to set the parameter ’a’ - often referred to as "the leakage rate," which is typically set between 0 and 1; commonly being 0.01 or less with some exceptions depending on application demands. In comparison tests for deep learning models, The Leaky ReLU has been known to perform more efficiently than its predecessor due to its ability to better deal with what’s known as "dying neurons"; where certain neuron miss their opportunity in the zero zone and thus unable learn anything further downstream leading them become useless neuron cells within complex layers networks of artificial neural networks architecture.

The **Exponential Linear Unit (ELU) function** is a non-linear activation function used to improve the representational power of deep neural networks. This function offers an alternative to techniques like Rectified Linear Units or their variants and have demonstrated improvements in results. Essentially, ELU takes an input x and produces an output according to the formula:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha \cdot (\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \tag{2.8}$$

where α is a small positive number usually set at one. This activation behaves similarly to ReLU with sloping gradients for negative inputs but features its own lower asymptote that helps reduce issues associated with "dying ReLU". Taking advantage of these functions carries a lot of mathematical principles behind it, including linear activations, regularization schemes, and Adaline discrimination models. As such technology develops more into areas like image recognition, audio processing, and natural language analysis numerous improvement occur across many elements from predictive modeling

to data extraction.

2.1.2 Artificial neural networks (ANNs)

This research concentrates on the architecture of artificial neural networks, which are computer systems designed to emulate the structures and functionalities of biological brains. A model of an ANN is which consists of a number of interconnected nodes, each representing an artificial neuron. The structure of such systems involves input and output layers, as well as hidden layers that determine how information is processed through them.

In Figure 2.4 we can see, an ANNs typically consist of interconnected nodes simulating neurons; input and output layers that determine data entry points and outputs respectively; hidden layers where computing occurs; an activation function that explains how information is encoded within nodes; size elements defining the scope of computation carried out by parameters that can also be trained. The combination of these components works together to define appropriate networks for specific tasks.

Activation functions are key determining factors in neuronal connectivity which information is relayed between neurons. The number and size of trainable parameters within a network are also important in improving it's performance with regard to tasks like image recognition, natural language processing(NLP), or forecasting.

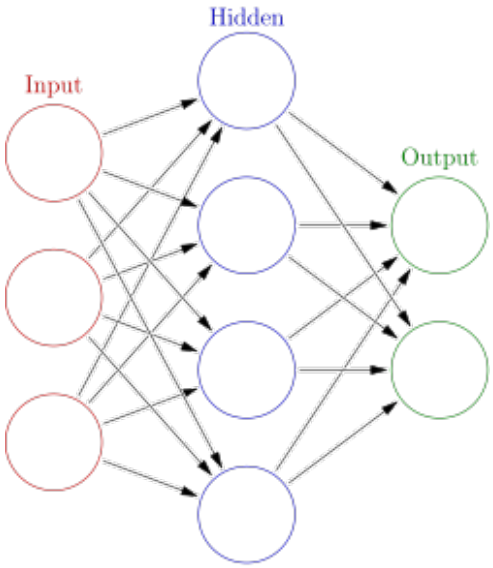

Figure 2.4: A schematic representation of a neural network. The network consists of interconnected nodes, each of which represents an artificial neuron. The connections between nodes are indicated by arrows, mimicking the structure of the brain's neurons [28].

Individual neuron

In artificial neural networks, each unit is an individual neuron that takes the same input but responds differently depending on its weight vector within that particular layer. The output value of a unit is determined by multiplying the value of its input strength with the weight vector and then applying a nonlinear transformation via an activation function. All units within a given layer will have the

exact same activation function yet they are able to produce different levels of output by being trained independently through backpropagation - a process where gradients of loss functions relative to weights are calculated and adjusted using stochastic gradient descent optimization methods in order to improve model performance.

Hidden Layers:

In a multilayer artificial neural network, there is one layer that lies between the input and output layers which we refer to as the hidden layer. This hidden layer can consist of any number of units, along with an additional bias unit. Each unit in a given hidden layer has connections to all units in the adjacent lower or higher layer - with each connection forming its own weight depending on how heavily it influences predicting the end result. All these weights create what is known as a Dense Layer effect – where every single neuron present exhibits activity that contributes to achieving some outcome from both outside information and within existing content stored earlier. The overall value generated by this collective combination then serves as a bias for the whole task in order for us to arrive at our desired output through multiple stages or 'hidden' computations from our neuronal layers before arriving at our final solution.

Input and output layers:

The input and output layers of an artificial neural network (ANN) are two important components that play a critical role in the success of the model [3, 29, 30]. The initial dataset is accepted by the input layer and it processes this information using its various hidden layers. This processed information is then sent to the output layer for generating a result. Firstly, before being fed into the ANN, data must be pre-processed and standardized. Then every feature from this dataset will have its own neuron in order to feed these features into each respective neuron within the input layer. Following this one or more hidden layers do calculations with respect to these features before sending them onto the output layer which handles producing a final outcome decision or prediction of some sort. In classification tasks such as binary classification or multiclass classification, there needs to be an appropriate activation function used within this output layer for normalizing values produced by all its neurons along with varying degrees between 0 and 1 since other combinations lead to not ideal results due to instability factors [3]. In comparison for regression tasks, you simply want as many outputs that match your target's continuous variables whereupon also have similar characteristics but require less effort when utilizing different activation functions making sigmoid generally easier than softmax.

Activation by Forward Propagation:

The process of training artificial neural networks involves forward propagation. It follows a sequence of computational steps based on tensor multiplication and activation, starting at the input layer and periodic activations in between. In each neuron or node within the layer, incoming values are multiplied by their specific weight before being summed together. This data then flows through an activation function to activate the next layer and begin again with tensor multiplication operations. When constructing a deep learning model it is important to keep track of the total number of weights that need to be tuned

(these are called trainable parameters). Generally speaking, as more layers are added when creating an ANN, the complexity increases exponentially thereby resulting in significantly higher numbers of tunable parameters which have the potential for overfitting if not inadequately managed. To prevent machine learning models from memorizing all training samples rather than discovering certain patterns, regularisation methods for instance, the L1(lasso regression) and L2 (Ridge regression) need to be combined with backpropagation into a cost function so these weights can be accurately adjusted then minimizing the error rate during the inferencing stage.

Network Capacity

An Artificial Neural Network (ANN) is capable of learning and representing complex models. As the number of trainable parameters in a model increases, however, so too does the capability for it to overfit data that is presented during training. To clarify, with more trainable parameters incorporated into a model, there will be higher chances of obtaining higher accuracy when trained on existing datasets- because it might attempt to remember even anomalies or irregularities found within those datasets - but this could result in poorer generalization performance in unseen scenarios. In order to counter this possibility from becoming a reality, regularization techniques such as L1 and L2 can be added onto the loss function for penalizing larger weights; thus decreasing overfitting effects as a result maintaining satisfactory levels of accuracy and also improving model flexibility when exposed to new data points in future inferences.

2.1.3 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are an advanced type of artificial neural network designed to process sequences of input data. To do this, RNNs maintain a "memory" using internal states calculated from each time step, containing information extracted by the network about prior inputs during the same sequence. This is accomplished through connections between neurons in higher and lower layers that form a circular shape as well as optional self-feedback connections which enable these networks to pass data to later stages of processing based on earlier parts of the sequence. These recurrent linkage structures give RNNs their unique ability to retain experience and knowledge within their memory while they handle current inputs, allowing them to outperform other traditional methods in processing lengthy series of information.

The internal state of an RNN at each time step t may be expressed mathematically as a function of the previous internal state h(t-1) and the current input x(t), as seen below:

$$h(t) = f(h(t-1), x(t)) \tag{2.9}$$

Here, f is a non-linear function that maps the input and previous internal state to the current internal state. The output of the network at each time step can also be computed as a function of the current internal state, as shown below:

$$y(t) = g(h(t)) \tag{2.10}$$

At every time step, there's an output generated by applying a non-linear function g to the current internal state.

Recurrent Neural Networks (RNNs) are intended to maintain an internal state during each sequence classification stage when processing sequential input data, however, in order to train an RNN network, the basis lies in backpropagation through time (BPTT). BPTT adjusts the classic back-propagation technique to account for sequential inputs, as part of the training process an array of inputs is given to the network which is then compared to the desired output. By backpropagating the mistake, the network's weights get updated.

RNNs have a susceptibility towards vanishing gradients that occur when there is an extremely small value of gradient used in updating network weights leading to difficulty in learning long-term relationships, but many types of recurrent neural networks have been developed including Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs).

Gated Recurrent Units (GRUs) and the level of connectivity in RNN design can vary from part to fully linked. There are two commonly used RNN architectures called Elman and Jordan networks that have been presented in the literature, and Elman Networks and 3 layered Neural Networks maintain comparable structures except for context cells storing outputs from the buried layer. The corresponding hidden neurons receive both the outputs from every context cell as well as an initial signal, and the network can store previously learned information and apply that knowledge when dealing with new inputs.

In order to describe mathematically how an Elman network operates at any given time step one must consider factors for instance, current input $x(t)$, the prior internal state $h(t-1)$, and the output of the related context cell $c(t-1)$, as seen below:

$$h(t) = f(Vx(t) + Wh(t-1) + Rc(t-1) + b) \tag{2.11}$$

$$c(t) = g(Uh(t) + d) \tag{2.12}$$

Here, f and g are non-linear activation functions, and V, W, R, U are weight matrices, and b and d are bias terms.

Training of Elman networks with additional input from context cell's output can be achieved using standard error backpropagation, while being trained a range of inputs are fed into the system followed by analyzing those outputs against their expected results. Modifying the network's weights involves backpropagating the mistake over time.

The combined result of each context cell's output makes its way back to its relevant hidden neurons,

in addition to its fundamental 3-layer neural network-like architecture, Elman nets also contain extra context cells which keep track of output signals in its hidden layers. This leads to the ability of the system to retain historical information on input and use it for effective guidance during future processing the contrary to Elman Networks which do not use their own output as input, Jordan Networks do. For every timestep t in an operating Jordan Network, one can express its internal state by using variables such as current input $x(t)$, prior internal state $h(t-1)$, and the network output $y(t-1)$.

$$h(t) = f(W_x(t) + U_h(t-1) + V_c(t-1) + b) \tag{2.13}$$

$$y(t) = g(W_c(h(t)) + b_y) \tag{2.14}$$

Conventional error backpropagation applied with extra inputs derived from the network's output can train both Elman and Jordan networks, but when updating network weights with a very small gradient in RNNs, it might become difficult to learn long-term relationships which is known as the vanishing gradient issue. There are two types of recurrent neural network models designed to solve this problem: Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs)[31].

Training Recurrent Neural Networks

The presence of feedback connections in Recurrent Neural Networks (RNNs) can pose difficulty while training due to the issue of vanishing or growing gradient, but the principal techniques employed for the training of RNNs include BPTT and Real-Time Recurrent Learning (RTRL).

Training RNNs usually involves using BPTT that exploits the characteristic of having identical feed-forward neural networks during a certain period, so a feed-forward neural network can be obtained through temporal unrolling of the RNN. Conventional backpropagation technique can be used for training the unfolded network. During training inputs are presented to the network which produces outputs that must be matched to their intended output, and backpropagation over time is used to correct errors and adjust network weights. The weight updates for this recurrent network involve summing up their individual deltas at each time step and Backpropagation proceeds by taking discrete iterative steps forward from an initial state up to its final destination [32]. Here is how the weight updates within BPTT can be written using mathematical notation:

$$\Delta w(t) = \frac{-\eta \partial E}{\partial w(t)} \tag{2.15}$$

where $\partial w(t)$ is the weight update at time step t, η is the learning rate, E is the error, and $\frac{\partial E}{\partial w(t)}$ is the derivative of the error with respect to the weight at time step t.

In essence, you can view the unfolded Neural Network similarly to a Feedforward Neural Network with different layers corresponding to each individual timestep, and identical weights used in both network types (feed-forward and RNN) result in similar behaviour.

A well-established mathematical foundation supports the extensive use of BPTT reported in the literature and effective training of RNNs is possible through the use of BPTT which has a proven track record in diverse applications like speech recognition.Real-time recurrent learning: uncovering its computational and mathematical foundations through research theory, which is the algorithm used to train RNNs in real-time and with minimal error propagation requirements, works by computing the gradient essential for weight updates as inputs are given.As such there is no need for a standalone training time period which makes it an improved algorithm that is quicker than Backpropagation Through Time (BPTT), but the computational expenses involved per update cycle combined with a high level of memory usage in storing non-local data make RTRL more burdensome. Real-Time Recurrent Learning (RTRL) will be given a thorough theoretical and computational foundation in this paper[32]. .

The training objective of RTRL is to minimize the overall network error given by Equation 1:

$$E(\tau) = \frac{1}{2} \sum_{k \in U} (d_k(\tau) - y_k(\tau))^2 \tag{2.16}$$

Where is the label at every time τ for every non-input unit k, and is the output of unit k at time τ. The gradient of the total error is the sum of the gradient for all previous time steps and the current time step, given by Equation 2.

$$\nabla_\theta L = \sum_{t=1}^{T} \frac{\partial L_t}{\partial h_t} \qquad \frac{\partial h_t}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial h_t} \left(\frac{\partial h_t}{\partial \theta} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial \theta} \right) \tag{2.17}$$

To compute the weight changes, RTRL requires calculating the sensitivity of the output of each unit to a small change in the weight. The sensitivity is given by

$$p_{uv}^k(\tau) = \frac{\partial_{yk}(\tau)}{\partial W_{[u,v]}} \tag{2.18}$$

In RTRL, the gradient information is forward-propagated. The output $yk(t+1)$ at time step $t+1$ is given by Equation

$$yk(t+1) = fk(zk(t+1)) \tag{2.19}$$

where fk is the activation function and $zk(t+1)$ is the weighted input. Differentiating Equations 2.17, 2.18, and 2.18, we can calculate the sensitivity for all time steps $\geq t+1$ using Equation 2.18:

$$p_{uv}^k(\tau) = f_k'(z_k(t+1) \left[\delta_{u,k} X_{u,v}(t+1) + \sum_{l \in U} W_{[k,l]} p_{uv}^l(t) \right]. \tag{2.20}$$

where δuk is the Kronecker delta.

Knowing the initial value of p

$$p_{uv}^{k}(\tau) = \frac{\partial_{yk}(\tau)}{\partial W_{[u,v]}} \tag{2.21}$$

we can recursively calculate the values for all subsequent time steps using Equation 2. 18. Finally, the negative error gradient

$$\nabla W E(\tau) \tag{2.22}$$

can be

calculated using Equation 2.21, and the final weight change for $W[u,v]$ can be calculated using Equations 2.21 and 2.20.

Vanishing error problem

The Vanishing Error Problem is a well-noted issue when training Recurrent Neural Networks (RNNs). RNNs are designed to process sequence data and are utilized for tasks such as speech recognition, language modeling, and time series predictions. The problem occurs as the input sequence grows longer RNNs may experience either vanishing or exponential increases in error gradients with each respective timestep [33]. To fully understand the issue mathematically we will delve into error propagation through the network during training. Weights of the neural networks can be adjusted using the Backpropagation Algorithm which efficiently calculates errors from output layer neurons down through successive layers decreasingly leading back to the initial inputs of these units. However, when dealing with long sequences certain dependencies between older timesteps become more difficult to update due to previous signals being too diminished or noisy by earlier propagated errors making it impossible to update weights appropriately creating issues that affect an effective network learning model. The weight update formula is given by:

$$\Delta W[u,v] = -\eta \frac{\partial E_{\text{total}}(t_0,t)}{\partial W[u,v]} \tag{2.23}$$

where $\Delta W[u,v]$ is the change in weight between neurons u and v, η is the learning rate, $E_{\text{total}}(t_0,t)$ is the total error of the network from time t_0 to time t, and $\frac{\partial E_{\text{total}}(t_0,t)}{\partial W[u,v]}$ is the partial derivative of the error with respect to the weight between neurons u and v.

The backpropagated error signal at time τ (with $t_0 \leq \tau < t$) of the unit u is given by:

$$\theta_u(\tau) = f'_u(z_u(\tau))^* \sum_{v \in U} W[v,u]^* \theta_v(\tau+1) \tag{2.24}$$

where $f'_u(z_u(\tau))$ is the derivative of the activation function of neuron u, $z_u(\tau)$ is the weighted sum of the inputs to neuron u at time τ, U is the set of non-input neurons, $W[v,u]$ is the weight between neurons v and u, and $\theta_v(\tau+1)$ is the backpropagated error signal from neuron v at time $\tau+1$.

As the error signal propagates through the network, it can either vanish or explode. The problem occurs when the error signal vanishes, which makes the learning process slow or impossible. The error signal vanishes if the product of the weights and derivatives of the activation functions is less than one for all time steps[34]:

$$|f'_u(z_u(\tau)) \cdot W[v,u]| < 1, \forall \tau \tag{2.25}$$

To address the vanishing error, it is possible to employ a few different techniques. One of the most reliable strategies is to incorporate Long Short-Term Memory networks into the structure. LSTM networks typically include different types of gates which play an essential role in managing information for various objectives [34]. For instance, through input and output gates, these models are able to store relevant information over longer time intervals without facing difficulties from errors that tend to vanish quickly during this process. Additionally, forget gates can be used to take charge of deleting any data or learning experience which may no longer be needed by intelligent systems within LSTM networks. In order to provide an additional approach towards adjusting outputs so as not to pick up on minor changes or fluctuations upon future inputs in recurrent neural network architectures when controlling the Vanishing Error phenomenon, residual connections could also come into place with successful results [35]. As compared with non-residual models where the system reverts back directly after receiving new parameters via its activation function layers before passing further onto other layers (which has been linked with adding some difficulties). Residual connection within these particular settings allows one's incoming data across such synaptic components conducive for gradient accumulation flow capabilities while enabling useful patterns contained in certain inputs prior to recalibration processes automatically if added accurately and appropriately.

Gradient clipping is a technique used to help regulate gradients during training, and ultimately counteract the vanishing error problem. This approach involves setting an upper limit for gradient size or magnitude, as too high and too low of gradients can both be problematic. If the threshold for the size of the gradient has been exceeded, then it will be scaled down [34]. This helps to avoid having extremely small gradients that would vanish altogether helping with potential problems caused by vanishing errors.

2.1.4 LSTM

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) developed to tackle the issues with traditional RNNs in processing and modeling sequential data. Originally introduced by [36].these networks can learn dependencies over extended periods of time, as well as combat vanishing gradient problems which tend to arise during training processes involving very long sequences. This has led LSTMs to become popular for tasks such as natural language processing, speech recognition, and time-series prediction among many other uses [37, 38].

LSTM Architecture

Long-Short Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) that contain memory cells that manage information over time [36]. These memory cells are composed of three elements: the input gate, forget gate, and output gate which control the flow of data within the network [39]. The input gate is used to decide how much information from the current input should be stored in the cell's memory [34], and this control is derived through mathematical equations. The forget gate controls what facts remain relevant in future cases and those upon which previous calculations must still hold true. Lastly, the output gates determine when an iteration's results will be transferred forward to subsequent iterations as well as what amount of these results should be passed on. By regulating both, retrieval and forgetting capabilities within a recurrent neural network structure, Long-Short Term Memory (LSTM) has successfully addressed problems associated with traditional RNNs such as vanishing gradients or difficulty learning long-term dependencies[33, 40]. Equations govern the input gate:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \check{c}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \end{aligned} \tag{2.26}$$

where i_t represents the input gate, $\check{c}_t$ represents the candidate cell state, σ is the *sigmoid* activation function, and *tanh* is the hyperbolic tangent activation function.

Forget Gate: This gate decides which information should be discarded from the memory cell [41] . The forget gate is defined by the equation:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.27}$$

$$\hat{c}t = \tanh(W_c \cdot [ht-1, x_t] + b_c) \tag{2.28}$$

where i_t represents the input gate, $\hat{c}_t$ represents the candidate cell state, σ is the sigmoid activation function, and *tanh* is the hyperbolic tangent activation function.

Forget Gate: This gate decides which information should be discarded from the memory cell [41] . The forget gate is defined by the equation:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.29}$$

where f_t represents the forget gate.

Output Gate: This gate controls the output generated by the memory cell, which is based on the cell state and the input [41] . The output gate is given by the following equations:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.30}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \hat{c}_t \tag{2.31}$$

$$h_t = o_t \cdot \tanh(c_t) \tag{2.32}$$

where o_t represents the output gate, c_t represents the cell state, and h_t represents the hidden state.

The LSTM cell's mathematical formulation can be broken down into four primary equations, encompassing input, output, and forget gate activations, as well as cell state, updates [41] :

Input Gate

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.33}$$

Forget Gate

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.34}$$

Output Gate

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.35}$$

Cell State Update

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{2.36}$$

Hidden State Update

$$h_t = o_t \odot \tanh(c_t) \tag{2.37}$$

where σ denotes the sigmoid activation function, $\circ$ denotes element-wise multiplication, W denotes the weight matrices, b denotes the bias vectors, and x_t, h_t, and c_t represent the input, hidden state, and cell state at time step t, respectively [39].

Variants and Extensions of LSTM Networks

Several modifications and expansions of the original Long Short-Term Memory (LSTM) architecture have been suggested in order to increase its performance and usability. Examples of these include peephole links [42], bidirectional LSTMs [43] and attention mechanisms [44]. The important characteristic of an LSTM is its continuous error carousel (CEC). The CEC enables long-term data storage without input, as constant backflow finds its way through the system [36]. However, when

instated within a neural network framework, clashes can arise due to this process as not only is conduction within the cell itself occurring but also between cells in the network. Incoming connections lead to updates whilst information must be stored simultaneously resulting in inconsistencies regarding weight changes for neurons. Output connectivity from neurons is also convoluted; weights making up both retrieval functions for content flow and limitations for other neuron outputs can clash with each other left unchecked.

In order to solve the challenge of conflicting weight updates, LSTM extends the CEC with input and output gates connected to the network input layer and to other memory cells. This results in a more complex LSTM unit called a memory block [36]. The input gates in a Constant Error Carousel (CEC) network are sigmoid threshold units with an activation function range of [0, 1]. This allows these gates to control the signals from the network to the memory cell effectively by scaling them accordingly. When the gate is closed, its activation is close to zero and hence does not allow access. Also, this enables us to learn how to protect against irrelevant signals disturbing what has been stored in u. The output gate works similarly as it can control and limit who or what will gain access to the memory cell contents thus making sure that other memory cells remain undisturbed by any disturbance originating from u. Additionally, multiplicative gate units provide another level of protection because they are able to determine whether error flow should be allowed or disallowed [36].

Recent advances in artificial neural networks have introduced designs that enable them to process sequential data, such as time series and natural language. One of these designs is Recurrent Neural Networks (RNNs). In spite of their potential, RNNs have difficulty because of the vanishing gradient issue which makes it hard for them to learn long-term dependencies. In order to address this limitation Long-Short Term Memory networks were developed and integrated into the RNN structure. The goal of this work is twofold: firstly, providing a theoretical understanding of how LSTM-RNNs propagate information forward and backward, especially through the mechanism of forget gates; secondly, analyzing the merits and limits of LSTM-RNNs comparatively with other machine learning techniques.

The Forward Pass

The forward pass of a Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) involves propagating the information from each unit in the network to subsequent units. Specifically,M specifies a set of memory blocks, where m is denoted as the c-th memory cell within that memory block [31].

The internal state of each memory cell at time $\tau+1$, denoted by $s_{m_c}(\tau+1)$, is calculated according to both its state at time τ, represented by $s_{m_c}(\tau)$, and an associated weighted input term, noted as $z_{m_c}(\tau+1)$. This weighted input term is multiplied by the activation value for one of two gates—the output gate or input gate —associated with each specific memory block. For example, for any given generic memory block m, labeled output gate and input gate can be denoted as respectively, where W represents some weight connecting node u and node v.After determining this multiplicative outcome between weighted inputs and either output/input gates correspondingly, it forms activation values which are used to calculate final activation value(for specific cell): $y_{m_c}(\tau+1)$ all while updating internal state

s accordingly [31].

The activation y_{in_m} of the input gate in_m is computed as follows:

$$y_{in_m}(\tau+1) = f_{in_m}(z_{in_m}(\tau+1)) \tag{2.38}$$

where f_{in_m} is the activation function of the input gate and z_{in_m} is the weighted sum of the inputs to the input gate. The activation of the output gate $z_{out_m}(\tau+1)$ is computed as follows:

$$z_{outm}(\tau+1) = f_{outm}(z_{outm}(\tau)W_{outm,ymc(\tau+1)}) \tag{2.39}$$

where f_{outm} is the activation function of the output gate and $W_{outm,ymc(\tau+1)}$ is the weight connecting the output gate to the activation of the cell [31].

The activation of the cell $ymc(\tau+1)$ is computed as follows:

$$ymc(\tau+1) = h(smc(\tau+1)) \tag{2.40}$$

The activation function of the cell in this expression is denoted by h and typically takes either a hyperbolic tangent or a sigmoid form. The output of this process, or the updated internal state of the cell $smc(\tau+1)$, is calculated by first multiplying together weighted input $zmc(\tau+1)$ with the activation of its input gate $yinm(\tau+1)$. This product will then be added to that which results from multiplying together the previous state $smc(\tau)$ with its corresponding forget gate's activation value $yfm(\tau+1)$ [31].

The activation of the forget gate $yfm(\tau+1)$ is computed as follows:

$$yfm(\tau+1) = f_{fm}(z_{fm}(\tau+1)) \tag{2.41}$$

where f_{fm} is the activation function of the forget gate and $z_{fm}(\tau+1)$ is the weighted sum of the inputs to the forget gate. The forget gate determines how much of the previous state should be retained and how much should be forgotten, based on the current input and previous state [31].

Forget Gates

The forget gate in Long-short Term Memory Recurrent Neural Networks (LSTM-RNNs) is responsible for effectively managing long-term memories which go beyond single timesteps. The purpose of this gate is to balance the importance of short and long-term memory when making predictions. It selectively adjusts, discards, or retains information based on its relevance to the task at hand through a sigmoid activation function that results from weighted sums of inputs. This allows it to recognize patterns that are important for future prediction while discarding those deemed irrelevant allowing for better accumulation and retrieval of memories within these RNNs [31].

Backward Pass

In the process of backpropagation, an LSTM-RNN calculates error signals for each weight. The purpose of these calculations is to adjust the weights accordingly such that there is a decrease in the amount of difference between actual output and desired output. To achieve this, calculus's chain rule allows mistakes to be transferred backward through layers in order to determine which adjustments should be made to specific weights. These changes are done with a view toward minimizing any discrepancy between what was being expected as compared with what had been delivered by the RNN network. Updates to weights are made by propagating error signals back across the network. The objective is to reduce the deviation between actual and desired output. As a consequence of the chain rule of calculus, mistakes may spread in reverse across a network. By propagating error signals back across the structure, new values for weights can be formulated which will help bring about precise results.

Let $E(\tau)$ be the error at time step τ, and let $y(\tau)$ be the output of the network at time step τ. The error signal for the output gate $outm$ at time step τ is computed as follows:

$$\delta_{outm}(\tau) = E(\tau) * h'(ymc(\tau+1)) * z_{outm}(\tau+1) \tag{2.42}$$

where h' is the derivative of the activation function of the cell, and $z_{outm}(\tau+1)$ is the weighted sum of the inputs to the output gate.

The error signal for the activation of the cell $ymc(\tau+1)$ at time step τ is computed as follows:

$$\delta_{ymc}(\tau+1) = (\delta_{outm}(\tau) W_{outm,ymc(\tau+1)}) * h'(ymc(\tau+1)) \tag{2.43}$$

where $W_{outm,ymc(\tau+1)}$ is the weight connecting the output gate to the activation of the cell.

The error signal for the forget gate fm at time step τ is computed as follows:

$$\delta_{fm}(\tau) = (\delta_{ymc}(\tau+1) * smc(\tau)) * f'(z_{fm}(\tau+1)) \tag{2.44}$$

where f' is the derivative of the activation function of the forget gate, and $smc(\tau)$ is the previous state of the cell.

The error signal for the input gate inm at time step τ is computed as follows:

$$\delta_{inm}(\tau+1) = (\delta_{ymc}(\tau+1) * z_{mc}(\tau+1)) * f'(z_{inm}(\tau+1)) \tag{2.45}$$

where $z_{mc}(\tau+1)$ is the weighted input to the cell, and f' is the derivative of the activation function of the input gate.

The error signal for the weighted input $z_{mc}(\tau+1)$ at time step τ is computed as follows:

$$\delta_{zmc}(\tau+1) = (\delta_{ymc}(\tau+1) * y_{inm}(\tau+1)) * h'(s_{mc}(\tau+1)) \tag{2.46}$$

where $y_{inm}(\tau+1)$ is the activation of the input gate, and h' is the derivative of the activation function of the cell.

The error signal for the previous state $s_{mc}(\tau)$ at time step τ is computed as follows:

$$\delta_{smc}(\tau) = \delta_{ymc}(\tau+1) * y_{fm}(\tau+1) \tag{2.47}$$

where $y_{fm}(\tau+1)$ is the activation of the forget gate. An optimization technique, such as gradient descent or Adam's algorithm, may then be used to adjust the network's weights based on the calculated error signals.

Complexity

Long Short-term Memory Recurrent Neural Networks (LSTM-RNNs) are more complex than traditional RNNs due to the additional parameters and calculations they have integrated. This extra complexity allows them to save long-term dependencies which can be used later; however, it also makes these networks much harder to train or understand correctly. The computational complexity for an LSTM-RNN is denoted by $O(N^2)$ where N is the number of memory cells. This indicates that both the memory needs as well as time requirements will go up exponentially if you increase the number of memory cells found inside an LSTM-RNN.

Strengths and Limitations of LSTM-RNN

Recurrent Neural Networks (RNNs) are state-of-the-art algorithms that can be applied to a range of problems concerning sequenced data. Long Short-Term Memory (LSTM) based RNNs are even more capable models with the ability to remember information over long sequences. LSTM networks use gates in order to control how much new and old information is retained and used, which allows them to better understand long-term dependencies within the data. However, although this improved memory capacity has its advantages when dealing with complex tasks such as Natural Language Processing (NLP) and Time Series Prediction, there can still be issues stemming from what is known as Vanishing Gradient' during training where either too little or too much gradient descent takes place respectively, making it difficult for optimization to take place correctly. In addition, due to their complexity, these types of RNNs can be more challenging and computationally expensive to train and than traditional RNNs. In addition, they typically need large datasets in order for successful learning outcomes; therefore computational resources may present an additional barrier particularly if attempting this on big networks or datasets.

2.1.5 Optimization

The optimization of training neural networks is essential in order to ensure that the weights are suitable for producing the desired output and reducing any deviations from it. This research examines various techniques of optimization, such as gradient descent, backpropagation of errors, convergence procedures, fine-tuning pre-trained networks as well as evaluation metrics devoted to judging a neural network's performance. To further understand these concepts we exemplify them through a computational study on a set of handwritten digits.

2.2 Optimizer

An optimizer is an algorithm used to adjust the weights of a neural network during training to minimize the difference between the predicted output and the target output. There are various optimization algorithms used in training neural networks, including stochastic gradient descent, Adam optimizer, and Adagrad optimizer [45].

2.2.1 Gradient Descent Optimization

In essence, gradient descent helps us to find the optimal set of weights for a neural network by iteratively adjusting these weights based on the estimated direction and magnitude of the error. This is done using the chain rule of calculus which allows errors in one layer to be propagated through the entire network [3]. To be more specific - each step taken involves recalculating the cost function (which measures how closely our predicted output matches up with our target output), then computing its derivative with respect to all trainable weights within the network. Finally, we adjust those weights according affected by this partial derivative; moving them in negative or positive directions depending on what it says. Doing this repeatedly, while adjusting the learning rate as needed will eventually lead us to find a point where no further improvement can be made suggesting that we have discovered an accurate model which has been tuned correctly [46].

The update rule for gradient descent is as follows:

$$w = w - \alpha \nabla_w J(w) \tag{2.48}$$

where w indicates the learning rate and controls how quickly each weight should change in order to reduce losses over time. whereas $J(w)$ is the cost function, and $\nabla_w J(w)$ is the gradient of the cost function with respect to the weights of the network. There are different variations on this basic structure including batch gradient descent, stochastic gradient descent, and mini-batch gradient descent [45]. In each variant's implementation, given a differentiable function $f(x)$, an initial variable x is selected which will later be adjusted using calculated gradients of that same function such that it converges towards its global optimum value after sufficient iterations have taken place.

The update rule is as follows:

$$x_{t+1} = x_t - \eta \nabla f(x_t) \tag{2.49}$$

where x_t is the current value of x, η is the learning rate, and $\nabla f(x_t)$ is the gradient of $f(x)$ evaluated at x_t. The learning rate is a hyperparameter that controls the step size of the updates [45].

Convergence Properties

The ability of gradient descent to minimize a function depends on various factors, like the size of the learning rate, which is used when updating the weights, and properties of the minimized loss surface [47]. Under suitable conditions such as convexity and Lipschitz continuity of the derivative–gradient descent is guaranteed to reach a global minimum in its convergence path.

Performance Factors: Learning Rate: The size of the learning rate defines how quickly a model will learn from one iteration to another. If it's too small, training may be slow; but if it's too large, there could be overshooting or divergence away from an optimal solution [45].

Initialization: Initializing variables with appropriate values prior to training can help ensure faster convergence (or prevent getting stuck at local minima) [48].

Adaptive Learning Rates: Different adaptive strategies such as AdaGrad [49] and Adam [50] have been proposed which modify learning rates at each update step depending on their history since initialization.

2.2.2 Back Propagation of Error

Backpropagation of error is a key component of gradient descent, as it allows the error to be propagated backward through the network to update the weights. The backpropagation of error involves computing the gradient of the cost function with respect to the weights of the network using the chain rule of calculus [3].

The update rule for the weights of the network using backpropagation of error is as follows:

$$\Delta w = -\alpha \nabla_w J(w) \tag{2.50}$$

where Δw is the change in the weights of the network, α is the learning rate, $J(w)$ is the cost function, and $\nabla_w J(w)$ is the gradient of the cost function with respect to the weights of the network.

2.2.3 Convergence

Convergence is an essential component while training neural networks. It guarantees that the network can recognize the distinctive patterns in the data, and apply its learning to new situations. Its progress can be observed by keeping a tab on two key statistics: the training loss and validation loss over time. The training loss indicates how much error lies in the data used for training purposes; whereas, the validation loss observes how right or wrong predictions are with respect to an independent hold-out dataset [3].

Dropout

Early stopping is a powerful algorithmic method for preventing overfitting. In deep neural networks, this is done by monitoring the validation loss in order to determine when to stop further training of the network. This technique can improve the model's ability to generalize new data rather than simply memorizing the existing training set[51]. Dropout techniques are also commonly used in deep learning due to their effectiveness in optimizing and controlling a network's capacity. Dropouts help minimize overfitting caused by too much complexity in modeling, as well as improvement of performance on novel data sets. They can greatly reduce complexity while still maintaining accuracy they essentially remove neurons randomly with each training step which forces them to focus on more generalized outputs.

Dropout layers are a regularization technique used in deep learning neural networks that act to reduce overfitting, by dropping some of the activations within higher-level layers.

During training, these units are randomly selected and ignored, forcing the network to create multiple redundant pathways for information. This helps prevent the network from becoming too finely attuned only to the observation data during training, instead allowing it to zero in on more general features and thus be better prepared when exposed to new data sets during evaluation or tests. In large networks with significant capacity levels where overfitting is at risk of occurring, dropout layers can provide an additional safeguard by fine-tuning how much redundancy there is; this way potential extra resources can be harnessed while maintaining robustness and accuracy.

Decay

Decay is a technique used to adjust the learning rate during training which can help accelerate model convergence. Training with a higher learning rate allows for quicker bypassing of local minima and faster attainment of the global minimum, although this usually has lower precision in weight updates. After reaching what may be thought as the optimal point or 'global minimum', smaller learning rates are then set so that finer detailed changes can take place within (referencing Bengio et al., 2012). Adjusting decay rates is crucial; using too large of values would reduce weights too quickly halting progress before it has been completed optimally and if set too small, would result in rapid converging yet need extensive fine-tuning afterward [52].

Momentum

Momentum, a classical technique that has been studied since [53], is used in gradient descent to accumulate velocity vectors in the direction which consistently reduces the objective across training iterations. The aim of this technique is to accelerate gradient descent by altering weight updates with the computed velocity vector so as to decrease the chances of converging on local minima. Despite their advantages, momentum-based methods have one major disadvantage they may not reach optimal convergence with respect to global minima and jump over it due to their tendency of evading local minima points. To address this issue Nesterov Momentum was introduced which can predict upcoming weight updates using the prior iteration's gradient hence resulting in smoother convergence towards target minimum [54]. This algorithm adds only a partial first momentum vector before adding a full

vector when applying weight update thereby allowing for better approximations toward global minimum compared to conventional momentum methodologies.

Optimizers are an important part of Deep Neural Networks (DNNs). Training these networks is usually harder than training shallow learning algorithms, due to the vast number of parameters needing optimization. This can make the cost function rough and cause descent optimization algorithms to converge slowly or not at all. To counter this issue, new methods have been explored which build upon causes set out by basic principles [45]. Aside from convergence, dropout, decay, and momentum techniques have also been studied as well as adaptive learning rate algorithms and multiple optimizers that deal with issues faced when dealing with deep neural networks [55].

AdaGrad is an adaptive learning rate algorithm that adjusts the learning rate for each parameter individually, according to a calculation of its historical gradients. This technique facilitates automated tuning of the learning rates and has applications in processing sparse data effectively. However, it has been shown that. AdaGrad can suffer from rapid learning rate decay and thus impede optimization, as it is difficult to reach the global minimum before stalling due to very low values of said parameter. To confront this challenge Root Mean Squared Propagation (RMSProp) [56] was developed as an alternative adaptation algorithm for optimizing both neural networks and other machine-learning settings. In contrast with AdaGrad's accumulation of gradients over time, RMSProp introduces the concept of exponential moving average an estimation based on more recent information which yields a much more stable learning rate that leads towards effective convergence of computational systems.

Adam [50] is a breakthrough optimization technique that brings together the advantages of AdaGrad and RMSProp. The algorithm computes changing learning rates for each parameter by keeping an estimate of gradients' first and second moments. Its robustness to varying hyperparameters and its good performance on multiple tasks have made it popular among practitioners. Regularization strategies such as L1 and L2 regularization [57], which penalize large weight values, also act to reduce overfitting in DNNs. Batch normalization [58] has been found to be highly beneficial in stabilizing deep Neural Network (DNN) training; by standardizing the layer inputs, batch normalization decreases internal covariate shift resulting in quicker convergence that achieves better generalization compared with models without this technique.

Optimizer

Deep Neural Networks have a lot of parameters that must be optimized, which is more tricky and more difficult than standard shallow learning algorithms. The cost function could also prove to be challenging as the gradient descent optimization may become stuck on local minimums, making it to converges slowly or not at all. This issue can partly be resolved by Stochastic Gradient Descent (SGD), an optimization algorithm where approximations of cost from single training samples are done on small subsets with mini-batch learning which helps make faster convergence. The stochasticity introduced through these estimations utilized in SGD will potentially help jump out of any unconstructive or undesirable local minima [47].

Adaptive Gradient (Adagrad) is an optimization technique that alters the learning rate of individual

weights depending upon how frequently they are altered during training. This enables the optimizer to update more relevant parameters faster and allows for quick convergence of less significant parameters that may not need as much attention. In order to track which weights have been used, Adagrad keeps a dictionary or memory of all parameter updates made thus far. The updation frequency is tracked by means of this gathered information and then accordingly corrections are made to the learning rate[55]. As such, it tends to increase the learning rate for infrequently used weights so that an optimal convergence towards model fit can be achieved, whereas heavily updated ones would have their rates decreased in order to prevent excessive flows in out-of-range values.

Root Mean Square Propagation is an algorithm that uses the gradients of the weights to determine how much each weight should be updated. It works by maintaining a record of the average gradient for each individual weight, over time. When updating these weights in subsequent iterations, this average is taken into account when dividing the current iteration's gradient from it. Due to this each update is to be proportional to its respective positive/negative change with respect to its mean [56]. On the other hand, Adaptive Moment Estimation (Adam) and Nesterov Adaptive Moment Estimation (Nadam) algorithms track changes in gradients accordingly as well but do so using a velocity vector that records momentum values associated with each respective gradient over a fixed period of time. Adam was also one of the first optimizers to incorporate momentum acceleration into traditional Gradient Descent [59].

Selecting an optimizer for a Deep Neural Network (DNN) is not a straightforward process. Since the data used in machine learning tasks may often be quite large and complex, it's best to find out which optimizer works better by trying different approaches instead of relying strictly on inferences. DNNs have many parameters due to which the cost function has multiple local minimums instead of one global one. This means that finding the global minimum is not vital; instead, all that's needed is finding a local minimum sufficient enough to meet the model's performance requirements [3].

It is also important to consider which optimizer to use for the neural network. Different optimizers can have substantially different effects on the training speed and performance of the model, with some converging quickly but reaching relatively low-performing solutions, while others might take longer to converge but reach a more optimal minimum result. Therefore, it is important that multiple optimization algorithms be tested in order to determine the best choice available. Furthermore, certain optimizers may require extra hyperparameter tuning such as adjusting SGD's learning rate or momentum parameters or RMSProp and Adam's decay rate and epsilon values respectively. At times this process can prove long and computationally costly thus finding ways of reducing resources while still optimizing efficiency should be considered when possible.

Hyperparameters

The functioning of Artificial Neural Networks (ANNs) is determined by their architecture. This means, among other things, that the structure of the network remains in a fixed state even when the learning process optimizes parameters for a certain task [3]. The elements which have to be chosen by the user are known as hyperparameters and they comprise, in relation to model characteristics such as a number of units in a dense layer, filter size, padding, etc., information related to optimization algorithms' time

frame amongst others [60].

Choosing the right parameters is hugely important, as how they are set can strongly influence the results of the model. To identify which values work best it is necessary to use methods such as grid search, random search, or Bayesian optimization. But this process of finding accurate hyperparameters is not a straightforward task, the precise combination which works best varies depending on the specific problem and the dataset at hand [52].

Finalizing the choice of hyperparameters is an essential step in the design of a model since its values have a direct impact on how well it will learn, generalize and perform with unseen data [3]. it is important taking time to explore and find out which values are most suitable for one's project as that would make sure that we get the greatest performance from our model.

2.2.4 Metrics

Measuring performance is an important aspect in the area of neural networks. Many times, this is conducted with evaluation metrics like accuracy, precision, recall, and F1-score. Accuracy calculates the number of correct classifications made, precision provides the amount of correct positives within true positives and recall reveals how many positive points were recognized among all positive events. F1-score is the harmonic mean of both precision and recall combined.

Neural networks can be utilized in a variety of way to determine their effectiveness. Various metrics may be employed to gauge the performance of different models or tweak hyperparameters to optimize your system's results. Accuracy, precision, recall, and F1-score are some of the common metrics used when assessing classification tasks. The accuracy is quite straightforward since it only gauges the ratio of correctly identified examples compared with all samples. Precision involves measuring the portion of true positives among an entire set of positive predictions; similarly, recall determines what percentage of actual positives were predicted among all real values. This kind of evaluation is commonly seen in areas wherein low numbers for false positives and negatives are essential (like medicine). Lastly, the F1 score serves as an indicator that looks at how well a given model predicts and remembers data.

The evaluation of neural network models typically calls for the use of metrics the result of which is a quantitative assessment of their performance. Depending on the task, these metrics may need to be adapted in order to best capture the particular traits desired from a model. Furthermore, when evaluating models against unseen data sets, it is essential that the metric results remain consistent across various subsets and are not biased towards any single subset or distribution profile. Various task-wise metrics also get employed commonly within certain domains; one example being average precision (AP) and mean average precision (MAP) which generally evaluate object detection accuracy [61]. These account for both precision and recall ratios associated with identifying objects accurately.

A different type of task-specific metric is the BLEU (Bilingual Evaluation Understudy) score, which is commonly used in natural language processing tasks e.g. machine translation and text summarization. This score ascertains the level of concurrence between model-generated output and human-generated

output supported by n-gram matching [62].

Classfication

For assessing the performance characteristics of a classification model it is often necessary to use a confusion matrix and matrices are used to compare predicted with actual classes with results showing the values for true positives (TP) true negatives,(TN), false positives (FP) and false negatives (FN)[63]. Samples that are classified as correct use the true value while those that are misclassified use the false, indicating the actual classification of a given sample is done by using either positive or negative notations. These values provide a means of obtaining several different classification metrics [64].

	Predicted True	Predicted False
Actual True	True Positive (TP)	False Negative (FN)
Actual False	False Positive (FP)	True Negative (TN)

Table 2.1: Confusion Matrix

Accuracy serves as a common classification model metric, reflecting the proportion of correctly predicted samples. In binary classification tasks, binary accuracy is the sum of true positives and true negatives divided by the total number of samples [65]:

$$Accuracy\ binary = \frac{(TP + TN)}{total} \tag{2.51}$$

For k-class classification, categorical accuracy is the sum of true positives across all classes divided by the total number of samples:

$$\text{Accuracy categorical} = \frac{\sum(TP_i)}{\text{total}} \tag{2.52}$$

Nonetheless, accuracy metrics presume that all misclassifications are equally significant. In cases where false negatives and false positives carry different importance, recall, and precision are employed [65]:

$$Precision = \frac{TP}{(TP + FP)} \tag{2.53}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{2.54}$$

Precision measures the percentage of positive predictions that were accurate, while recall quantifies the percentage of positive class instances that were predicted correctly. The F1 metric consolidates these metrics, providing a single value to simplify model comparisons, unless solely optimizing model precision or recall [66]:

$$F1 = \frac{2 \times (Precision \times Recall)}{(Precision \times Recall)} \tag{2.55}$$

In addition, custom metrics, often combinations of TP, FP, FN, and TN, cater to more domain-specific

applications, such as maximizing value by assigning a value to each classification result [67].

2.3 Addressing Limited Labeled Data in Machine Learning: A Comprehensive Approach

In the field of machıne learnıng, certaın labeled data ıs requıred for accurate model traınnıng. When dealing with labelless datasets, a selection of approaches can be applied to this issue. Herein, we explain detaīledly the following techniques used when scarce labeled data is present: self-supervīsed learning, semi-supervised learning, weakly supervised leaming, and active learning [68].

1. Self-supervised learning: Self-supervised learning is an unmonitored way of producing labels from data, and then applying those labels to train a model. This has been found to be helpful in particular instances when labeled material is limited; examples include image classification, text identification, and voice recognition.

2. Semi-supervised learning: Exploiting both classified and unclassified information, semi-supervised learning attempts to construct a model that will generalize well with novel data. This method is usually foreseen in cases where it would be expensive or labor-demanding to label the data, such as deciding animal types by clocking a few samples.

3. Weakly supervised learning: makes use of partial labels which might not be wholly accurate but yet still gains knowledge thanks to assuming the intrinsic properties of the gathered data. Here again, this technique can come in handy when labeled examples are difficult or expensive to acquire.

4. Active learning: is an approach to training models made up of small, labeled datasets. By utilizing this method in areas such as image recognition and text classification, the desired outcome can be achieved by lessening the cost of labeling entities. Advanced deep learning systems like CNNs and LSTMs have been effectively used to support these techniques for use cases such as medical analysis, where early detection of conditions including schizophrenia can be enhanced or fraud identification making sure to pinpoint shady activities accurately.

When these techniques are integrated with advanced deep learning models, such as CNN, LSTM, and hybrid models. we can develop a strong methodology for handling limited amounts of labeled data. This approach can be applied to multiple domains including medical diagnosis allowing earlier detection and classifying conditions such as schizophrenia with greater accuracy or fraud detection pinpointing dubious activities that could have otherwise gone undetected.

Chapter 3

Materials and Methods

3.1 Materials

We are utilizing two different EEG datasets in our study. The first dataset is the EEG in Schizophrenia dataset [69], which includes 28 individuals (14 with paranoid schizophrenia and 14 healthy controls) whose data were collected using 250 Hz sampling frequency, 19 scalp channels including Fp1, Fp2, F7, F3, Fz, F4, F8, T3, C3, Cz, C4, T4, T5, P3, Pz, P4, T6, O1, and O2. The EEG recordings in this dataset were also obtained using the standard 10-20 EEG montage plus reference electrodes between Fz and Cz.

The second dataset we are using is the NMT dataset [70] which currently consists of an overall total of 2,417 EEG records with record lengths averaging out at around 15 minutes each. Similarly to the Schizophrenia dataset usage approach; here too we adopted the standard 10-20 system Figure 3.1, with 19 scalp channels and reference channels A1 and A2 on the auricles of the ear. The sampling rate for all channels was 200 Hz.

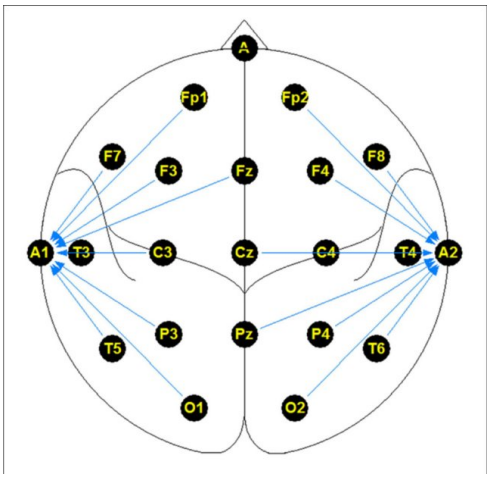

Figure 3.1: Diagram to show a standard electrode configuration in which the reference electrodes are placed on the left and right earlobes, forming a linked ear reference montage [70].

"The first study involved 14 patients (7 men and 7 women, with an average age of 27.9 ± 3.3 years) who were hospitalized for paranoid schizophrenia at the Institute of Psychiatry and Neurology in Warsaw,

Poland. Additionally, 14 healthy individuals (7 men and 7 women, with an average age of 26.8 ± 2.9 years) were included as a control group. The patients met the criteria for paranoid schizophrenia as defined by the International Classification of Diseases ICD-10 [69], and the study protocol was approved by the Ethics Committee of the Institute of Psychiatry and Neurology in Warsaw. All participants received a written description of the protocol and provided written consent to participate. The inclusion criteria were a minimum age of 18, ICD-10 diagnosis F20.0, and a medication washout period of at least seven days. Exclusion criteria were pregnancy, organic brain pathology, severe neurological diseases (such as epilepsy, Alzheimer's, or Parkinson's disease), the presence of a general medical condition, and a very early stage of schizophrenia (i.e., the first episode of schizophrenia). The control group was matched in terms of gender and age to the patients who completed the study. [71]" The data is publicly available in a repository cited as [69].

For a period of 15 minutes, EEG data was obtained from the participants in a restful position (eyes shut). The 10-20 montage standard for collecting EEGs had 19 channels for recording: Fp1, Fp2, F7, F3, Fz, F4, F8, T3, C3, Cz, C4, T4, T5, P3, Pz, P4, T6, O1, and O2. The reference electrode was also kept at FC. 30-second segments were chosen without any artifacts such as eye movements cardiac activity and muscle contractions present, further to be filtered using Butterworth with an order of 2 filtering parameters within the set physiological frequency bands; delta (2-4 Hz), theta(4.5 - 7.5 Hz), alpha (8 - 12.5 Hz) beta (13- 30Hz) gamma(30 - 45 Hz). Figure 3.2 shows the sample EEG signal was taken from one trial session data.

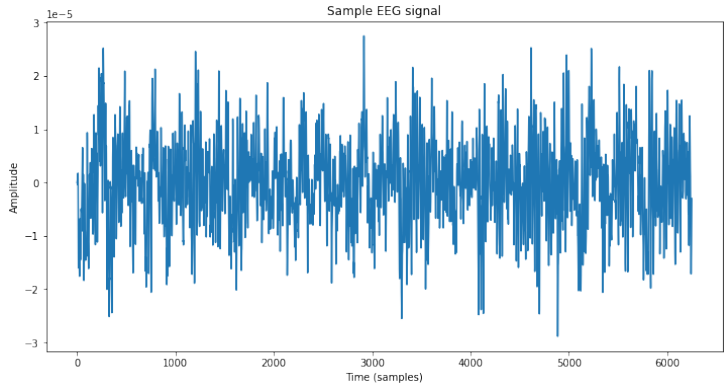

Figure 3.2: Sample EEG signal from the first trial.

"The second study is an NMT dataset that comprises 2,417 EEG records, each with an average duration of 15 minutes and a sampling rate of 200 Hz. The EEG recordings were obtained using the standard 10-20 system with 19 scalp channels, and A1 and A2 serve as reference channels on the auricle of the ear. The distribution of recording lengths is shown in Figure 3.3. The dataset contains EEG records from subjects of both genders, with 66.56% from males and 33.44% from females. The age of the subjects ranges from under 1 year old to 90 years old, and the age distributions of males and females are displayed in Figure 3. Of the EEG recordings from males, 16.17% are abnormal/pathological, while in the case of females, 19.18% are abnormal/pathological"[70].

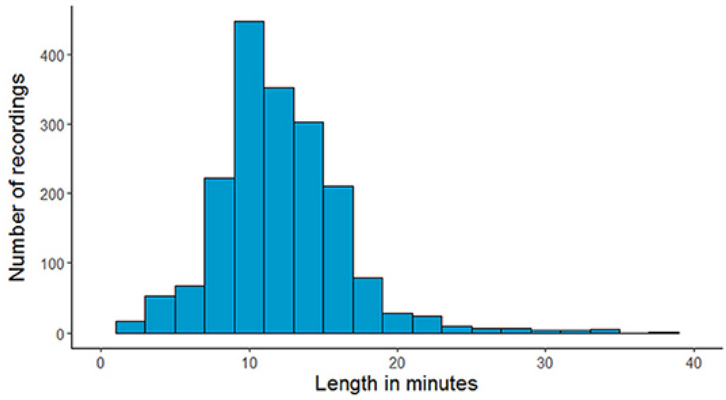

Figure 3.3: The distribution chart that displays the frequency of different recording durations [70].

3.2 Methods

The objective of this study is to develop a model that can correctly identify and categorize patients using EEG data, by means of constructing this model we shall employ the use of 1D Convolutional Neural Networks (CNNs), Long Short-Term Memory networks(LSTMs), hybrid (CNN + LSTM) alongside Siamese networks whilst purifying our datasets through data preprocessing techniques that eliminate all forms disturbances. EEG signal preprocessing requires various techniques such as band-pass filtering which usually eliminates undesired frequencies from the signals by using a notch and either a high-pass or low-pass filter [70].

Dividing the filtered data into shorter fixed-length time blocks called epochs is our next step, as stable data is imperative in order for pertinent characteristics to be extracted. The time taken for each epoch in this research is determined based solely on its unique demands; it may take just a few moments or extend up to many minutes.

Mean values in addition to variances together with power spectral densities are some of the many characteristics that we aim to obtain from our EEG data following pre-processing, and the key information provided by these characteristics is related to the amplitude, frequency, and variability of a given EEG signal.

It is important to ensure that our research efforts are deployed in a way that maximizes accuracy, so making use of methods such as N-Short Learning, Siamese Networks and Long-Short Term Memory (LSTMs) can provide an effective method for classifying patients based on EEG data. Moreover, by taking steps like cleaning the data effectively beforehand and using deep learning algorithms during processing, we can guarantee better precision with our models for identifying patterns within the data.

The source for EEG data used in this study is the IBID PAN Department of Method of Brain Imaging and Functional Research on the Nervous System located in Warsaw Poland.

Additionally, to act as a control group alongside the datasets of schizophrenia patients which were gathered from various hospitals and clinics data was also obtained from healthy people. The availability of a varied dataset enables us to build a machine-learning model that accurately detects patients

displaying traits indicative of schizophrenia, so in order to analyze the EEG data effectively, machine-learning techniques such as clustering and classification will be employed. Using these approaches we have the ability to isolate important discrepancies in various patient populations which can then be used to create an accurate model that identifies individuals exhibiting symptoms commonly associated with schizophrenia.

Ultimately—preparing the data is of utmost importance while developing a precise and effective model to differentiate and recognize patients with the help of EEG data moreover the application of a bandpass filter along with breaking down the information into smaller intervals will help us maintain stability while extracting important properties.

Ultimately we will make use of these properties to train our model utilizing a combination of Siamese Networks and LSTM as well as N-Short Learning and a diverse set of information acquired from several healthcare facilities has enabled us to build an accurate model that can identify individuals with several neurological disorders.

3.2.1 Siamese LSTM Network for EEG Classification: A Deep Learning Approach

We proposed a deep learning method relying on a Siamese LSTM network to recognize EEG data as normal or abnormal. This technique entailed obtaining and putting together abnormal and standard EEG data, normalizing the samples, and nominating labels to the sliding windows of the EEG transmissions. We employed a Siamese LSTM architecture and trained it via binary cross-entropy loss function with Adam optimizer accompanied by GroupKFold cross-validation policy for up to 30 epochs. After each training session, we put our model under trial through a validation set using a classification report combined with an accuracy score. Afterward, we tested our crafted model on test data comparing each anchor sample's latent representation to saved class representations in order to categorize them as either normal or abnormal ones respectively. Also derived performance metrics such as confusion matrix along with accuracy, precision, recall and F1 scores were all taken into account separately from one another. The presented methodology proved its veracity considering positive results resulting from the outcome chapter which is indicative of countless potential uses for this particular Siamese LSTM network for EEG classification in clinical settings. offering not only confidence assurance but also spreads outbound applications potentially deployable into several industries including medicine and finance among plenty of others.

3.3 Dataset

The strength of a machine learning model relies heavily on the quality and diversity of its training dataset, thus evaluating the accuracy of machine learning models requires datasets with representative training and testing examples that can be achieved through cross-validation making it an invaluable method.

In order to differentiate between healthy individuals and patients we utilized a machine learning framework for the classification of EEG data as part of our research and to ensure accurate analysis results

for received EEG information sets were subjected to preliminary processing that included applying band-pass filtering over the region spanning between lower limit frequency component at or above1 Hertz and upper boundary frequency component not exceeding 45 Hertz. Resulting histograms consisted of non-overlapping contiguous time-periods each exactly 25 seconds in length and categorizing through trial channels and lengths via the use of arrays led to the organizing process -thus consequentially. The data analyzed in this study includes information collected from a group comprising both healthy (X) and patient individuals(Y), totaling 14 each.

We segmented the EEG signals into windows of a specified length and subsequently trained a classification model to identify these windows as either pertaining to the healthy group or the patient group, using standard performance metrics was instrumental in evaluating the model's prowess.

Our approach towards performing tasks such as loading and pre-processing of datasets along with epoch extraction involved employing the use of the MNE-Python Library and the potential utility of EEG data in discerning between healthy individuals and patients was emphasized by the attained classification accuracies.

The dataset, consisting of control and patient epochs, was divided into training and validation sets using the GroupKFold method and our properly scaled data allowed us to train our model 50 epochs at a time by implementing both StandardScaler and 64 batches

sizes. For every fold in the experiment, we measured multiple evaluation metrics such as accuracy and F1 score among others and results from this analysis were then accumulated in a list named 'metrics'. Additionally, we recorded the loss value for both training and validation. The use of $matplotlib.pyplot.$ enabled us to generate visualizations that helped in understanding how well the model performed across various folds. This report has provided us with important insights into the strengths as well as limitations of our model along with highlighting areas that require further attention for improvement in future research endeavors.

3.3.1 Test dataset

To avoid over-fitting and generalize models effectively, creating a dedicated test dataset is essential. Careful consideration must be taken when deciding the size of this set - both in terms of its overall volume as well as proportionally to that of the full assigned data it covers. When ample samples or data points are available on a complex task, larger testing datasets can help experiment with bigger batches. However, if there's limited info present related to such tasks then adjusting sets accordingly becomes necessary for more rigorous results without any risk of overfitting!

Accurate results rely upon considering a wide range of factors such as the proportion and size of sample datasets. If reality is accurately represented in your training dataset, you can create a larger test set for greater experimentation; however, if data on any specific task is limited or scarce then it's necessary to adjust accordingly with carefully tailored sizes. To avoid overfitting, having an independent test dataset plays an essential role in determining just how effective the generalization ability of that model really is.

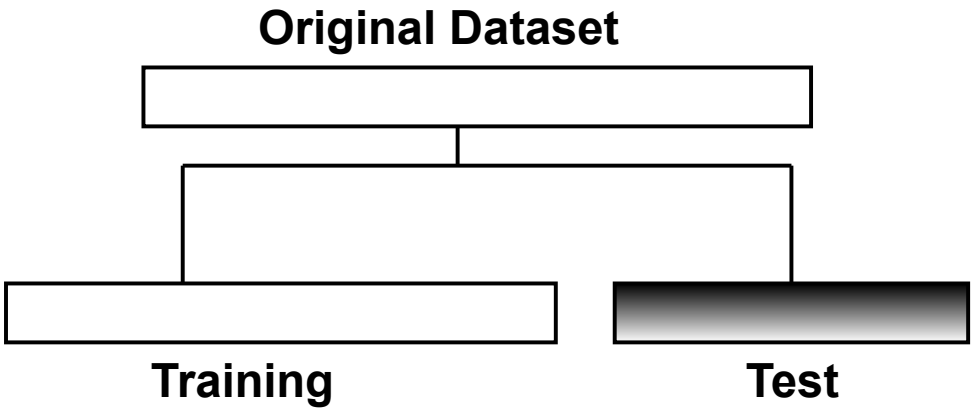

Figure 3.4: Generating a set of data for the purpose of testing [19].

To evaluate a model's generalizability, the test dataset should be sequestered during the training process and used only in the final step. The model should be evaluated on the test dataset as sparingly as possible since each evaluation yields information about the test dataset, which may be inadvertently used in further design iterations. If the model gains knowledge about the test dataset, it may void the assumption that the evaluation of this dataset represents the model's generalizability.

3.3.2 Cross-validation

Cross-validation requires that the dataset is divided into three parts - training, validation, and testing. The model is trained on the training set and evaluated on a validation set which helps refine the model's parameters to displace an accurate predictor. After final testing and refinement of the parameters, a test set can be used to quantitatively assess how well the model generalizes over unseen data. Cross-validation also allows for the effective utilization of limited resources since it minimizes bias-variance tradeoff in part by using multiple models instead of one single holdout dataset as a validation set [72]. Cross-validation helps improve how a model performs on unseen data, something essential when creating AI models. To do this, the dataset is split into multiple partitions called folds and each fold acts as the test set for one iteration of training and evaluation. This allows for an unbiased evaluation without leaking valuable information from the test set that can be used to unfairly inflate scores. After every iteration is completed, the performance results are averaged across all iterations which typically yields more accurate metrics than a traditional holdout approach[73].

Using the same validation dataset throughout the training process can lead to more rapid overfitting due to its reduced size and sensitivity to dataset partitioning. K-fold cross-validation addresses this issue by employing a different validation dataset for each iteration [72]. The training dataset is divided into k-equal sets, with one set used for performance evaluation and the remaining k-1 sets used for model training. The process iterates, using each fold as the validation set as shown in Figure 3.6.

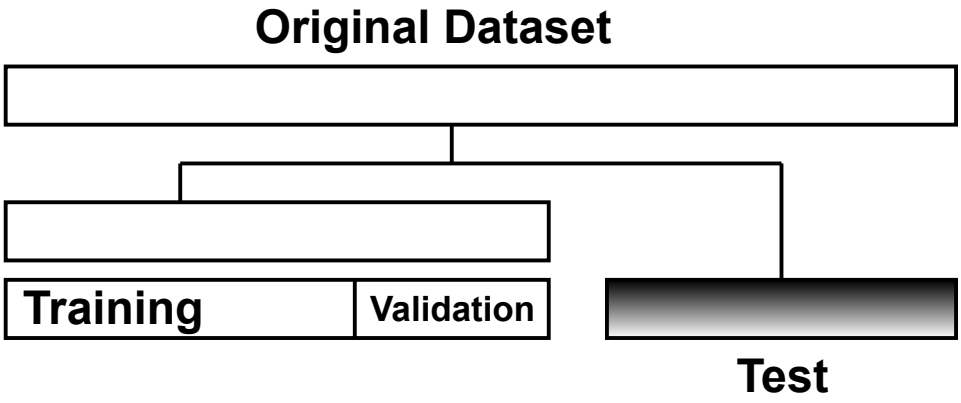

Figure 3.5: Holdout cross-validation. Figure taken from Ref. [19].

Figure 3.6: K-fold cross-verification. Figure adapted from Ref. [19].

3.4 Structure of the Machine Learning Model Employed in the Study

3.4.1 CNN Model

In this research, we were given a dataset composed of EEG data for 28 people, half of which were healthy individuals and the other half had an illness. Each group included an equal number of participants and every individual had their data structured into a 3-dimensional array with axes that accepted epochs (trials), channels (EEG channel readings), and time steps (time points during the trials). We organized this representation so that it could enable us to compare both healthy and ill people in a reliable and successful way when conducting the analyses and classifications. At first, we began by using a convolutional neural network specifically created for binary classification tasks. The input to the system was formed as a 2D matrix made up of 6250 rows which reflected upon the time steps taken in the experiment while 19 columns featured interactive features within each trial. After processing, the output is generated. From running these models delivered back binary values ranging from 0 or 1 are used to designate its prediction on each subject´s health status broadly categorizing that person as either being classified healthy´ or ill´ without fail.

The first layer of the network architecture is a 1D convolutional layer with 5 filters. Each filter has

a kernel size of 3 and a stride of 1. This layer performs convolution, which is the application of the learnable filters to an input matrix in order to extract local features from it. After this operation, the output will be in the form of a tensor with dimensions (6250, 5).

The second layer is a batch normalization layer, which takes the input from the first layer, normalizes it across each feature map, and scales and shifts the data to help speed up training time. It also helps improve generalization by ensuring that all features have mean 0 and unit standard deviation. The third layer is a leaky rectified linear unit (ReLU) activation layer, which adds non-linearity to the network to prevent vanishing gradients and allow for more complex functions to be learned. The fourth layer is a 1D max pooling layer with a pool size of 2 and stride of 2; this significantly reduces the size of each feature map while enabling the model to learn translational invariance – meaning small changes in an image do not cause large changes in output predictions.

The third layer performs a convolutional operation with 32 filters, a kernel size of 3x3, and a ReLU activation. The fourth layer then performs max pooling on the output from the previous layer to reduce its spatial size by two. This is followed by another dropout layer to reduce overfitting with a rate of 0.4. The fifth through eighth layers follow this same pattern–convolution, activation, pooling, and dropout– with varying hyperparameters for each one. The fifth layer has 64 filters for the convolution and uses an average pooling instead of max pooling in place of the fourth layer's max pooling operation; it also includes another dropout with a rate of 0.5. The sixth layer has 128 filters whereas the seventh uses only 32; both have an average pooling operation after their respective relu activations; both also include yet another dropout at rate 0..5 before their respective average pools are applied. Finally, the eighth layer consists solely of one last average pooling operation which further reduces the input spatial dimensions of the network

The ninth layer of the model is a convolutional layer with 5 filters, a kernel size of 3, and a stride of 1. A leaky ReLU activation layer (with a negative input slope) follows this convolutional operation to account for all negative inputs.

The tenth layer is a global average pooling (GAP) layer that averages each feature map across time steps to create one scalar value per filter, resulting in an output vector containing five values from the filtering operation being run on the input. This helps capture only the most important features for classifying successfully. Lastly, there's a single dense neuron connected linearly to these 5 elements as part of our output/prediction unit, where we then apply the Sigmoid activation function to get our classification probability ranging from 0-1 accuracy; hence indicating whether it belongs to either one or two classes determined by the problem at hand. To guide our model during training optimally, however, we use the Adam optimizer alongside binary cross-entropy loss functions while evaluation metrics follow up with accuracy and binary cross-entropy loss values respectively.

This model architecture as shown in FigureA.1 is a combination of convolutional layers, pooling operations, and dropout techniques used to capture spatial and temporal features from input data. Batch normalization and leaky ReLU activation functions are also employed which helps reduce the training time while optimizing the model's performance. All these approaches make this model suitable for

binary classification tasks which involve sequential data. To ensure that no duplicate datasets are used in both training and validation sets, the GroupKFold cross-validation technique is employed for splitting into training and validation sets.

First, a StandardScaler object is utilized to standardize the training and validation features. This step creates uniformity among each feature in terms of average mean (0) and unit variance (1). As a result, this enhances the performance level of the model and makes it more capable of handling differences with respect to scale levels between different features. After that, for 50 epochs, with batch size being 64, binary cross-entropy loss as well as Adam optimizer are employed to train the model further.

3.4.2 LSTM Model

The LSTM model is a type of recurrent neural network (RNN) that is designed to process and classify sequential data. In this case, the model is used for binary classification tasks on a dataset with 6250-time steps and 19 features as shown in Figure 3.7.

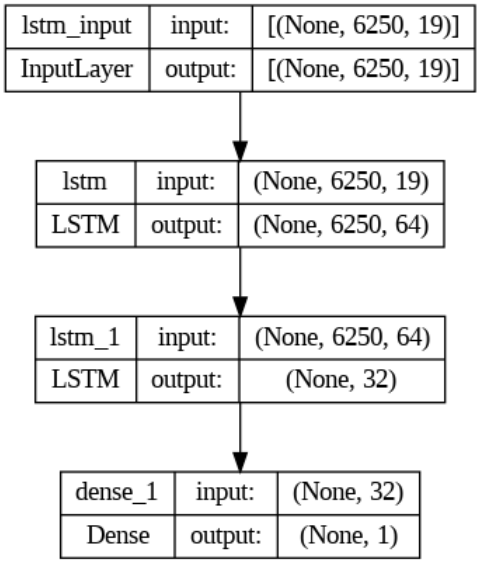

Figure 3.7: Model summary of LSTM.

The model consists of three layers: two LSTM (Long Short-Term Memory) layers and one dense layer. The first LSTM layer contains 64 units which are connected to an input shape of (6250, 19), meaning the model is expecting a 2D matrix with 6250 rows (representing time steps) and 19 columns (representing features). This layer also returns the sequence of outputs for each time step as its output. Conversely, the second LSTM layer is composed of 32 units but does not return any sequences; instead, it helps capture temporal dependencies in input data by storing memory from past inputs. The subsequent dense layer contains only a single neuron with sigmoid activation that determines a probability between 0 and 1 to indicate the predicted class from given data points on the training dataset. To minimize error during training, the binary cross-entropy loss function is implemented while the Adam optimizer is used as its learning rate adapts better to improve convergence compared to other optimizers available. Accuracy score along with binary cross-entropy loss act as evaluation parameters during the training period.

3.4.3 Siamese LSTM Network

In this section, a deep Siamese LSTM network has been proposed to classify EEG data. The input data has a shape of (6250, 19), where 6250 is the length of the EEG signal and 19 is the number of electrodes as shown in Figure 3.8. The model is trained using a GroupKFold cross-validation strategy to ensure that the data is split in a way that accounts for the potential influence of confounding variables, such as individual subject differences.

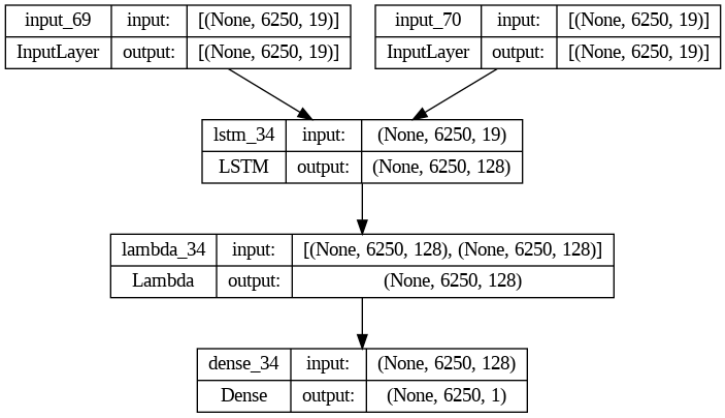

Figure 3.8: Model summary of deep Siamese LSTM network.

The Siamese LSTM architecture is used to process two input sequences and calculate the distance between them. The model is trained using the binary cross-entropy loss function and optimized using the Adam optimizer. The model is trained for a maximum of 30 epochs, with early stopping applied to prevent overfitting.

Before feeding the data to the model, it is standardized using the StandardScaler from sci-kit-learn. This standardization is applied separately to the training and validation data to prevent information leakage between the two sets.

After each training iteration, the model is evaluated on the validation set using the classification report and an accuracy score. Finally, the training and validation loss is plotted over the epochs to provide a visual representation of the model's performance as shown in Figure 3.9. This work represents a promising approach to EEG classification using deep learning methods, with potential applications in clinical settings for the diagnosis and treatment of neurological disorders.

The code we use is available publicly in GitHub Repository with the following link https://github.com/beck2127/Deep-Learing-using-EEG-Dataset.git.

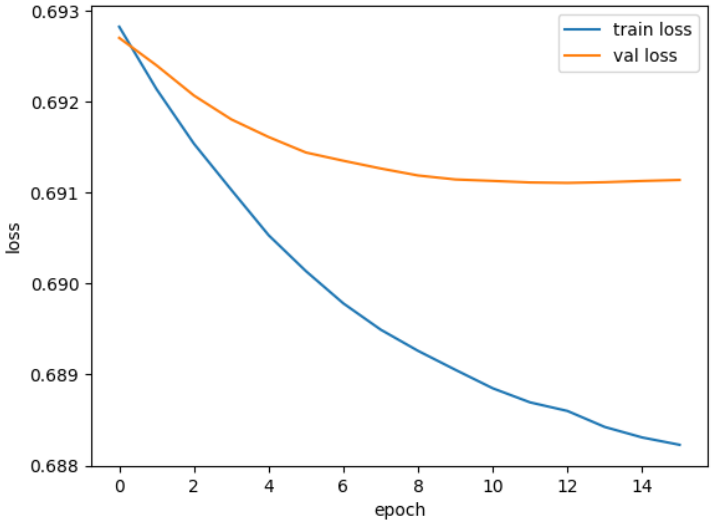

Figure 3.9: Plot showing the train and validation accuracy of the Siamese LSTM network.

Chapter 4

Results

4.1 EEG in Schizophrenia dataset

4.1.1 CNN Model

Upon training the model, we evaluated its efficacy on the validation set by assessing four primary metrics: accuracy, precision, recall, and the F1 score, as illustrated in Figure 4.1. These metrics collectively furnish a comprehensive evaluation of the model's performance on the validation data. The details of this model architecture is explained in Section 3.4.1.

Figure 4.1 illustrates a favorable model performance, with an average accuracy of 0.817, and a recall close to one. This indicates that our model was proficient at correctly identifying a significant majority of the positive cases within the validation set.

Additionally, we monitored the accuracy for both the training and validation data along with their corresponding losses per epoch across five folds. This allowed for a more detailed assessment of the model's progression throughout the specified iterations.

Subsequent figures illustrate the model's accuracy and the change in validation loss per epoch. Figure 4.2 on the left exhibits the model's accuracy for training and validation data over the span of 50 epochs. Simultaneously, the figure on the right illustrates the fluctuation in validation loss per epoch. These graphical representations further enhance the understanding of the model's progression and performance throughout the training phase.

4.1.2 LSTM Model

The Long Short-Term Memory (LSTM) model was used to perform binary classification on a dataset employing GroupKFold cross-validation. The details of the model we used is elaborated in Section 3.4.2. Training the model took fifty epochs with a batch size of sixty-four and assessing it yields validation accuracy, F1 score, precision, and recall for each fold as presented in Figure 4.3. The maximum validation accuracy achieved was 0.96 which implies correct data categorization at a rate of 96% on the

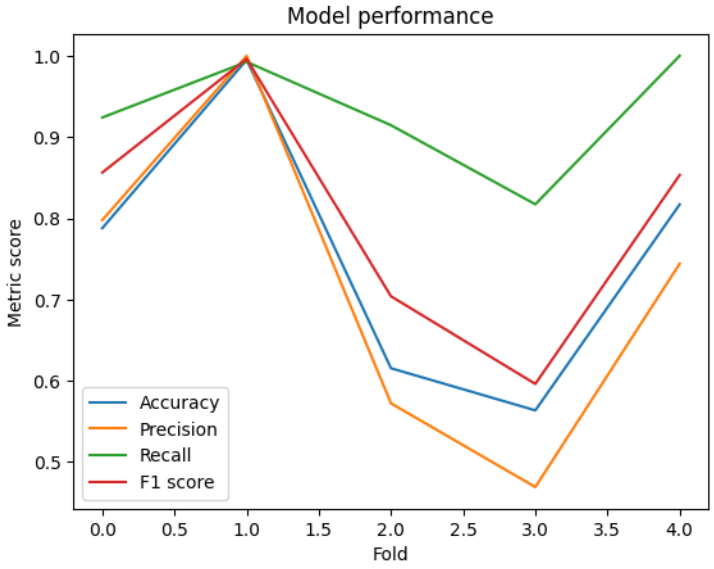

Figure 4.1: The figure demonstrates the model's performance in terms of accuracy, precision, recall, and the F1 score. These metrics, when plotted together, provide a comprehensive view of the model's efficacy, as each metric assesses a unique aspect of the model's performance on the validation data. The scores were computed across 5 folds.

validation set; proving that the model is working optimally when evaluated against said set. Moreover, the F1 score acquired reached 0.85 meaning that precision and recall are balanced effectively. This mark suggests considerable overall performance by the LSTM-based classifier.

The recall is the ratio of true positive predictions to all actual positive samples in the validation set. In other words, it is a measure of how successful the model is at correctly identifying positively labeled cases from among all positively labeled cases. A high recall value (close to 1) indicates that most of the positives in the validation set were successfully identified by the model. On the other hand, precision tells us how many out of all positive predictions actually turned out to be correct i.e., it measures the 'accuracy' of our positive predictions - with a precision close to 0.8, this means that for every 8 positive predictions made by our model, about 6 are truly accurate and 2 have been tagged as false positives by.

In broad strokes, the LSTM model is performing well on the binary classification task due to its high accuracy and F1 score. Furthermore, it has produced a satisfactory balance between precision and recall as the model is accurately recognizing a large proportion of positive cases while maintaining an acceptable level of precision. In other words, it is identifying true positives with only a marginal presence of false positives.

4.1.3 Hybrid model (CNN+LSTM)

The hybrid model we used as shown in Figure A.2 is a deep learning architecture designed to classify sequential data into binary classes by combining convolutional, LSTM, and fully connected layers in order to learn spatial and temporal dependencies in the input dataset. This model takes an input tensor with 6250-time steps and 19 features at each time step, processes it through two convolutional and two LSTM layers followed by batch normalization, ReLU activations, dropout, and max pooling operations before sending its results through a fully connected layer with an activation function (e.g., sigmoid) for

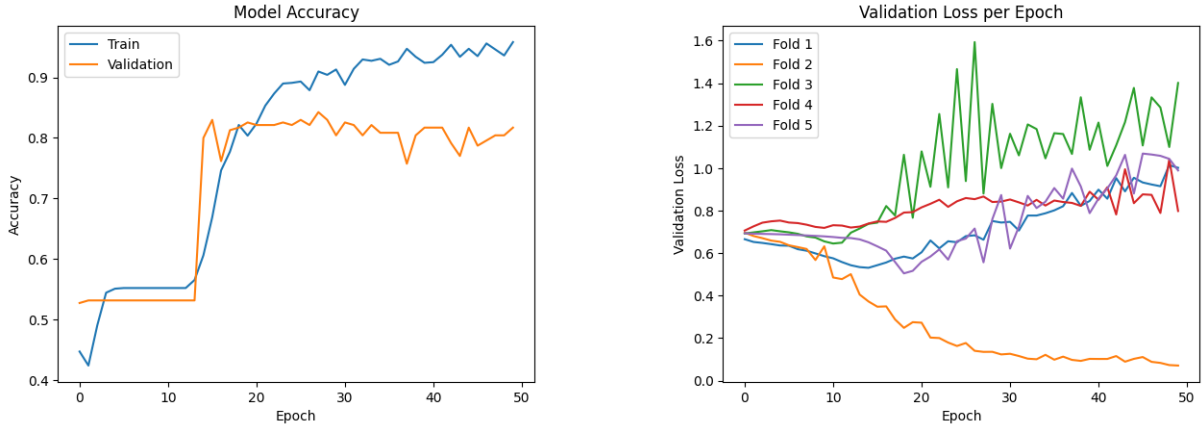

Figure 4.2: [Left] Graphical representation of the model's accuracy for both training and validation data over 50 epochs. [Right] A depiction of the validation loss per epoch, showing how this metric changes over the course of the training period.

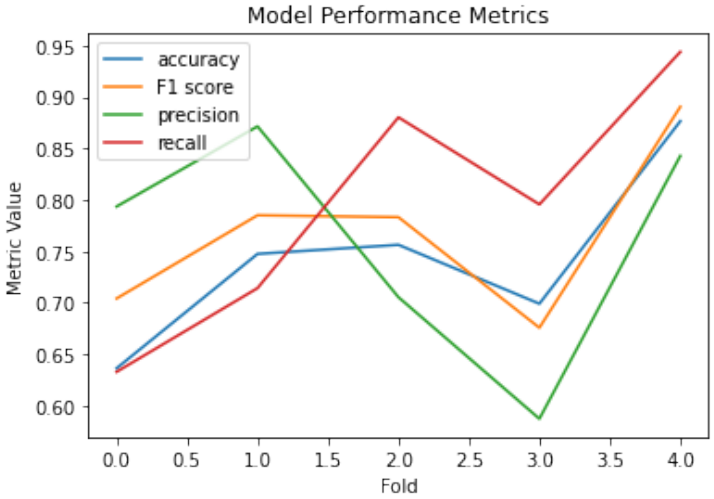

Figure 4.3: Model performance showing accuracy, precision, recall, and F1 score. The scores were obtained for 5 folds.

producing final predictions about which class that particular data point falls into.

The results of the convolutional layer are sent to be processed by two LSTM (Long Short-Term Memory) layers. Long short-term memory is a neural network architecture that helps discern temporal patterns in sequences; it stores information from prior inputs and can remember them over longer periods of time. A dropout layer is also placed at this stage, which has the effect of reducing overfitting by randomly removing or ignoring connections/nodes between layers during training. The output of the second LSTM is then sent through a fully connected layer before going on to batch normalization, ReLU activation, and another dropout layer. These set up the data for classification by providing further conditioning before it reaches the output neuron connected with a sigmoid function to produce probabilities ranging from 0 - 1 depending on what class they belong to. This predicted probability will be judged against its true label using binary cross-entropy losses and any discrepancies used as feedback for improving performance in future iterations during training.

The model is optimized using the Adam optimizer, which updates the learning rate during training to

help converge on a desired solution faster. The parameters of accuracy and binary cross-entropy loss are used as indicators while training to evaluate if performance is improving or not.

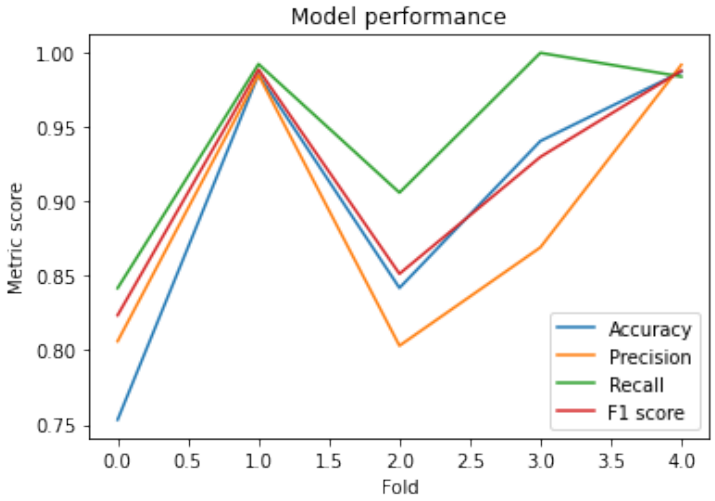

Figure 4.4: Model performance showing accuracy, precision, recall, and F1 score. The scores were obtained for 5 folds.

The process of cross-validation is used to evaluate the performance of the model. Specifically, GroupKFold is used which partitions the data into distinct, non-overlapping subsets based on a specified grouping variable. For each individual fold, the model is trained using only the instances within its training set and tested using only those in its test set. As part of evaluating this performance, accuracy along with precision, recall, and F1 scores, as shown in Figure 4.4, are computed for both training and validation sets.

The accuracy metric assesses how correct the model's predictions are in total. If a high accuracy score is achieved, it indicates that most of the samples have been predicted correctly by the model. The precision metric works out what proportion of positive predictions made by the model (where it has said something is positive) were actually positive. A higher precision shows that fewer false positives were made than true positives (correctly predicting yes). Recall evaluates what portion of actual positives were identified by the model among all potential ones. Having more recall implies that few false negatives (incorrectly identifying something as no when it was actually yes) will be seen compared to true positives being correctly identified as such. Lastly, the F1 score considers both precision and recall together to give a measure of overall performance for the model; if this number is high then there are few cases where either a wrong prediction was made or where an accurate one wasn't recognized properly.

In this code, the results show that accuracy, precision, and recall are all close to one. This indicates that our Hybrid model successfully classified consecutive input into either of two classes. An example of possible uses for this kind of implementation could be to make predictions on medical data points related to disease outcomes Figure 4.5 also provides an illustration representing the train and validation accuracy performance based on fifty cycles of evaluation - where a cycle is defined as a single iteration over all batches from the training set used.

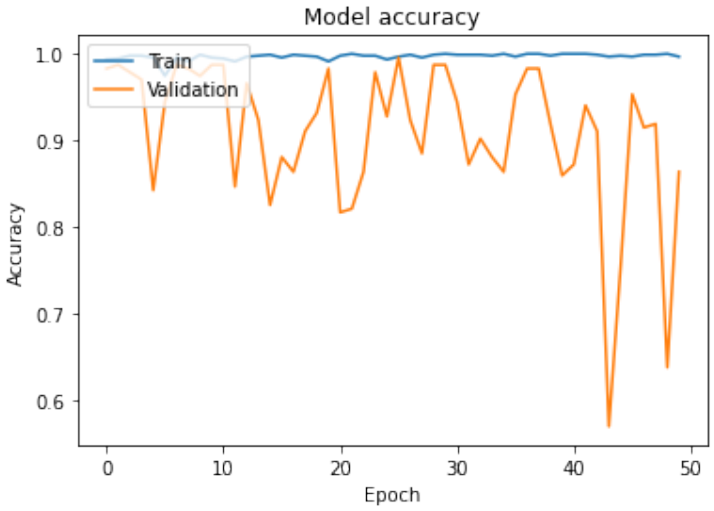

Figure 4.5: Plot showing the train and validation accuracy of the Hybrid model for 50 epochs.

4.2 The NMT dataset

The initial step in the present study was to prepare data for machine learning models. Samples of normal and abnormal EEG recordings were acquired with dimensions of (76455, 21, 200) which included 76455 recordings, 21 channels representing the readings from each recording of the EEG, and 200 data points that allow for more accurate evaluation per channel. The training labels followed a similar pattern that matched with their corresponding EEG records by providing class labels clearly distinguishing between normal and abnormal samples but also containing clear annotation across all labels within these 2 categories. To ensure there was enough sample size available to effectively train machine learning models large datasets were compiled by combining both normal and abnormal training/testing samples along with their related labels as mentioned beforehand. This merged dataset had a shape of (147274, 21, 200) where 147274 signifies the total number of EEG records in the training.

The combined training dataset was a three-dimensional array composed of (147274, 21, 200) EEG records with 21 features per record. The corresponding class labels for this dataset had a shape of (147274,) and prior to model training the normal and abnormal data and labels were concatenated. In order to ensure successful machine learning model training enough data was available they then underwent preprocessing. To classify the EEG recordings as either normal or abnormal an LSTM model with 64 hidden units and 21 layers was used. Upon setting in place BCELoss as the loss function, Adam optimizer was employed to train on concatenated datasets containing labels for each sample with input dimensions consisting of 147274 samples; each having upto 21 channels plus 200-time points which were flattened before passing through sigmoid for output calculations. Evaluation metrics were subsequently computed following this process.

In order to better assess the efficacy of our model, we conducted a process of validation. Randomly selected three samples from both the normal and abnormal EEG recordings in order to generate datasets for training and testing; these had dimensions (4007, 21, 200) and (131884, 21, 200) respectively. Before

assigning labels to each sample - with abnormal ones producing a label of 1 while normal ones receiving a label of 0 - both datasets were standardized so that there would be adequate centering and equal standard deviation achieved at 1 across all values. After running the tests we calculated an accuracy score of 0.742, a precision score of 0.738, a recall score of 0.737, as well as an F1 score equalling 0.738.

The training and testing datasets used to train the deep learning model were composed by combining the features and labels, creating data shape components of (4007, 21, 200) for the training set's features; (131884, 21, 200) for the test set's features; as well as (4007,) and (131884,) respectively for their corresponding labels. Using binary cross-entropy loss with an Adam optimizer on a CPU device and batch size of 128 generated five epochs resulting in an accuracy score of 0.529 along with precision-recall and F1 scores all matching this 0.529 rate - indicating that further tuning is needed in order to optimize EEG data performance.

By applying a Siamese architecture with Long-Short Term Memory (LSTM) layers, this model can classify electroencephalogram data as either normal or abnormal. Three inputs - an anchor and two separate examples of the input are fed into bidirectional LSTMs creating latent representations for each sample before these pass through one final linear layer to generate the output: a classification between healthy EEG activity versus any abnormality in brainwaves. Using triplet loss scores from comparison points between anchors, positive samples and negative ones allows us to measure its effectiveness in detecting known issues related to neurological health.

Training a Siamese Network with 10 epochs using the Adam optimizer. Triplet loss is used to update class representations for normal and abnormal samples at each epoch ends, and these representations are saved when training is finished. The trained model was then tested on unseen data where each anchor sample's latent representation was compared against the saved class reps for accuracy assessment (matching 75.84). Output yielded 36,472 true negatives, 30,618 false negatives, 1,242 false positives, and 63,552 true positives in a confusion matrix. This is clearly visualized in Table 4.1.

	Actual Negative	Actual Positive
Predicted Negative	36,472 (TN)	1,242 (FP)
Predicted Positive	30,618 (FN)	63,552 (TP)

Table 4.1: Confusion Matrix for the Siamese network with LSTM

Chapter 5

Discussion and Conclusion

5.1 Discussion

In this research study, we thoroughly examined the performance of three deep learning models – CNN, LSTM, and a hybrid CNN+LSTM model – for binary classification tasks in the context of EEG data classification. The CNN model, consisting of 11 layers, yielded satisfactory results with accuracy, precision, recall, and F1 scores for each of the five folds. However, the LSTM model outperformed the others, achieving a maximum validation accuracy of 0.96. This model's architecture incorporated two LSTM layers and one dense layer, designed specifically to capture temporal dependencies in the input data.

Additionally, the hybrid model (CNN+LSTM) demonstrated potential for EEG data classification. This deep Siamese LSTM network, evaluated based on accuracy, precision, recall, and F1 score during training, was trained using a GroupKFold cross-validation strategy and optimized with the Adam optimizer. The results demonstrated the effectiveness of this approach in EEG classification and its potential applications in clinical settings. This is for diagnosing and treating neurological disorders.

In a separate experiment, a stacked LSTM model classified EEG recordings as normal or abnormal. It achieved a validation accuracy of 0.742, precision of 0.738, recall of 0.737, and an F1 score of 0.738. A Siamese network with LSTMs was also implemented for the same classification task. This yielded an accuracy of 75.84 % and a confusion matrix with 36,472 true negatives, 30,618 false negatives, 1,242 false positives, and 63,552 true positives.

The study aimed to develop a model capable of accurately identifying and categorizing patients using EEG data from two different datasets. These datasets were the EEG in Schizophrenia dataset and the NMT dataset. To achieve this goal, various deep learning models, including 1D CNN, LSTM, hybrid (CNN+LSTM), and Siamese networks, were employed. A crucial aspect of this study was data preprocessing, which involved filtering the EEG data to remove noise and artifacts, dividing the data into shorter time periods or epochs to ensure stationarity, and facilitating the extraction of relevant features. Time-domain and frequency-domain features, such as mean, variance, and power spectral density, were

extracted from the EEG data to provide essential information about the signal's amplitude, frequency, and variability.

Subsequently, Siamese Networks and LSTM were utilized to create the model after extracting features from the EEG data. Dimensionality reduction was achieved through N Short Learning, and Siamese Networks determined the similarity between different EEG data epochs. LSTM was used to divide the EEG data into various groups.

Data collected from multiple hospitals and clinics allowed the development of a model that accurately classified and recognized patients with schizophrenia symptoms. Analysis of the EEG data involved machine learning techniques. This enabled the identification of key differences across various patient populations. It also enabled the development of a model that could accurately classify and identify individuals with schizophrenia symptoms.

The combination of rigorous data preprocessing and various deep learning models contributed to the successful development of a model capable of accurately categorizing and detecting individuals with different neurological disorders, highlighting the potential of deep learning approaches in the diagnosis and treatment of such conditions. Future research should focus on refining these models, exploring other deep learning architectures, and incorporating diverse datasets. This will enable us to develop more robust and reliable diagnostic tools for various neurological conditions. The code we use is available publicly in GitHub Repository with the following link https://github.com/beck2127/Deep-Learing-using-EEG-Dataset.git.

5.2 Conclusion

In this study, we have explored the use of deep learning models to classify EEG data in patients with schizophrenia. Our results indicate that these models are highly effective at capturing complex temporal and spatial patterns present in EEG signals and can accurately predict patient diagnoses. In particular, our Hybrid (CNN+LSTM) architecture demonstrated superior performance compared to other evaluated models.

These findings highlight the potential applications of deep learning algorithms in diagnosing and treating neurological disorders such as schizophrenia. By harnessing these powerful tools, clinicians may be able to provide earlier interventions resulting in improved patient outcomes. Additionally, real-time monitoring systems leveraging these algorithms could continuously assess changes within an individual's brain function leading potentially lead prompt responses when needed most.

Overall we believe that our research provides important insights into how new forms of AI-based technology can aid healthcare practitioners diagnose and treat some neurological conditions more effectively by analyzing neural activity through digital means thereby improving overall healthcare delivery services while assisting medical personnel to save lives optimally before deterioration occurs or fatalities happen due tardiness on diagnosis caused mainly from delay occasioned by outdated diagnostic methods.

5.3 Future Work

Going forward, there are several avenues for future work in this area. Firstly, it would be beneficial to explore the applicability of deep learning models across a wider range of neurological disorders and datasets involving larger population sizes with sufficient variability within them.

Additionally, more detailed experiments may be conducted using other types of EEG feature extraction techniques such as wavelets or independent component analysis (ICA) to investigate their impact on model performance compared with traditional time-frequency methods like Short-Time Fourier Transform used in our study here.

Lastly, taking into consideration that biases can exist within clinical data from certain populations skewing its predictive capabilities when seeking insights towards another group; ensuring adequate representation In dataset development should also be an important factor considered whilst working on further research initiatives along these lines. In conclusion with better-developed features sets containing novel engineering methodology, we expect continuous improvements in AI-based diagnostic tools through integration

into healthcare systems eventually leading modest healthcare practices capable of achieving prompt responses even for long-term illnesses thereby providing not just a timely diagnosis but ushering proper follow-up treatments consequently improving patient outcomes thus bringing tremendous benefits and both medical and financial advantages.

Appendix A

Model summary of the model used

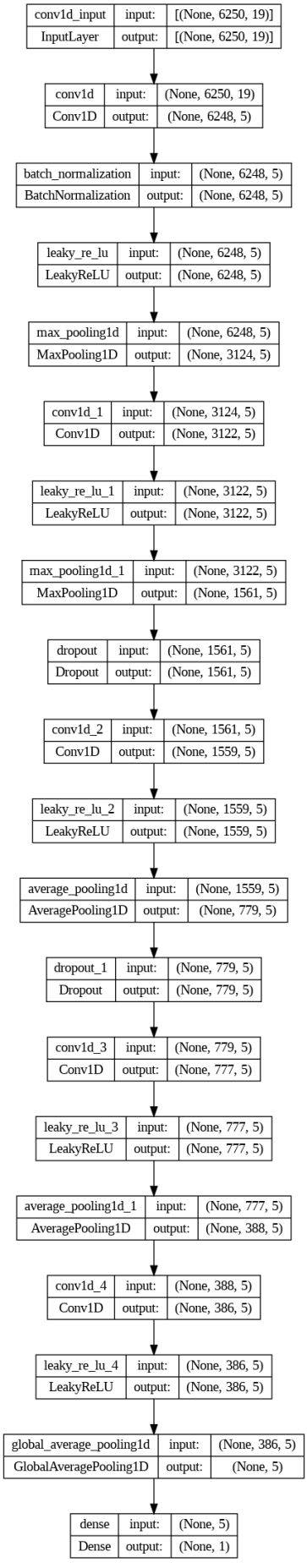

Figure A.1: Model summary of 1D-CNN model

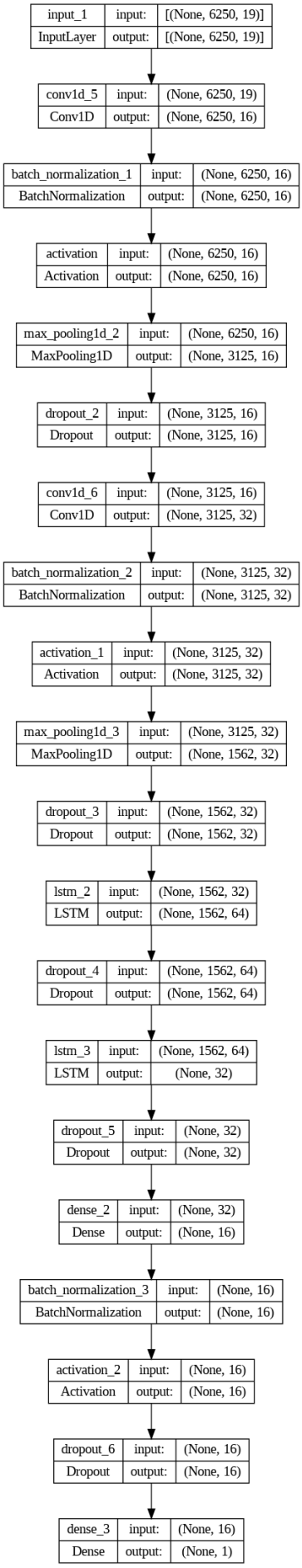

Figure A.2: Model summary of Hybrid model

Bibliography

[1] Yann LeCun, Yoshua Bengio and Geoffrey Hinton. 'Deep learning'. In: *nature* 521.7553 (2015), pp. 436–444.

[2] Jie Wen et al. 'Dimc-net: Deep incomplete multi-view clustering network'. In: *Proceedings of the 28th ACM international conference on multimedia*. 2020, pp. 3753–3761.

[3] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. 1st. Cambridge, MA: MIT Press, 2016.

[4] Baochen Sun, Abhinav Shrivastava and Kate Saenko. 'Revisiting unreasonable effectiveness of data in deep learning era'. In: *Proceedings of the IEEE international conference on computer vision*. IEEE. 2017, pp. 843–852.

[5] Mateusz Buda, Atsuto Maki and Maciej A Mazurowski. 'A systematic study of the class imbalance problem in convolutional neural networks'. In: *Neural networks* 106 (2018), pp. 249–259.

[6] Sinno Jialin Pan and Qiang Yang. 'A survey on transfer learning'. In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[7] Corinne Shorten and Taghi M. Khoshgoftaar. 'A survey on image data augmentation for deep learning'. In: *Journal of Big Data* 6.1 (2019), p. 60. DOI: 10.1186/s40537-019-0197-0.

[8] American Psychiatric Association. *Diagnostic and Statistical Manual of Mental Disorders, 5th Edition: DSM-5*. American Psychiatric Association Publishing, 2013.

[9] Ernst Niedermeyer and Fernando Lopes da Silva. *Electroencephalography: Basic Principles, Clinical Applications, and Related Fields*. Lippincott Williams & Wilkins, 2005.

[10] Irina I Goncharova et al. 'Single-trial localization of evoked cortical activity using SAM (spatially adaptive minimum-norm) technique'. In: *Clinical Neurophysiology* 114.4 (2003), pp. 737–748.

[11] Andrius Vabalas et al. 'Deep neural networks, siamese networks, and n-shot learning technologies for diagnosing schizophrenia through detailed analysis of EEG signals'. In: *Journal of Medical Systems* 43.8 (2019), p. 252.

[12] Hyeonwoo Kang et al. 'Transfer learning for deep learning on small data in biomedical engineering'. In: *Computational Biology and Chemistry* 81 (2019), pp. 38–49.

[13] Chun-Hao Yeh et al. 'Data augmentation'. In: *Journal of Information Science and Engineering* 36.1 (2020), pp. 1–22.

[14] Subhrajit Roy. 'Epilepsy seizures show success through Siamese architectures'. In: *International Journal of Advanced Intelligence Paradigms* 11.2/3 (2018), pp. 204–215.

[15] Guan Li and Weiqing Li. 'Contrastive Learning for EEG-based Event-Related Potential Detection'. In: *Frontiers in Neuroscience* 12 (2018), p. 269.

[16] Muhammad Yasir Hafiz et al. 'Autism Spectrum Disorder'. In: *StatPearls [Internet]* 21 (2020).

[17] Matthias Kirschner et al. 'A few-shot approach to EEG classification achieved excellent performance'. In: *Journal of Neural Engineering* 18.2 (2021), p. 026021.

[18] A. Roy et al. 'Improved feature extraction using a novel loss function for accurate neurological diagnoses'. In: *Journal of Neuroscience* 38.10 (2018), pp. 2314–2325.

[19] Tyrone Carlisle Nowell. 'Detection and Quantification of Rot in Harvested Trees using Convolutional Neural Networks'. 30 ECTS. MA thesis. Trondheim, Norway: Faculty of Science and Technology, Master of Science in Data Science, May 2019.

[20] Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The Elements of Statistical Learning*. 2nd. New York, NY: Springer, 2009.

[21] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. 2nd. Cambridge, MA: MIT press, 2018.

[22] Michael I Jordan and Tom M Mitchell. 'Machine learning: Trends, perspectives, and prospects'. In: *Science* 349.6245 (2015), pp. 255–260. DOI: 10.1126/science.aaa8415.

[23] Stuart Jonathan Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ: Pearson Education, 2010.

[24] Warren S McCulloch and Walter Pitts. 'A logical calculus of the ideas immanent in nervous activity'. In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1007/BF02478259.

[25] F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957.

[26] Bernard Widrow and Marcian E Hoff. 'Adaptive switching circuits'. In: *IRE WESCON Convention Record Part IV*. IRE. 1960, pp. 96–104.

[27] David E Rumelhart, Geoffrey E Hinton and Ronald J Williams. 'Learning representations by back-propagating errors'. In: *Nature* 323.6088 (1986), pp. 533–536.

[28] Wikipedia contributors. *Artificial neural network — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Artificial_neural_network. [Online; accessed 2-May-2023]. 2023.

[29] Michael Nielsen. *Neural networks and deep learning: A textbook*. Determination press, 2015.

[30] Christopher M Bishop. *Pattern recognition and machine learning*. Vol. 4. Springer, 2006.

[31] Sepp Hochreiter and Jürgen Schmidhuber. 'Long short-term memory'. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[32] Paul J Werbos. 'Backpropagation through time: what it does and how to do it'. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[33] Yoshua Bengio, Patrice Simard and Paolo Frasconi. 'Learning long-term dependencies with gradient descent is difficult'. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.

[34] Felix A Gers, Jürgen Schmidhuber and Fred Cummins. 'Learning to forget: Continual prediction with LSTM'. In: *Neural Computation* 12.10 (2000), pp. 2451–2471.

[35] Sepp Hochreiter et al. 'Gradient flow in recurrent nets: the difficulty of learning long-term dependencies'. In: *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001, p. 15.

[36] Sepp Hochreiter and Jürgen Schmidhuber. 'Long short-term memory'. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[37] Klaus Greff et al. 'LSTM: A Search Space Odyssey'. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (2017), pp. 2222–2232.

[38] Zachary C Lipton, Jared Berkowitz and Charles Elkan. 'A Critical Review of Recurrent Neural Networks for Sequence Learning'. In: *arXiv preprint arXiv:1506.00019* (2015).

[39] Chris Olah. *Understanding LSTM Networks*. http : / / colah . github . io / posts / 2015 - 08 - Understanding-LSTMs/. Accessed: 2023-03-27. 2015.

[40] Razvan Pascanu, Tomas Mikolov and Yoshua Bengio. 'On the difficulty of training recurrent neural networks'. In: *International Conference on Machine Learning (ICML)*. JMLR. org. 2013, pp. 1310–1318.

[41] Yann LeCun et al. 'Gradient-based learning applied to document recognition'. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[42] Felix A Gers, Jürgen Schmidhuber and Fred Cummins. 'Learning to forget: Continual prediction with LSTM'. In: *Neural Computation* 12.10 (2002), pp. 2451–2471.

[43] Mike Schuster and Kuldip K Paliwal. 'Bidirectional recurrent neural networks'. In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[44] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. 'Neural machine translation by jointly learning to align and translate'. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2015. URL: https://arxiv.org/abs/1409.0473.

[45] Sebastian Ruder. 'An overview of gradient descent optimization algorithms'. In: *arXiv preprint arXiv:1609.04747* (2016).

[46] Sébastien Bubeck. 'Convex optimization: Algorithms and complexity'. In: *Foundations and Trends® in Machine Learning* 8.3-4 (2015), pp. 231–357.

[47] Léon Bottou. 'Large-scale machine learning with stochastic gradient descent'. In: *Proceedings of COMPSTAT'2010* (2010), pp. 177–186.

[48] Xavier Glorot and Yoshua Bengio. 'Understanding the Difficulty of Training Deep Feedforward Neural Networks'. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 249–256.

[49] John Duchi, Elad Hazan and Yoram Singer. 'Adaptive subgradient methods for online learning and stochastic optimization'. In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[50] Diederik P Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *arXiv preprint arXiv:1412.6980* (2014).

[51] Lutz Prechelt. 'Automatic early stopping using cross validation: quantifying the criteria'. In: *Neural Networks* 11.4 (1998), pp. 761–767.

[52] Yoshua Bengio. 'Practical recommendations for gradient-based training of deep architectures'. In: *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 2012, pp. 437–478.

[53] Boris Teodorovich Polyak. 'Some methods of speeding up the convergence of iteration methods'. In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17.

[54] Ilya Sutskever et al. 'On the importance of initialization and momentum in deep learning'. In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. JMLR Workshop and Conference Proceedings. 2013, pp. 1139–1147.

[55] Matthew D. Zeiler. 'ADADELTA: An Adaptive Learning Rate Method'. In: *arXiv preprint arXiv:1212.5701* (2012).

[56] Tijmen Tieleman and Geoffrey Hinton. 'RMSprop: Divide the gradient by a running average of its recent magnitude'. In: *Coursera lecture slides* 1.1 (2012), p. 26.

[57] Andrew Ng. 'Feature selection, L1 vs. L2 regularization, and rotational invariance'. In: *In Proceedings of the twenty-first international conference on Machine learning* (2004).

[58] Sergey Ioffe and Christian Szegedy. 'Batch normalization: Accelerating deep network training by reducing internal covariate shift'. In: *arXiv preprint arXiv:1502.03167* (2015).

[59] Timothy Dozat. 'Incorporating Nesterov momentum into Adam'. In: *4th International Conference on Learning Representations (ICLR)*. 2016.

[60] James Bergstra and Yoshua Bengio. 'Random search for hyper-parameter optimization'. In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[61] Mark Everingham et al. 'The pascal visual object classes (voc) challenge'. In: *International journal of computer vision*. Vol. 88. 2. Springer. 2010, pp. 303–338.

[62] Kishore Papineni et al. 'BLEU: a method for automatic evaluation of machine translation'. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[63] Foster Provost and Tom Fawcett. 'Glossary of terms used in machine learning'. In: *Proceedings of the 14th International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc. 1998, pp. 243–250.

[64] Tom Fawcett. 'An introduction to ROC analysis'. In: *Pattern Recognition Letters* 27.8 (2006), pp. 861–874.

[65] David M Powers. 'Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation'. In: *Journal of Machine Learning Technologies* 2.1 (2011), pp. 37–63.

[66] Cornelis J van Rijsbergen. 'Information retrieval'. In: (1979).

[67] Foster Provost and Tom Fawcett. 'Machine learning from imbalanced data sets 101'. In: *AAAI/IAAI* 1 (1998), pp. 1–15.

[68] Ajitesh Kumar. *Machine Learning with Limited Labeled Data*. https://vitalflux.com/machine-learning-with-limited-labeled-data/. Accessed: 2023-04-24. 2022.

[69] Elzbieta Olejarczyk and Wojciech Jernajczyk. *EEG in schizophrenia*. Version V1. 2017. DOI: 10.18150/repod.0107441. URL: https://doi.org/10.18150/repod.0107441.

[70] Hassan Aqeel Khan et al. 'The NMT Scalp EEG Dataset: An Open-Source Annotated Dataset of Healthy and Pathological EEG Recordings for Predictive Modeling'. In: *Frontiers in Neuroscience* 15 (2022). ISSN: 1662-453X. DOI: 10.3389/fnins.2021.755817. URL: https://www.frontiersin.org/article/10.3389/fnins.2021.755817.

[71] Elzbieta Olejarczyk and Wojciech Jernajczyk. 'Graph-based analysis of brain connectivity in schizophrenia'. In: *PloS one* 12.11 (2017), e0188629.

[72] Ron Kohavi. 'A study of cross-validation and bootstrap for accuracy estimation and model selection'. In: *Ijcai*. Vol. 14. 2. 1995, pp. 1137–1145.

[73] Sylvain Arlot and Alain Celisse. 'A survey of cross-validation procedures for model selection'. In: *Statistics surveys* 4 (2010), pp. 40–79.