Università degli Studi di Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Computer Science

Preliminary study on the use of neural networks for the real-time detection of LFP evoked by sensory stimuli

Master Thesis

Supervisor

Prof. Alessandro Sperduti

Master Candidate

Leonardo Amato

Academic Year 2022-2023

Abstract

To create a neural prosthesis, multiple studies on how the brain works are necessary. Stefano Vassaneli's NeuroChip Lab of the University of Padova studies how the brain responds to sensory stimulation. The laboratory carried out multiple experiments with stimulation of the whiskers of rats to study how *evoked potential* can be used for neural prostheses. My thesis work focuses on the analysis, implementation and comparison of various deep neural network techniques for the task of labeling brain recordings to detect evoked potentials.
The analysis focused on three types of networks: temporal convolutional networks (TCN), gated recurrent unit (GRU) networks, and a model that uses both techniques. To create these models, the data set was first created from recordings of the barrel cortex of rats. The input data are the local field potential (LFP) features while the output data are the labels of which segments contain evoked responses. The data set was enriched in the training phase by using a data augmentation layer.
I proceeded to perform a model selection process to extract the best hyperparameters from the models. The selected models were then compared to each other, to a set of baseline models, and to an ensemble technique. The temporal convolutional network was selected as the best in terms of efficacy. This model was the worst in terms of efficiency, so a magnitude pruning process was performed to reduce the number of parameters of the model. After pruning, the model is 10 times more efficient.
The test performed on full recordings of a test rat shows clear advantages in using deep learning techniques. The final model Temporal Convolutional Network is capable of identifying in real time up to 86% of the evoked responses, with high precision, low energy consumption, and a delay of just 80 ms.

Ringraziamenti

Vorrei innanzitutto ringraziare mia madre *Cristina*, da cui ho appreso creatività e generosità, mio padre *Gigi*, che da sempre mi ha insegnato ad avere pensiero critico, mio fratello *Francesco*, con cui condivido moltissimi interessi nonostante la diversa personalità, e mia nonna *Ivana*, che ogni giorno arriva in bici portandoci le sue storie, il suo ragù e il suo frico.

Un grande ringraziamento va a tutti i miei amici.
Agli amici di Cividale, *Sofia*, *Amy*, *Luca* e *Jacopo*, per la musica, le belle serate, i discorsi sui film, per avermi sopportato fino ad ora e per esserci sempre stati.
A *Davide* per essere un ottimo amico e per tutte le serate passate a parlare di scienza e storia.
A *Claudia* che ci ha portato la sua dolcezza dalla Spagna.
A *Solmaz*, con cui condivido il mio banale senso dell'umorismo, per essere l'unica persona che conosco ad amare i Doors quanto me e per essere sempre stata un'ottima amica.
A *Sepide*, che come me ha un profondo amore verso i gatti.
Alle le mie coinquiline *Noemi* e *Sabrina*, con cui vivere è un piacere.
Al resto dei miei amici conosciuti a Padova, *Marco*, *Angelica*, *Claudia*, *Benedetto* e *Aurora*, che hanno reso questa città un po' più casa.

Volevo ringraziare il Prof. *Alessandro Sperduti*, il Prof. *Stefano Vassanelli*, il dr. *Mattia Tambaro* e la dr.ssa *Claudia Cecchetto* per l'aiuto offerto e l'opportunità data.

Infine, il ringraziamento più importante va alla mia ragazza *Veronica*, per avermi migliorato la vita e aver reso ogni giorno più bello. Ogni ricordo dei momenti magnifici passati con lei riesce a farmi sorridere anche nei momenti più difficili.

Contents

Introduction

Prof. Stefano Vassanelli's 'Neurochip Lab' at the Department of Biomedical Sciences (DBS) of the University Of Padova studies the various technologies and approaches required to build a functional neural prosthesis, a system capable of reading and changing the behavior of a neural population through the use of electrical probes. An electrical probe is a needle-shaped array of electrodes, which can be intrusively introduced in the brain cortex to record the change in electrical potential. Central in the building of such a system is the study of the various functions of the brain, such as the response to a stimulation. All the analyses carried out by the Neurochip lab are done on brain recordings extracted from the barrel cortex of multiple rats, i.e. the section of the cortex that handles the inputs from the whiskers. Several studies conducted in the laboratory focus on the **potential evoked by the sensory stimulation** of the whiskers of rats. A potential evoked is a precise change in the electrical potential of the cortex and is recorded in response to an event or stimulation. In Figure 1, a general scheme of this recording process is illustrated.

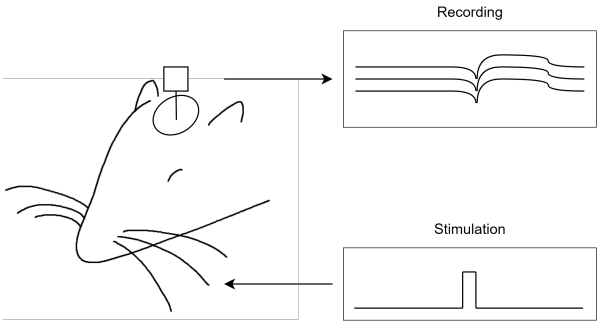


Figure 1: Scheme of the data acquisition process. A stimulation is sent to the whisker and neural signal and the response to the stimulation are recorded.

Two extracellular signals are used to analyze the response to whisker stimulation:

- The **local field potential** (LFP) is the change of potential recorded with the use of the probes in the range 1-100Hz. The LFP contains information about the synchronization of the activity of multiple neural population.
- The **multi-unit activity** (MUA) measures how many neurons touching the probe are firing at a given instant. The MUA is extracted from the high 300-6000Hz frequency range. It contains information about a single population of neurons and the codification used by them to communicate.

My work focuses on the detection of the patterns generated in response to the stimulation through the use of deep learning techniques, used on the LFP signal. The task of the program is to label the evoked potential in the recordings, as shown in Figure 2.

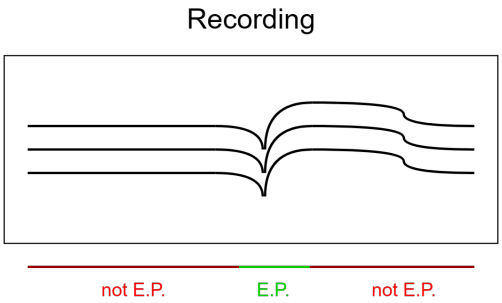


Figure 2: Example of evoked potential labelling in *EP* (evoked potential) and *not EP* (not evoked potential).

Deep learning is a branch of machine learning that focuses on the creation of deep neural networks. A deep neural network is a multilayered model capable of learning from a data set how to extract complex features. The reason why deep learning models generally perform better than shallow ones lies on the fact that each layer of the model has the ability to recognize a complex sub-feature from the output of the previous one. The model will learn the optimal pipeline of sub-feature extraction needed for the final output feature. Within this framework, a supervised learning approach was used, which requires the designer to specify the intended behavior that must be learned. In this case, the behavior the model must learn is the detection of the evoked response. Various deep learning models are currently being used to analyze brain signals, for example to predict emotions on electroencephalography data. However, there are no studies on the use of deep learning models that are capable of identifying the time window of the evoked potential in real time. To create a working model, two different analyses were performed. The first analysis, used as a baseline, was performed using linear classifiers on three types of features: the LFP signal, the time-frequency features extracted from the LFP, and the MUA signal.
The second analysis focused on the use of deep learning. The main types of techniques used are models built on **Gated Recurrent Unit**, **Temporal Convolutional Block**, and a mixture of the two.
I have performed all the analyses, pre-processing and designed the models using MATLAB 2022b.

Problem statement and Aim of the thesis

The NeuroChip Lab provided a set of recordings of rats anesthetized with urethane. A first problem arises when we want to create a data set that will be used in a machine learning context. It is important that the labels assigned to the data correspond to reality. If we label a time window as an "evoked response" when there is none, the model would learn the wrong behavior. In terms of functionality of the models other types of problems occur. The provided recordings contain multiple evoked potential

mixed with spontaneous activity of the brain. The main aim of the work is to develop a system capable of correctly identifying the evoked potential. Spontaneous activities of the brain are patterns of activity that occur in the absence of external stimuli. These patterns can be really similar to those of an evoked potential. For this reason, a simple system, such as the linear classifier, would struggle to discriminate between the two. In addition, a further issue involves the efficient identification of the evoked potential in real time. A system that requires a small amount of operation per second needs to be developed to be used on a chip with a low energy and short delay requirement.

Significance of the Study

My work focuses on sensory stimuli on whiskers. However, a system with the ability to detect evoked potential could have further uses in fields such as medicine, bio-engineering and neuroscience. The model I have built could be yet slightly modified to be used for detecting potential evoked by other events and stimulation, for example, the detection of a visual or auditory stimuli, or the detection of events internal to the brain. Currently, when conducting a study on evoked potentials, information about each stimulation, such as instant, duration, or intensity, must be recorded; otherwise, it would not be possible to find the evoked potentials. With a model capable of detecting evoked potentials without the need for stimuli information, new types of experiments could be carried out.
Furthermore, this type of model can be used in real-time closed-loop systems to restore or improve brain functionality. Close-loop is a term that describes all techniques where the state of the brain is iteratively analyzed and changed in an autonomous system. These systems are often used as an alternative or in support of drug treatment for disorders such as Parkinson's disease or epilepsy.

Methodology

Regarding the methods employed to create the final model, I followed four main steps: the creation of the data set, the selection and creation of the models, the comparison between the models, and the optimization of the best one. At first, after defining and extracting the main features from the recordings, the best recordings were selected and those where the evoked response is too weak or not present were removed. Currently, this is done visually with a manual selection. This process was slightly improved by analyzing and comparing the average response of each recording.
Furthermore, a set of linear models was selected. These models were used as baseline to analyze the improvement gained by the use of deep learning techniques.
To design the structure of deep learning models, a brief analysis of how similar models are built for similar issues was performed. The four deep neural network architectures that were analyzed and implemented are: a **Gated Recurrent Unit network**, a **Temporal Convolutional Network**, a **Recurrent-Convolutional mixed network**, which uses both GRU and convolutional layers, and an **ensemble** of the three. Taking this into consideration, a model selection phase was then performed, where I defined a set of possible networks, one for each architecture with the exception of the ensemble. By means of the information extracted from a random search, the four models were constructed. This search was carried out on data from all rats except one, which was used to test and compare.
Furthermore, the best deep learning model was selected and then optimized with a

magnitude pruning algorithm. Pruning is a process in which the least important parts of a neural network are removed, leaving only the fundamental sections. The use of pruning greatly reduced the amount of computation required by the final network. The work was concluded with a cross-validation of the models to give a better overview on how they perform in recordings of different rats.

Structure of the thesis

Chapter 1, *Theoretical background*, contains all the knowledge that I consider fundamental for a clear understanding of my work. Information about how neurons, barrel cortex of the rat, local field potential, and multi-unit activity is provided in the first section. The second section includes a brief explanation of the main concepts of deep learning, such as neural networks, learning processes, convolutional networks, and recurrent networks. Finally, the definitions of the metrics of comparison and evaluation are given.

Chapter 2, *Data acquisition, pre-processing and analysis*, focuses on the data provided by the NeuroChip lab and the creation of the final data set, which will then be used for training. The process followed by the lab to record the data and their content is briefly described. The following sections are centered on the definition of the various features, such as LFP and MUA, and how they were extracted and preprocessed. In addition, a brief analysis of the evoked response and the average response to stimulation is given. This is followed by a brief explanation of how the best recordings are extracted and how the data were labeled. Finally, a description of how the data were formatted in order for them to be used by the models is provided.

Chapter 3, *Methodology and results*, presents how the various models used for the detection of evoked responses were created and compared. The first section regards the preliminary analysis required to build a functioning model. The second section focuses on how model selection was performed, providing the results of the selection process. In the third part, the various selected models are presented. In the fourth section, the performance of the models on the test set are discussed and compared. In the fifth section, the optimization of the best-performing model is discussed. Finally, a final general comparison is made on all models with a cross-validation technique.

The *Conclusion* provides a brief summary of the completed work and the results obtained. In addiction, further possible improvements of the work are mentioned, as well as an account on the significance of the results for future researches in the field of neural prosthesis.

Chapter 1

Theoretical background

In this chapter, a brief introduction is presented to all the topics required to fully comprehend the work. The various topics are organized in three sections. The first one will provide an explanation on the origin of the data, i.e. which cerebral process generates the data used in this work and what is the information contained in it. In the second section, an introduction will be given on the concept of machine learning and deep learning, as well as an explanation of the main types of layer used in the models. In the last section, the various metrics that were used for comparing the performance of those models analyzed in this thesis are discussed.

1.1 Neuron, barrel cortex and brain signals

The data has been recorded from the extracellular space of the barrel cortex. To understand what it contains we first need to understand what is a neuron, what is an action potential, how action potential propagate in the extracellular space, what is the barrel cortex and what are the local field potential and the multi unit activity.

1.1.1 Neuron

A neuron is a specialized cell that transmits electrical and chemical signals. It works as the building block of the nervous system [1]. Its components, shown in Figure 1.1, are the cell body, also called *soma*, where the nucleus is found and where most logical "decisions" are made; the *dendrites*, which are the inputs of the cell; the *axon*, which functions as the output of the cell. The most important section of the axon is the *terminal*, which connects the neuron to other dendrites, allowing it to send signals to other neurons. A single neuron has little computational capabilities, but when located in a more complex network, the neural population is able to solve more complex tasks.

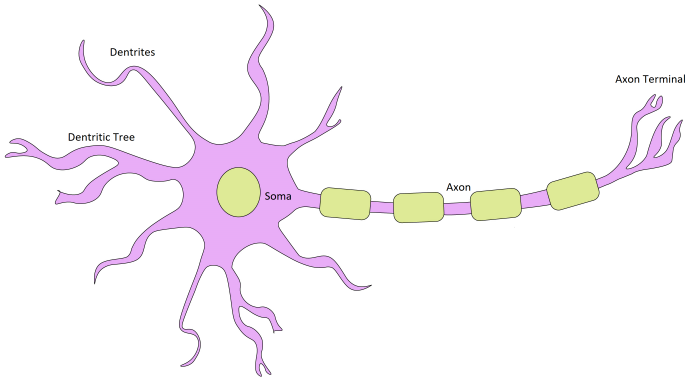


Figure 1.1: Scheme of the neuron and its main components.

To understand how the neuron is able to process information, we need to focus on its membrane. The inside and outside of the cell membrane contain different concentrations of sodium ions, positively charged and mostly present outside, and potassium ions, negatively charged and mostly present inside. Due to the different concentrations in ions, when the neuron is at rest, we see a -70mv difference of potential, or voltage, between the extracellular and intracellular space. If ions could cross the membrane, they would balance the voltage at 0. The membrane is full of gates, also called channels, with this exact function. These gates have the ability to allow potassium or sodium ions to flow to the other side of the membrane. The voltage-gated channels open when a voltage threshold is reached. An action potential (AP) is a chain reaction that is generated by these gates when a threshold of -50 mv is exceeded. AP propagates through the membrane in one direction, from *soma* to *axon* and in some cases to *dendrites*. After the action potential, there is a rebound phase that reduces the probability that another action potential occurs. [1][2][3]
In this thesis, I will also refer to the action potential as *spike* and to the event of an action potential occurring as *neuron firing*. An action potential is shown in Figure 1.2.

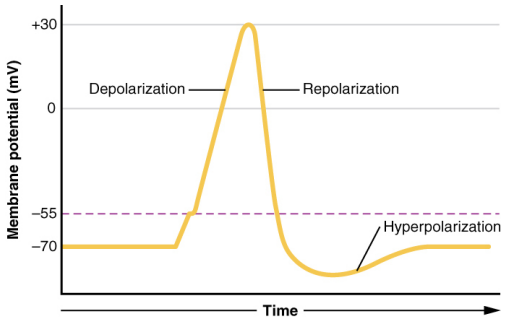


Figure 1.2: Phases of an action potential.[2]

The *synapse* is the connection between an axon terminal and the *dendrite* or the soma of the other cell. If the synapse is inhibitory, when the action potential reaches the terminal, it sends a chemical signal to the other cell, which activates some gates that lead to a negative change in potential. This negative change in potential reduces

the probability that a subsequent action potential of the other neuron occurs. If the synapse is excitatory, the chemical signal sent to the other cell causes the activation of a different type of gate, leading to an increase in potential, which increases the probability or directly causes a new action potential in the other neurons. [1][2]

1.1.2 The rat barrel cortex

The data used in this work have been extracted from the cerebral cortex of rats, more precisely, the barrel cortex, a region of the somatosensory cortex. Somatosensory processing handles information coming from a subset of body sensors such as touch, pain, and temperature. The barrel cortex handles the whisker stimulation signal. A "barrel" is present for each of the whiskers, as shown in Figure 1.3.

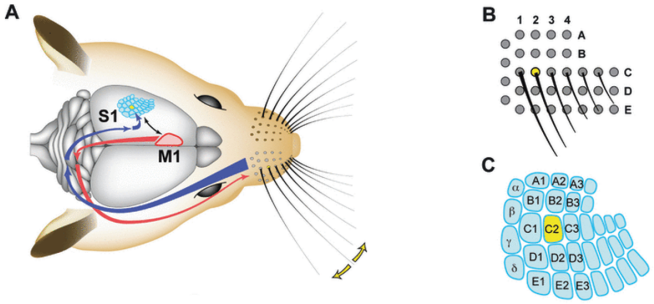


Figure 1.3: A) shows the position of the barrel cortex. B) shows the various whiskers and their position. C) shows the position of each "barrel" in the barrel cortex. Each barrel is connected to a relative whisker.[4]

Barrel cortex has six layers, ranging from the most superficial layer 1 to the deepest layer 6. The barrels are found at layer 4 and each is connected to one whisker; whereas other layers can receive information from multiple whiskers. The probe is inserted directly inside one or more barrels and is capable of recording information from all 6 layers. There are two main pathways that the stimulus information travels on. The first reaches Layer 4, the second reaches layer 5. These two layers are where the strongest evoked potential is recorded. The barrel cortex is able to process information about deflection of the whiskers. [5]

1.1.3 Local field potential

The local field potential (LFP) is the electric potential recorded in the extracellular space of the cerebral cortex. It measures the activity of multiple neural populations firing synchronously. An example of LFP signal is shown in Figure 1.4. These signals do not contain any information about a single neuron's firing, due to two factors. Firstly, if multiple neurons do not fire simultaneously, the values of the action potentials average out to 0. Secondly, even if there were no other interference, the action potential of a single neuron would be too weak to be detected from afar, since it would not be able to propagate through the extracellular without being mitigated. Therefore LFP signals are only used to analyze the neural activity of neural populations, rather than a single neuron.

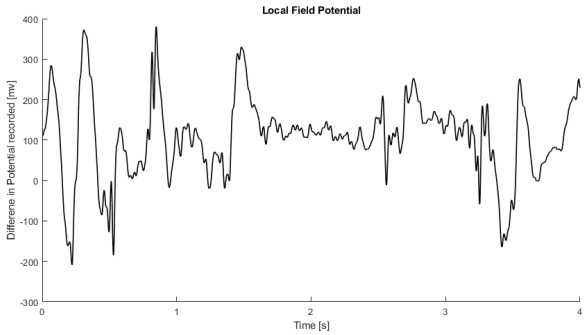


Figure 1.4: Example of LFP recording.

The most important features that can be extracted from a LFP signal are the frequency of the various oscillations. The intensity of the different oscillations gives us various information about the type of task that the cortex is performing. Five frequency bands are the most important in this context: Delta component at 0-4Hz, Theta at 4-8Hz, Alpha at 8-12Hz, Beta at 12-30Hz, and Gamma at 30-100Hz. The **evoked potentials** (EP) are electrical potentials that can be recorded in the LFP and are produced by the nervous system in response to an external stimulus. Evoked potentials differ in shape depending on the type of stimulation. There are three main types of study around EP: visual evoked potential (VEP), EP generated in response to visual stimulation, auditory evoked potential (AEP), generated in response to auditory stimuli, and somatosensory evoked potential (SSEP), generated in response to touch, pain, and temperature stimuli. In the barrel cortex, we can only record the somatosensory potential evoked by the whisker stimulation.

1.1.4 Multi unit activity

Multi-unit activity (MUA) is a measure connected to the number of neurons firing in a neural population at a given time[6]. It can be recorded with probes with a high sampling rate and low noise-to-signal ratio. Whenever a neuron touching the probe microelectrode fires, we can record an action potential. By selecting only the negative peak of the action potentials, we are able to create a MUA signal. An example of MUA recording is shown in Figure 1.5. These peaks are found in the 300-1000Hz range. A spike train is the sequence of neuronal firing. The temporal pattern of a spike train encodes information in various ways. In addition to firing rates, the temporal pattern of spike timings also carries important information about brain functions. For this reason, when preprocessing these data, it is important to leave both the spatial information and the fire-rate information intact.

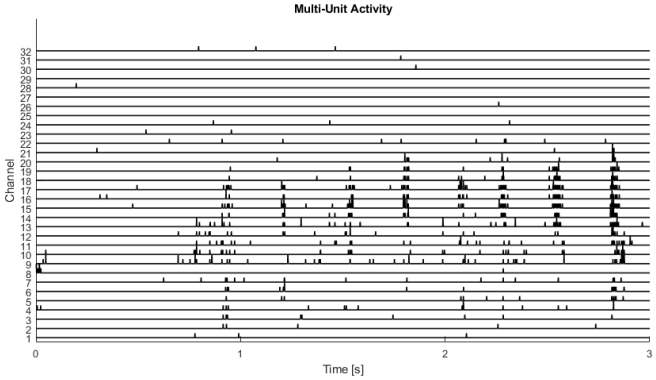


Figure 1.5: Example of MUA recording with multiple spike trains.

1.2 Machine Learning and Deep Learning

Machine Learning (ML), a subset of artificial intelligence (AI), studies how an algorithm can learn a behavior or information from data. Given an ML task, we call *model* an algorithm that has been trained on data to solve it. The field of Deep Learning (DL) is a subset of machine learning centered around nerual network models with multiple layers, each one containing multiple nodes. Each node is able to extract a feature from the output of the previous layer to be applied as an input for the next layer. We can look at the model as a pipeline where the input is progressively transformed at each layer in order to get the important features to solve the task. Deep learning uses nodes that mimic the functioning of the brain. The simpler node is the artificial neuron, a function that simulates the properties of the action potential of a neural cell. More complex nodes have been built over time, inspired by the function of a neural population. This section will mostly focus on the ones that were used in this work: **convolutional** and **recurrent** nodes. Convolutional layers uses array of filters, learned from the data, to compute complex feature from local portions of the input. For example, a convolutional layer is able to perform multiple types of edge detection; a following layer could then use this edges features to find more complex information about the structure of an image or of a signal. A recurrent node is a type of node built for causal type of data, such as time sequence or text. To understand the context of the sequence recurrent unit uses a memory block. Each time a new input is sent to the unit, the node computes a new memory state, starting from the current input and the previous memory state.

1.2.1 Supervised Learning

Supervised learning is a machine learning paradigm. In this paradigm data data is a set of pairs of input and output features. The input can be, for example, an image, a time sequence or a set of values. The output can be of the same type of the input, for example in image-to-image or **sequence-to-sequence** problems, otherwise it can have a different representation, for example, in image-to-class problems. We call "oracle" the function that maps the input X to the output Y. We want to create a function $h(X)$ that can approximate the oracle function that created the original data set. The oracle can be seen as a series of effects that, starting from the state of the environment

described as X, creates a measurable outcome that we can describe as Y. The field of deep learning tries to create models capable of imitating these pipelines of effect in order to predict the data Y starting only from X. In the context of this work, I analyzed an isomorphic sequence-to-sequence problem, where input and output have the same length. The input are features extracted from the raw signal of the brain recording and the output is a signal that labels the are evoked potential.

1.2.2 Artificial Neuron, neural network and training process

As previously mentioned, an artificial neuron is the simplest unit in the field of neural network. The artificial neuron has two sub-functions, the linear function and the activation faction. The linear function linearly transform the input vector x by applying a dot product with the weight vector w of the neuron. To the result of the dot product a bias term b is added.

$$z = x \cdot w + b$$

The activation function, usually a non linear function, is applied on the output of the dot product.

$$y = f(z)$$

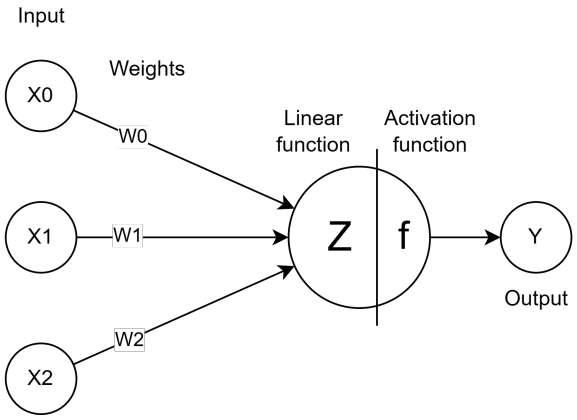


Figure 1.6: Scheme of artificial neuron.

Multiple neurons can be combined in a dense layer, the most common layer of a neural network. The reason why it is called dense, or fully connected, is given by the fact that each output of the layer has a weight that connects it to each input of the layer, as shown in Figure 1.7. While a single neuron computes a single value, a dense layer is capable of computing a vector of values.

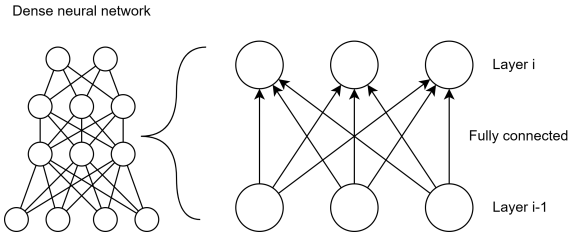


Figure 1.7: Scheme of a fully connected neural network.

Training process

Training a neural network is a optimization problem. In this process, we adjust the weights and biases to minimize a cost function J. This cost function must be differentiable and it must measure the difference between the output predicted by the model and the target output. The **backpropagation** algorithm takes the output of the cost function and computes the weight gradients [7]. The gradient is the vector of the partial derivatives $\frac{\partial J}{\partial w}$. We use the chain rule of the derivative to compute this value. The chain rule expresses the derivative of the composition of two differentiable functions in terms of derivatives of the two functions. So given f and g such that

$$h = f(x) \quad y = g(h) = g(f(x))$$

we can compute

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial x}$$

Since the cost function is a composition of multiple differentiable function, we can use this chain rule to compute its derivative.
Now we can use the gradient to find the direction of adjustment to minimize J. **Gradient descent** is an optimization algorithm that is used to minimize the cost function. This algorithm forwards the input data through the network in the *forward-step* and computes the cost function. Then, backpropagation is used to compute the adjustment of the weights and biases in the *backward-step*. These two steps are repeated until the algorithm converges to a solution. At each iteration, the weights are changed with the following formula:

$$w_t = w_{t-1} - \alpha \cdot \frac{\partial J}{\partial w_{t-1}}$$

where t is the iteration, w_t is the updated weights, w_{t-1} is the weight value in the previous iteration and α is the learning rate. If α is too big, there is a high risk of overshooting and missing the global minimum (the set of weights that return the minimum cost) of J. While if α is too small, there is a high risk of local minimum convergence. In this work, an alternative to gradient descent is used: **Adaptive Moment Estimation** (ADAM) optimization [8]. The most important thing to understand about ADAM is how it improves the training process. The method computes individual adaptive learning rates for each parameter from estimates of the first and second moments of the gradients. This makes training more stable and faster. [9]

Activation functions

If we stack multiple dense layers with linear activation function, we end up with a linear model, since the combination of linear function is itself a linear function. The

complexity of the model in this case would not scale with a deeper model. To give the model more expressive power, we use non-linear activation functions. A *non-linear activation* function has the ability to add non-linearity to the model by using non-linear transformation of the output of the neuron. The simplest non-linear activation is the sign function.

$$Sign(x) = \begin{cases} -1 & \text{for} \quad x \leq 0 \\ 1 & \text{for} \quad x > 0 \end{cases}$$

Sign activation has a major problem. The derivative is 0 which makes it unusable in backpropagation, since it would make the gradient of each weight equal to 0 due to the chain rule.
The **Rectified Linear Unit** (ReLU) is the most common non-linear activation function used in the hidden layers of neural networks.

$$ReLU(x) = \begin{cases} 0 & \text{for} \quad x \leq 0 \\ x & \text{for} \quad x > 0 \end{cases}$$

The derivative of ReLU for negative input is 0, while for positive output is 1. For this reason, when using ReLU, the weights of the neurons are only when the output is positive. This solves the problem of the sign function. However, this would mean that most of the weights during a training iteration would not change. Another major problem is the dying ReLU. If all pre-activation values are negative, then we would not be able to update the weights of the node. This would make the neuron always return 0. [10]
To fix these problems, alternatives to ReLU have been developed, where negative values are mitigated but not set to 0. One of these alternative is the **Exponential Linear Unit** (ELU):

$$ELU(x) = \begin{cases} \alpha\left(e^x - 1\right) & \text{for} \quad x \leq 0 \\ x & \text{for} \quad x > 0 \end{cases}$$

In a binary classification context, we have targets that are either 0 or 1. To train a model on these data, we require a final activation that returns a value in the range of (01). For this reason, we use the **Sigmoid** function:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid always returns a value in the range of 0-1 and is differentiable. Sigmoid is sometimes used in hidden layers and to create gate-like connections. All of the models analyzed in this thesis use Sigmoid in the output layer. [10]

Underfitting and Overfitting

Two main problems can arise when learning a model:

- Under-fitting happens when the model is too simple and it is not able to learn the complexity of the task.
- Overfitting occurs when the model is too complex and is not able to generalize what is being learned. In this case, the model would perform optimally on the data with which it was trained and poorly with new examples (test and validation data).

A third problem that may arise is the presence of errors in the data set. If the labels in a classification problem are wrongly placed, then the model would try to learn the wrong behaviours.
The optimal way to build a model is to use a complex one and regularize its behavior. Regularization is the set of techniques that are used to prevent a complex model from overfitting. We can either increase the size of the data set or put some constraints on the weights. In my work, two regularization techniques are used: **data augmentation**, a technique that virtually increases the size of the data set by applying small transformation on the inputs, and **dropout**, which randomly sets some of the output of hidden layer to 0 in order to imitate an ensemble of simpler models. [11]

Exploding and vanishing gradients

The vanishing gradient problem and the exploding gradient problem are two issues that can arise when training neural networks with gradient-based learning methods. The problem of vanishing gradients occurs when gradients become too small during backpropagation, making it difficult for the network to learn. The exploding gradient problem occurs when gradients become too large during back-propagation, which can cause the weights to update too much and make the network unstable. Both problems can make it difficult for a neural network to learn and converge on a solution. There are several ways to fix the problems of vanishing and exploding gradients in neural networks. One way to fix the exploding gradient is to use Gradient Clipping [12]. In this technique, we introduce a *threshold* value. Whenever the norm of the gradient $||g||$ exceeds this threshold, the gradients is multiplied by the factor $\frac{Threshold}{||g||}$. The threshold must be chosen carefully; If it is too high, we would not fix the exploding gradient, and if it is too low, we would slow down the learning. For fixing the gradient vanishing problem we could use a better random initialization or residual connection.

Normalization Layer

The output of each node of the network has a given distribution. During the training these distributions keep shifting and changing. This creates two problems: the training slows down since a smaller learning rate is needed to overcome this shift in distribution; If the pre-activation value is always negative, and we use activation like ReLU or ELU, smaller changes are made, since the gradient is either 0 or really small. To fix this, normalization layers are added in the pre-activation of each neuron. Normalization can be computed on different dimensions, such as normalizing the output of all layers (layer normalization), normalizing the output of each layer in a batch of data (batch normalization), or normalizing the output of each layer for any given input (instance normalization).[13][14][15]

1.2.3 Convolutional Neural Network

A convolutional neural network (CNN) is a deep learning neural network designed to process structured arrays of data such as images or sequences. They use a mathematical operation called *convolution* on at least one of their layers. They have three main types of layers, which are the convolutional layer, the pooling layer, and the fully connected layer. For this context, we will focus on 1-dimensional CNN. In a 1D CNN, the input is a one-dimensional signal, and the convolution operation is performed over the time dimension. The output of the convolution operation is then passed through

a non-linear activation function such as ReLU. The output of this layer is then passed to another layer for further processing.[16]
Given a signal channel, a *kernel* is a small vector that slides over the input data and performs the dot product with each segment of the input. A *filter* is a set of kernels, one for each input channel. The output of the kernels convolutions are summed together to create the output of the filter. This process is schematized in Figure 1.8.

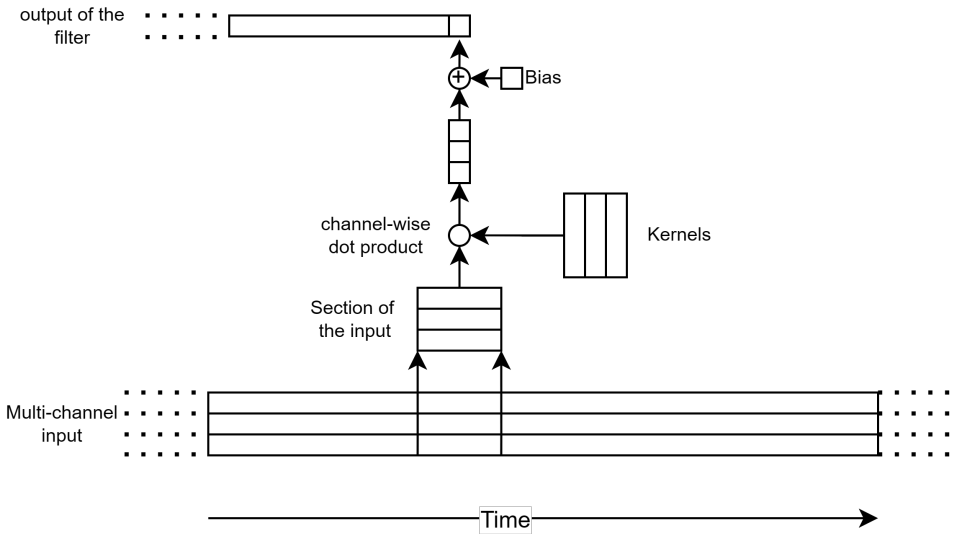


Figure 1.8: Example of convolutional filter on a multi channel sequence input.

A moving maximum window is often added after the layer. This function, also called *max pooling*, is used as a down-sampling technique, to reduce the dimensionality of the data while at the same time maintaining the most important information, the maximum activation. The main advantage of using max pooling is the translation invariance, since it removes little variation of local portion of the data, while maintaining the most important values. In the context of this task, we do not down-sample the data so we can use max pooling for the translation invariance and as a further non-linear activation.
The *receptive field* of a convolutional model is the window of data that the model is able to analyze in order to predict a point. To allow the network to better understand the context of a point in the signal, we would require a larger receptive field, but this would also increase the number of parameters. In this thesis, a specific type of convolutional network was used, a **Temporal Convolutional Network** (TCN) [17]. A TCN has the following characteristics:

- Uses 1d convolutional filters over the time dimension;
- Is causal, can only use present and past information;
- Is a sequence to sequence model, where the size of the output is the same size as the input;
- Uses a dilation factor in deeper layer;
- Can uses skip connection between layers.

This model has many advantages compared to other architectures and simpler convolution. The dilation factor, which adds space in-between the kernel's weights during the convolution, lets us increase the receptive field without increasing the number of parameters. An example of how the receptive field is increased is shown in Figure 1.9. Each arrow represents a parameter.

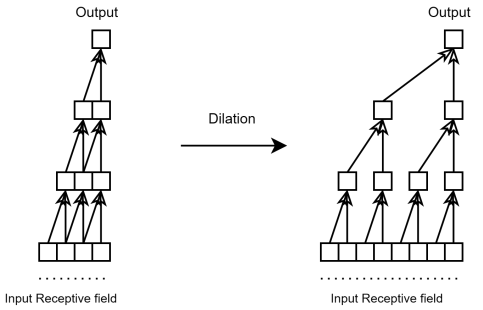


Figure 1.9: The 3 layers have the same number of parameters but different receptive fields.

The use of skip connection let us have a more stable gradient descent and faster training. The skip connection works by adding a linear transformation of the layer input to the layer output, as shown in Figure 1.10. This operation preserves the gradient in deeper layers.

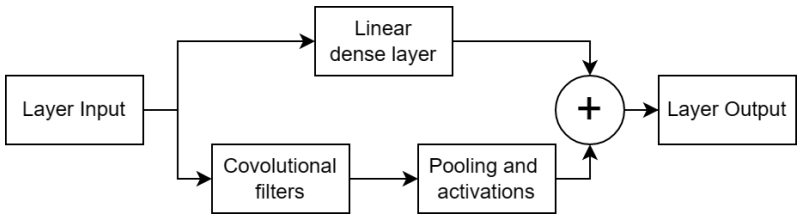


Figure 1.10: Scheme on how a skip connection can be created using a dense layer.

1.2.4 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a type of neural network that can process sequential data, such as time series data or natural language text. They are called recurrent because they perform the same task for every element of a sequence, with the output dependent on the previous computations. The key feature of RNNs is their ability to maintain an internal state or memory of past inputs, which allows them to capture temporal dependencies in sequential data.
A Recurrent Unit (RU) is a building block of an RNN that processes one element of a sequence at a time and updates its internal state based on the current input and its previous state. The internal state is then used to compute the output for that element of the sequence.

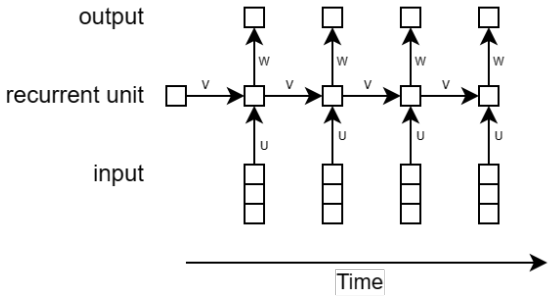


Figure 1.11: Example of Recurrent Neural Network.

The simplest RU is a fully connected network. Three different sets of weights are used by this unit, two used to combine the previous memory state and the current input to generate the new memory state

$$H_t = f(V \cdot H_{t-1} + U \cdot X_t + B_h)$$

and one used to compute the current output given the new memory state

$$Y_t = f(W \cdot H_t + B_y)$$

where f is the activation of the network. The process is schematized in Figure 1.11. The simple recurrent model suffers from both exploding and vanishing gradient. The model either cannot forget useless information that is propagated through time in the memory or cannot hold in memory the useful information without dissipating it with all the matrix multiplication. To fix this problem, gates capable of updating and resetting memory have been introduced. Gates are neural networks that can output values in the range of 0-1 and can be multiplied to other values to control the information flow.
The most common type of RU that uses gates is the **Long Short-Term Memory** (LSTM) unit, which has been shown to be effective at capturing long-term dependencies in sequential data [18]. The LSTM unit, shown in Figure 1.12, uses three gates, the *input gate*, the *output gate* and the *forget gate* and uses an additional memory block called *cell*. The input gate determines how much new information should be allowed into the cell state. The forget gate determines how much information should be discarded from the cell state. The output gate determines how much information the unit should output from the cell state to the rest of the network.

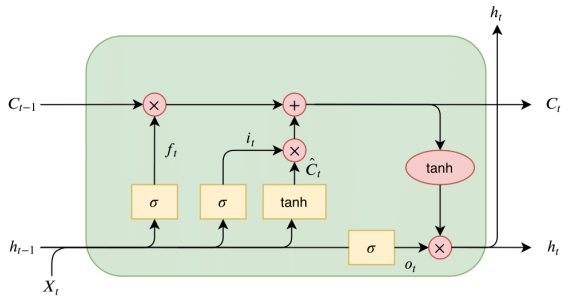


Figure 1.12: Scheme of a LSTM unit.

Another popular type of RU is the Gated Recurrent Unit (GRU) [19], shown in Figure 1.13, which has fewer parameters than LSTM and has been shown to be effective at capturing short-term dependencies. The GRU layer uses only two gates, the reset gate and the update gate. The reset gate handles how much information flows outside of the memory while the update gate handles how much information flows inside the memory.

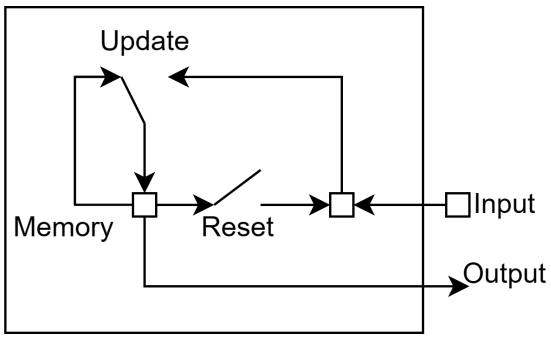


Figure 1.13: Scheme of a GRU unit.

1.3 Classification metrics

To analyze the classification ability of models, various formulas and metrics were used. In this section, these metrics are explained. Let's take for example a target vector of 0 and 1 that we want to generate starting from out data. Our classifier generates a vector of 0 and 1 of the same length and we want to measure how good this classification was. We start by computing the number of true positive **TP**, false positive **FP**, True Negative **TN**, False negative **FN**.

- TP: is the number of points where both the target and the predicted vector are at 1
- TN: is the number of points where both the target and the predicted vector are at 0
- FP: is the number of points where the target is a 0 and the predicted is at 1
- FN: is the number of points where the target is a 1 and the predicted is at 0

Starting from these values, we can compute the following metrics:
$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
$Precision = \frac{TP}{TP+FP}$
$Recall = \frac{TP}{TP+FN}$
$F1 - score = 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$
If the output of the classifier is not a vector of 0 and 1 but a real value vector in which we should set a threshold to obtain the two classes, 1 for values higher than the threshold and 0 for values below. Now all the performance that we just measured would change with the change in threshold, as shown in Figure 1.14. What we do now is to move threshold starting from the minimum value to the maximum. While moving it we keep track of the precision and the recall values. If we now plot all the

values keeping the recall on the Y axis and the precision on the X axis, we get a recall precision curve (not to be confused with the ROC curve), as shown in Figure 1.15.

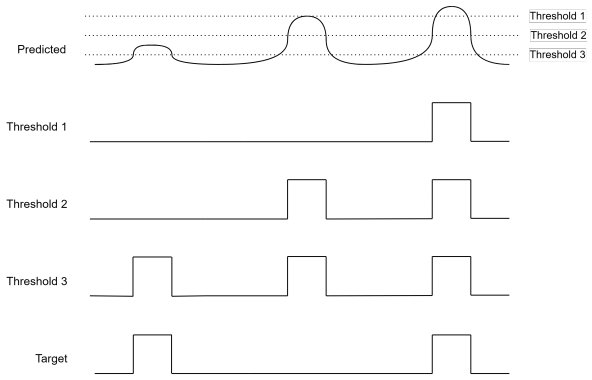


Figure 1.14: Example on how we can change the classes by using three different thresholds.

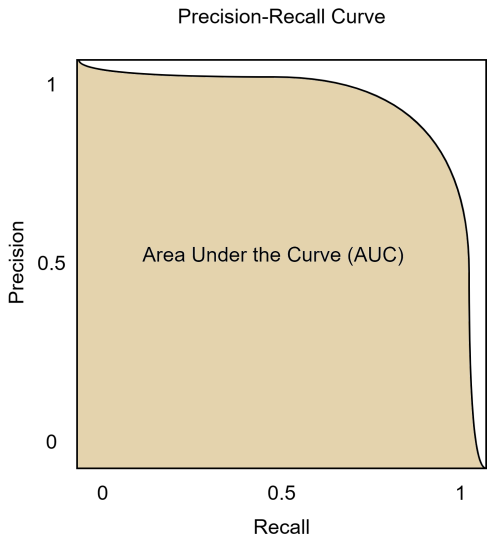


Figure 1.15: Plot of recall and precision change when moving the threshold. In orange, we can see the area under the curve.

If we compute the Area Under the Curve of this precision-recall curve, we have now a good metric on the ability of the classification to predict the target. The *precision-recall curve* is an optimal method of analysis, especially when the true target class is rare.[20]
A different curve, called the *Receiver Operating Characteristic* (ROC), is usually used for binary classification evaluation. However, this curve is not well suited on a strongly imbalanced data set.
In this thesis I will often refer to the maximum F1 score obtained by moving the threshold while computing the precision-recall curve as the *max F1*. The relative precision and recall used to compute this F1 will also be presented.

Chapter 2

Data acquisition, pre-processing and analysis

This chapter focuses on the analysis of the data and the creation of the final data set. First, the data acquisition method and an analysis of the recorded data are briefly described. This is followed by an analysis of the features that are required to solve the evoked potential detection task. Furthermore, all the pre-processing steps required to create the final data set are presented.

2.1 Data acquisition

The data has been extracted by the Neurochip lab of the University of Padova. The rats involved in the experiments are all young adults of both genders, ranging from an age of 25-35 days and a body weight of 90-120g. The rats are anesthetized with an intra-peritoneal induction dose of Urethane (0.15/100 g body weight) followed by a single additional dose (0.015/100g body weight). Each animal is positioned on a stereotactic instrument and the head is fixed by teeth and ear bars. A window on the skull is drilled over the position of the somatosensory barrel cortex. A single shank probe is inserted in the middle of the window on the barrel cortex. The shape of the cortex may vary between different rats, so it is not possible to always record from the same barrel. For this reason in an initial setup each whisker is stimulated and the response is analyzed. The whisker that returns the clearest and strongest evoked potential will be the one stimulated during the experiment.

2.2 Raw data structure

The data consists in total of 48 recordings, each of which is 5 minutes long, extracted from 25 different rats. The recording contains two vectors: brain signal and stimulation signal. **Brain signal**: A 32 channel matrix of 300 seconds of the electrodes signals at 25000Hz sampling rate. From this recording it's possible to extract both the LFP and the MUA signals. A sample from this signal is shown in Figure 2.1.

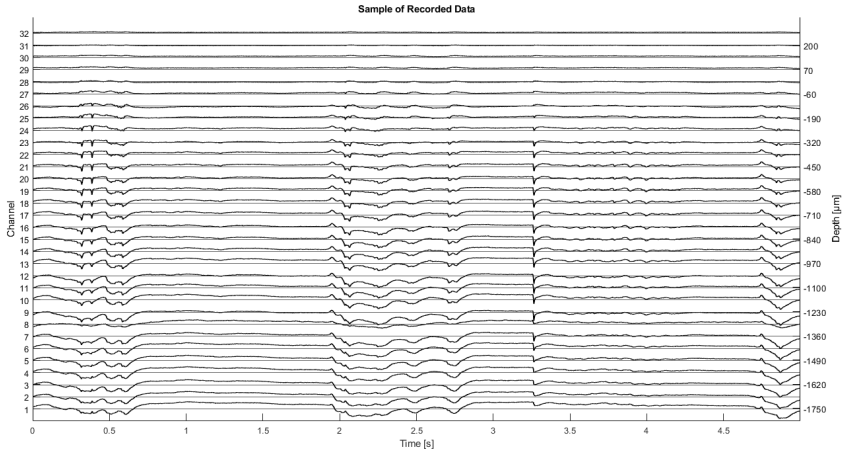


Figure 2.1: Section extracted from a recording. Each line represents a different channel.

Stimulation signal: A vector of 300 seconds at 25000 Hz sampling rate, recording the impulse sent to the stimulation machine. This vector contains mostly values at 0 mV, with a peak at 3 mV lasting for 1 ms each time a stimulation is given to the whiskers. A sample of this signal is shown in Figure 2.2.

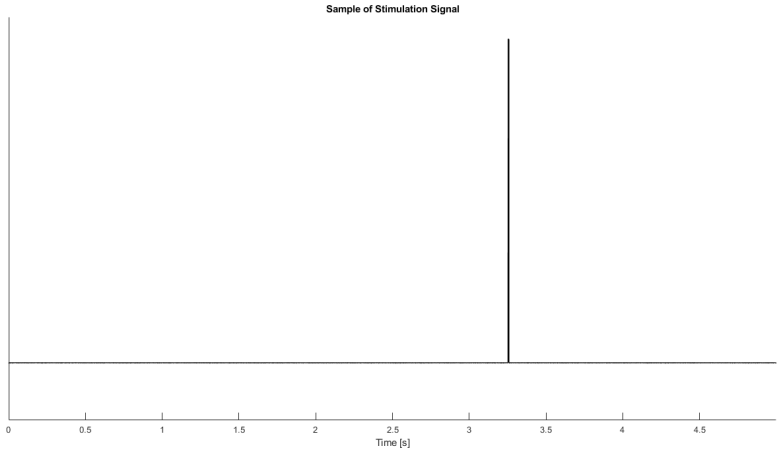


Figure 2.2: Section of the stimulation signal.

The data set is divided into three parts, respectively, recorded in 2021, 2022 and 2023. The position of the probe has been moved up by one channel in 2022, therefore, the channels in the recordings are not all aligned. Additionally, the probe has been changed in 2022; consequently, different types of artifact may be present for each year.

2.2.1 Spectrum of the channels

With the use of the fast Fourier transform, we can analyze the brain signal to detect artifacts and defective electrodes. In Figure 2.3, we can observe the presence of a faulty electrode (channel 8) and a component at around 5000 and 10000 Hz. This could be due to some noise generated by other electrical equipment present in the laboratory. We can see that most of the LFP is in the range of 0-100Hz. For the MUA signal, there is some variance between each recording, so it is not possible to visualize that component on the spectrogram.

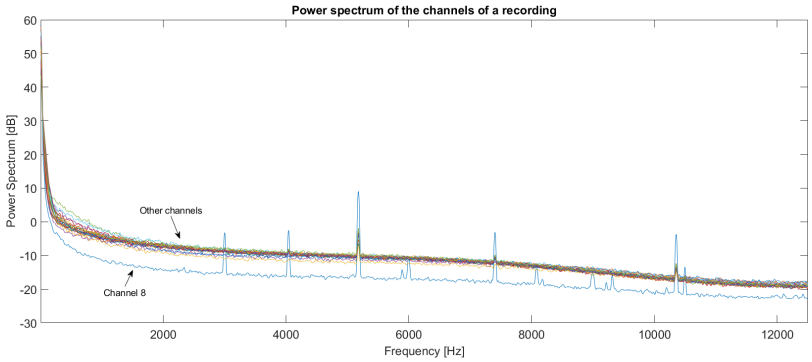


Figure 2.3: Spectrum of the channels from a 2021 recording. Channel 8, in light blue, is full of artifacts and is different from other channels in terms of magnitude.

2.3 Analysis of neural signal features

Given the *evoked potential detection* task, we need to define the main features required to solve it. A brief review of typical neural signal analysis methods was performed. The aim of this analysis was to collect the main set of features that can be used as input for the models. In this analysis, I was able to select the three types of feature that contain most of the information on responses to stimuli.

Time domain of Local Field Potential: the choice of this feature is directly correlated with the task. LFP in its raw form contains the unaltered shape of the evoked potential. The local field potential is the main feature that was analyzed. In this feature, we can use various techniques of evoked potential analysis, such as *event related potential*, a technique in which the average potential in response to a stimulation is computed. [21]

Time-frequency domain of Local Field Potential: The main problem of the time domain of the LFP is that different types of activity are mixed together. The evoked potential is not the only potential pattern contained in the signal, but those related to other events and spontaneous activity are also recorded. The response and

the rest of the activity may vary in terms of frequency. By splitting the LFP into multiple frequency bands, we could be able to extract only the frequency feature that we are interested in and filter the rest. Two alternatives were selected to analyze the time-frequency domain of the LFP: the simple division in the main frequency bands Theta, Alpha, Beta, and Gamma, and the more complex use of the Continuous Wavelet Transform (CWT). The time-frequency components of LFP are widely used to analyze the neuron population and brain behavior in different contexts.[22][23][24]

Multi Unit Activity: as cited in the theoretical background, the MUA signal contains the codification used by a neural population. Since this codification is linked to the fire-rate of neurons, we could use this value to predict the response of a stimulation.
All other features where the evoked potential can be altered were excluded.

2.4 Features Extraction

2.4.1 Extracting the LFP

To extract the LFP from the raw signal, we need to perform various steps. First, we need to identify the frequency band that we want to extract. In this case, we use a band-pass filter at 1-100Hz cut frequencies. In other similar works, higher cut frequency in the range 100-300Hz are chosen. This is a study that focuses primarily on the evoked potential, which is found in the lower frequency ranges, so we make sure that the 1-100Hz range is good for this task. An order 4 Butterworth filter and the *filtfilt* function from MatLab were used to perform this step. In Figure 2.4, we can see the before and after of the filtering.

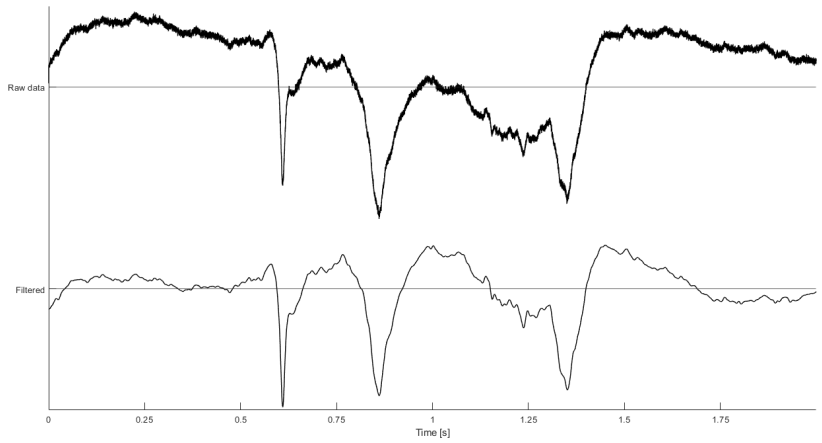


Figure 2.4: A plot showing the difference between a raw recording and the LFP extracted through filtering.

A second important step is to remove artifacts and faulty channels. The one that was detected as faulty is channel 8 in 2021 recordings and channel 3 in 2022-2023 recordings. With the aid of a simple mean between the adjacent channels, we can

easily remove them. Alternately, an interpolation process could be used.
To align all the recordings, a downward shift of 1 has been applied to the channels of the 2022-2023 data. Channels in the range 27-32 were removed, since the process we want to analyze is localized to the deeper layers, and those channels could have some artifact.
The problem faced then concerns the size of the data. At 25000Hz the full data set weights around 30GB and most of the information is useless since there is no component over 100Hz. Therefore, we can now down-sample the data to 250Hz, making it 100 times smaller. In these downsampled data, each time unit will correspond to 4 ms of recording.

2.4.2 Extracting the MUA

To extract the MUA signal, a different approach is required. The first step to perform consists in using a notch filter in the range of 300-5000Hz. The information that we now have is mostly action potential, or spikes, and noise. We can see from Figure 2.5 the various negative peaks of the spikes in the filtered signal. To extract the instant when a spike occurs, we first need to compute the RMS of the signal. Then we use the threshold $-\alpha \cdot RMS$ to detect the spikes, where α is in the range 3-5 and should be chosen based on the quality of the recording. In this case a value $\alpha = 4$ was used. We find now all the points that are below the threshold. [25][26] The next step consists in downsampling the data. We cannot simply use the down sample function; otherwise most of the spikes would be filtered out. A moving sum window of size 100 was used. This is done to prevent information from being lost in the downsampling phase.

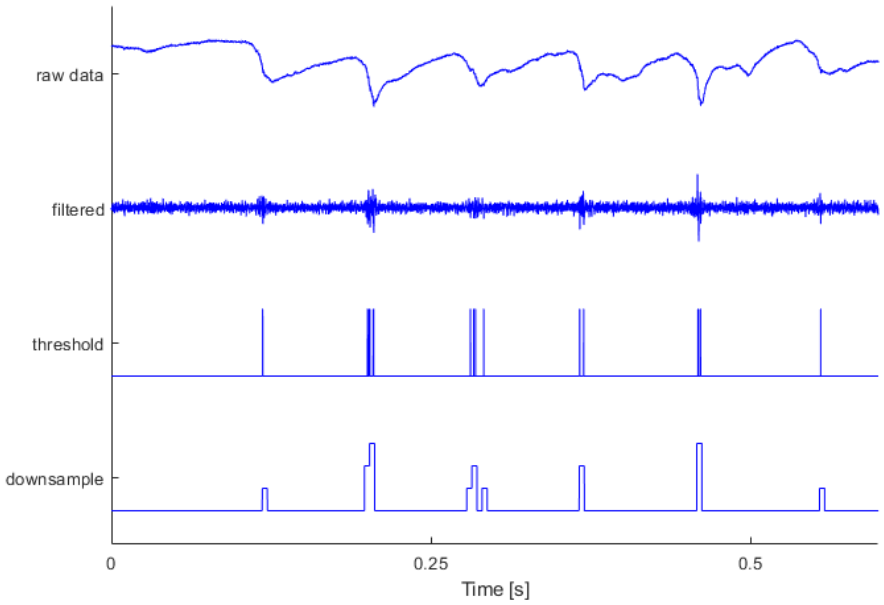


Figure 2.5: A plot showing an example of original recording and the three pre-processing steps required to generate the MUA signal.

Therefore, we just need to align the data from 2022-2023 and remove the faulty channels in the same way as we operated with the LFP case.

2.4.3 Extracting frequency features

The current LFP data is 1GB in size. The *Continuous Wavelet Transform* increases the size of the data size by a factor of 70. For this reason I have decided to use the CWT layer to compute the spectrogram of the LFP directly in the models. The same will be done for the Tetha, Alpha, Beta and Gamma divisions. I have created a new layer that extracts these features using a Butterworth filter and the *filtfilt* function. More details about these layers will be provided in the section dedicated to the description of linear models, Section 3.3.1.

2.5 Analysis of the data

Since the two MUA and LFP data are now ready, we can proceed with a brief analysis of what the signals contain. First, an analysis of the average response to stimulation was performed. This will be followed by a brief discussion on the presence of spontaneous brain activity, which the network should learn to filter out.

2.5.1 Average evoked potential

To compute the average response, we segment the data and align each segment where the stimulation starts. By computing the average value between all segments, we can now extract the average shape of the potential evoked by the stimulation. The average evoked potential is also refered as the *Event Related Potential* (ERP).The evoked potential computed in channels 1-20 of each rat are shown in Figure 2.6.

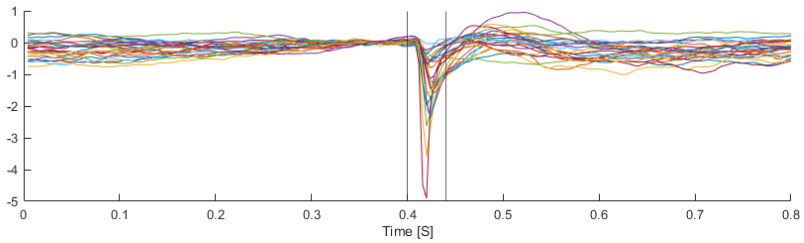


Figure 2.6: Event related potential computed on Layer 4 of the barrel cortex. Each line represents a different rat.

The information that is most relevant for our task is the negative peak of the LFP caused by the synchronous firing of multiple neural populations when the stimuli signal reaches the cortex. The ERP is strongest at layers 4, 5 and 6 and the negative peak is found in the first 40ms post-stimuli.
We can do the same for the MUA signal to see what is the average response in terms of firing neurons. Figure 2.7 shows how even the average MUA response to stimulation is contained in this small 40 ms range.

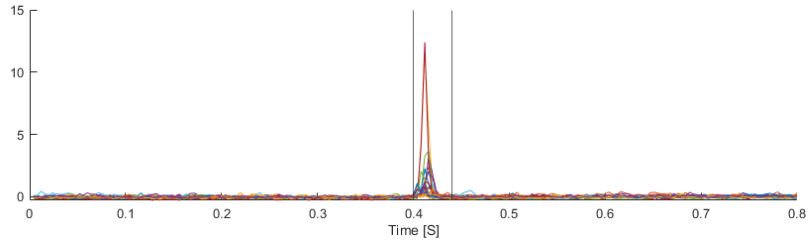


Figure 2.7: Average MUA response to stimulation computed on Layer 4 of the barrel cortex. Each line represents a different rat.

2.5.2 Spontaneous activity

The data are full of spontaneous activity. In some cases, as in Figure 2.8, a burst of activity occurs before stimulation and mixes with the evoked response. We can see how the EP is barely distinguishable from the negative peaks of other spontaneous activity.

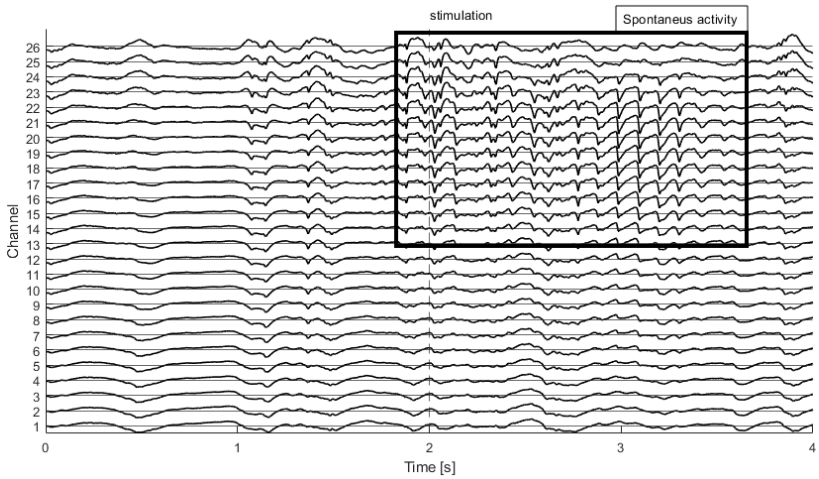


Figure 2.8: Section of LFP recording with high levels of spontaneous activity.

2.5.3 Removing bad recordings

It should be noted that not all recordings are of good quality; some almost do not contain any response to stimulation. Training on those recordings could just worsen the network performance, since the models could learn that spontaneous activity is linked to stimulation. To prevent this, we need to remove the weak response recordings. Currently, the technique used to check if an EP is present is to visibly observe the post-stimulation response in the recording. This system is slow and subject to operator biases. Therefore, we could automate this process with this pipeline:

- Remove any offset from the evoked responses by filtering in the range 8-100Hz;
- Normalize data from each rat;

- Compute the Event Related Potential of the bottom 20 channels, which corresponds to layer 4-5-6 of the barrel cortex, of each rat;
- Compute the global sum of the squares of the Event Related Potentials for each rat. This is the score value of the rat;
- Sort all rats' data on this score.

We can see the average evoked potentials of the various rats sorted by this value in Figure 2.9. There is a simple reason ERP is a good metric of comparison between recordings. If EPs are weak or frequently absent in a recording, then we would get a weak ERP. A strong ERP is present only if the recording has frequent and strong evoked potentials. After verifying the data, in the recording of rats with the highest

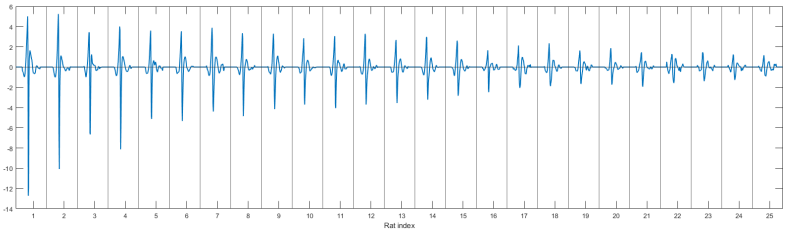


Figure 2.9: The ERPs computed on channels 1-20 on each rat. The ERPs have been sorted by the sum of squares value.

score the evoked potentials are clearly visible in response to stimulation, while in the recording of the rats with the worst score there are no responses in most of the cases. To select the best recordings, two initial models, which are discussed in Section 3.1.6, were implemented. A significant drop in validation performance was observed after adding data from 11 worst performing rats. For this reason, these 11 recordings were removed. After consulting the NeuroChip Lab, we agreed that in fact they had many missing evoked potentials.

2.6 Extracting the Stimulus response window

As shown in Figures 2.6 and 2.7 the main component of the evoked response is found in a 40 ms post-stimulus window. To create the output data, the stimulation vector was used. From this vector we can detect where the stimulation begins and then put at 1 all the samples in the next 40 ms and at 0 the rest of the recording. Value 1 will represent the *evoked potential* label, while value 0 will represent the *NOT evoked potential* label. An example of an output vector is shown in Figure 2.10.

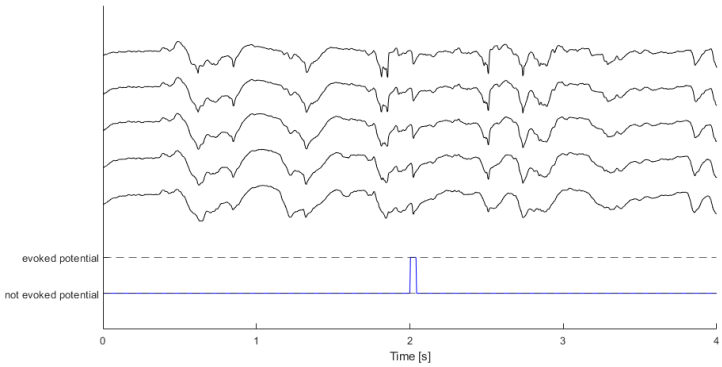


Figure 2.10: Example of data labelling. Each instant of the recording, shown above, is labelled either as "evoked potential" or "not evoked potential".

2.7 Partitioning the data for training

This step consists of partitioning the data in multiple segments, one for each stimulation. The division into segments centered on an event, also called *epoching* in neural signal context, is usually done in EEG and LFP recordings to perform an analysis of the response to the event. This segmentation is useful since it would lead to more heterogeneous mini-batches, which would lead to a faster training. We can see an example of a segment in Figure 2.11.

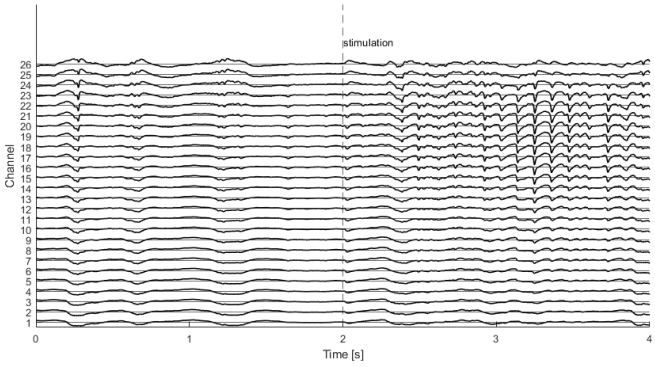


Figure 2.11: Example of segments. The stimulation is found at the center of the segment.

To perform the segmentation, we extract a 12 second data segment centered around each start of stimulation. Problem that may arise from the use of this segmentization are discussed in Section 3.1. Each segment has three labels. The first label identifies the index of the rat created when sorting the average responses. This is used to create cross-validation folds. The second label identifies the recording index. This is used for normalization. The third label is the segment index. The index i refers to the i-*th* stimulation of the recording. This is used to recreate the full recording starting from this data set.Finally, a channel-wise normalization is performed for each recording.

Chapter 3

Methodology and results

This chapter will discuss the most important part of the work I have performed: the creation of the model for the real-time detection of a stimulation.
First, an introduction to the general problem is required. From figure 3.1 we can see how the models are structured. Given the raw data, each model uses either hand-crafted features or learned features and detects the evoke response using a classifier.

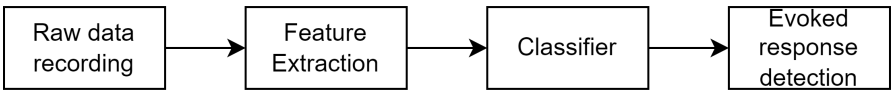


Figure 3.1: General scheme of the *evoked potential detection* model.

In this chapter, all the various steps required to build the models and compare them are discussed:

Preliminary analysis of the models: this section will focus on the definition of the techniques and components required to build the model. In addition, all the initial problems and solutions that were implemented are briefly described.

Model selection: Given the deep learning model found in the analysis performed in the preliminary analysis, a model selection was performed to find a set of good hyperparameters. This section will focus on describing the model ranges that were tested and the results of the selection.

Selected models: All selected models are shown and described in this section.

Comparison of the models: here, the performance of the models on the test set are presented and discussed. From the comparison of the models, the best one is selected.

Optimization of the best model: given the best performing model, a pruning optimization process was performed. In this section, an introduction to model pruning, the choices, and the results of the process are discussed.

Analysis with cross validation: At last, this section will focus on the results of a cross-validation analysis.

3.1 Preliminary analysis of the models

As a first step, an analysis must be performed on how a model can detect evoked potentials. The scope of this analysis was to formulate all the methods and approaches that were later implemented. At first, the general baseline model is discussed. This is followed by an analysis of the use of deep learning layers and methods that are commonly used in similar works. From this analysis, all the possible components that can be employed were mapped. All the fundamental elements that are required for the model to work are then described, such as the output block and the data augmentation. Finally, an initial test, which was performed to discover possible issues that could occur in the training of the models, is briefly discussed, as well as the implemented fixes.

3.1.1 Baseline models

All the four types of features (LFP, MUA, LFP frequency bands, CWT of the LFP) are used as input on a different linear model. The general structure of a linear model is the following:

- Input feature
- Single Convolutional filter

A convolutional filter was used for a simple reason. Different channels have different delay in the negative peak of the evoked potential. By using a single convolutinal filter, we are able to sum the channels taking in account this delay. From the analysis conducted in Section 2.5.1 we can see that the negative peak of an EP is brief, lasting around 5-10ms. Since we selected the window of the response to be 40ms, a max-pooling will be introduced to extend the duration of this peak and fit the target values. Max pooling is non-linear, but it is only going to be used in training and testing. In real time, we would use the output of the conjugate layer as the output for the model.

3.1.2 Analysis of deep learning techniques used on LFP and EEG

First an analysis of the models used in similar work was performed. This analysis mainly focuses on all deep learning techniques implemented in tasks that require LFP or EEG data as input. The task can be in real-time or not, the attribute that the model requires is the ability to extract features from this type of data. EEG data was chosen as possible type of input due to its similarity to LFP. Both originate from the same neural process, and both measure voltage. The main works that were analyzed are the followings:

Recurrent models on hand motion identification: in this work, various recurrent models, including models that use GRU layers and LSTM layers, are compared for the task of predicting the hand motion of a human subject starting from EEG

recordings. All models compared achieve good accuracy in terms of detecting hand motion. [27]

Hand kinematic decoding using a temporal convolutional network: in this work, a temporal convolutional network is used to decode the hand kinematic of a monkey, starting from the LFP data. This experiment using TCN demonstrates that this model architecture is able to provide stable and high decoding performance. The model uses temporal convolutional blocks implemented with casual dilated convolutions, batch normalization, ReLU, dropout, and skip connections. [28]

Thus, we can conclude that the GRU layers, the LSTM layer and the temporal convolutional layers are valid methods used to decode potential-based neural signals. A common approach employed for these types of work is the use of dropout layers and normalization layers. In this thesis work all these techniques were tested. In addition, a series of analyses of how the recurrent and convolutional layer could be mixed to create more complex models was planned. Due to time constraints, only one alternative was tested where recurrent layers are followed by convolutional layers.

3.1.3 Output block

To define the output layer of a model, we need to make various decisions based on the task.

Loss

the response data set is composed of two possible values, 1 for the stimulus response window and 0 for the rest of the signal. For this reason, I am using the binary cross-entropy formula as loss.

$$\mathrm{BCE}\left(Y, T\right) = -\frac{1}{|Y|}\sum_{i=1}^{|Y|} T_i \cdot \log Y_i \left(1 - T_i\right) \cdot \log\left(1 - Y_i\right)$$

The loss is not enough, as there are 300 more time units with value 0 than the ones with value 1. If we use the binary cross-entropy in the vanilla version, the model could just always return 0 and have a really low loss. For this reason, I need to introduce some weight for each class. First, we need to count in the full data set the number C_0 of instants that have label 0 and the number C_1 of instants that have label 1. Therefore, we proceed with the computation of the weights for the two classes.

$$W_0 = \frac{(C_0 + C_1)}{2 \cdot C_0}$$

$$W_1 = \frac{(C_0 + C_1)}{2 \cdot C_1}$$

Now we simply add the weights to the binary cross entropy formula.

$$\text{W-BCE}\left(Y, T\right) = -\frac{1}{|Y|}\sum_{i=1}^{|Y|} W_1 \cdot \left(T_i \cdot \log Y_i\right) + W_0 \cdot \left(\left(1 - T_i\right) \cdot \log\left(1 - Y_i\right)\right)$$

Shifting the window

Both the convolutional and recurrent models that are being tested are causal. This means that we cannot use any future information to predict the stimulus window. A question arises on whether the model would require a delay of the output to correctly find the stimulus response. Consequently, the model could never train on the data set since there is no delay on the output vector I have created. To fix this problem, we have two alternatives: shifting the output for all models or giving the ability to the model to shift the output. Since it is important to analyze the delay introduced by the model, a decision was made to use a final convolutional layer with a single filter, in order to give the ability to shift the output accordingly to the needs of the model. This filter will learn how to move the prediction to fit the training data, as shown in Figure 3.2. This filter is going to be used only in the testing and training phase. When using the model in a real-time scenario, it is not necessary.

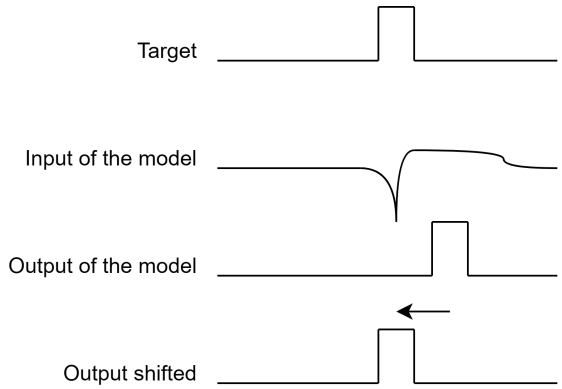


Figure 3.2: Example on how the last convolutional layer shifts the output to fit the target.

Output block structure

The final output block that was applied to all deep models is:

- Single neuron to mix the final hidden representation of the model;
- Single kernel of 200 ms length with right padding;
- Sigmoid activation;
- Weighted binary cross-entropy loss.

During testing time, the sigmoid and loss layer will be removed. In real-time use, the final convolutional layer will also be removed, leaving only a single neuron as the output.
For linear models, a simpler output is used:

- Sigmoid activation;
- Weighted binary cross-entropy loss.

The linear model requires only one convolutional layer. A choice was made not to use the shift window layer and to avoid using causal padding. If we were to use an additional filter after the maxpooling we could increase the complexity of the model and force the model to require non-linear processes.

3.1.4 Data augmentation layer

While we expect the probe to be in a fixed position so that, for example, channels 17-20 are in layer 4, there is no guarantee that the probe is correctly placed or that the layers are found where they are expected to be. Between recordings of different rats, the layer may have some variation in depth. If the data from the test rat come from probes with a different offset compared to the one present in the training data, then we could face a loss in performance. To fix this, I have implemented a data augmentation layer that randomly moves the channels up and down by 1. The chances of moving up, down, or not moving are 1 in 3.
This gives us multiple advantages:

- The data set is virtually 3 times bigger. This should prevent over-fitting since now we go from 800 stimulation to virtually 2400;
- The model gets stronger against offsets;
- The size of the data set does not change.

This layer works only in the forward phase for training. In the prediction phase, the data never moves. An example of how this layer works can be seen in Figure 3.3.

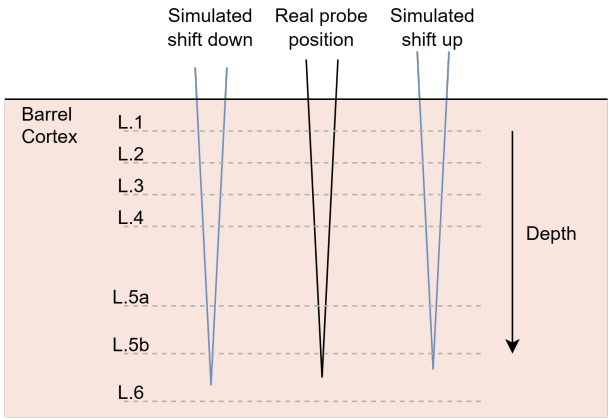


Figure 3.3: Example on how the layer simulates multiple positions of the probe.

3.1.5 Training, Validation and Test set

Training, validation, and test set were created as follows:

- The rat with index 11 has been chosen as the test rat. All the complete uncut recording from this rat will be used in the model comparison. The reason why the data from this rat was chosen is because these recordings contain clear evoked potential and frequent spontaneous activity.
- Data from the remaining 13 rats are split into 85% training set and 15% validation set. In alternative, data from 1 or 2 rats could be used as validation, but a decision was made to use a validation sets that contains data from all rats. The reasoning behind this choice is that, alternatively, we would have no guarantee that the rats chosen for validation are good representative of the whole dataset.

The validation will be used for early stopping and for model selection.

3.1.6 Preliminary setup

The preliminary setup was used to detect major issues early. First, two simple models were implemented, one recurrent and one convolutional. There is no major reason behind the choice of the shape and the parameters of these models. These models were only used to check that the layers worked correctly and were not used later. This section is very important to prevent future problems. As we will see, multiple adjustments to the models were added to prevent major training problems.
The initial convolutional model, described in Table 3.1, uses convolutional layers, non-linear activations, and poolings. The initial recurrent model, described in Table 3.2, uses GRU layers, and non-linear activations.

layer	size of filter	number of filters	activation	pooling	size of pooling
1	100 ms	16	relu	max	10 ms
2	200 ms	4	relu	average	40 ms

Table 3.1: Internal structure of the initial convolutional model.

layer	#GRU nodes	activation
1	24	relu
2	4	relu

Table 3.2: Internal structure of the initial recurrent model.

For two models, five trainings were repeated at each iteration. Any time a problem was detected, the fixes were implemented and the 5 training were repeated on the full training set (validation+training). This process stopped when all the models reached a stable training. Regarding the training option, ADAM optimization, *Glorot* initialization, a learning rate of 0.001 and 16 mini batches were used.

Slow and inconsistent training

As we can see in Figure 3.4, some of the trainings were fast and others much slower. If the purpose is to train hundreds of models in the selection process, we need this to be faster and consistent. The probability that the trained model will get similar results after retraining is really low. One training did not achieve any results in the convolutional case and kept returning class 0. The use of a normalization layer is very important here to improve the training speed and stability in both the recurrent and convolutional cases. After testing the various alternatives, I have concluded that the best normalization for the convolutional layers is the instance normalization, while for the recurrent layers, the layer normalization.

Exploding gradient

As shown in Figure 3.4, both initial models suffer from exploding gradient. To fix this, the **gradient clipping** technique was implemented. After some testing, a value of 16 as the gradient threshold value was found to be the optimal one, which is able to fix the exploding gradient and does not slow down the training.

Data segmentation problems

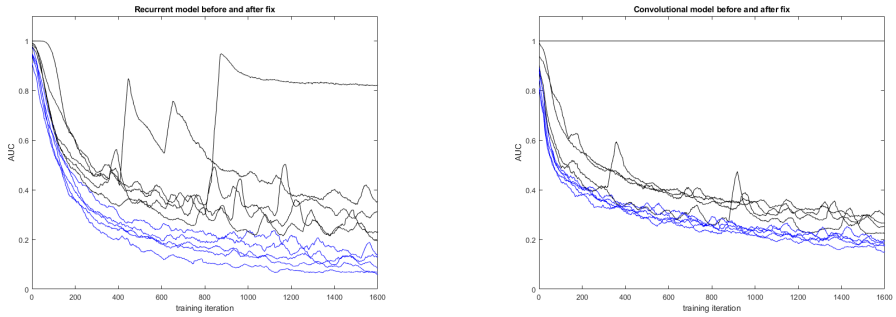
Since recurrent neural networks have a virtually unlimited receptive field, they can use the initial state of memory to detect when the evoked response is found. The evoked response is always found at second 6 in each segment, so the model could just learn how to "count" to 6 and overfit on the data structure. To fix this problem, we can just take a random 10 second window for each segment. This way, the evoked response is randomly placed around the center of the segment, and the network is not able to just "count" the number of seconds between the start of the segment and the evoked response. This problem does not arise for the convolutional models, since the receptive field is at maximum two seconds. A symmetric padding is applied on each layer to prevent possible problems.

Prevent overfitting

In order to prevent over-fitting, two techniques, other than data augmentation, were used. The first is the use of *early stop*. The output layer has been modified to use the AUC value and not the loss as a stopping metric. The second is the use of *dropout layers*. Each model analyzed in the different works uses a dropout factor in the range of 0.2-0.5. Value 0.2 was chosen given the fact that a higher dropout rate would require bigger and more resource-demanding models. The dropout layer is applied after the normalization of each dense, convolutional, or recurrent layer.

Fixed Models

Now we can ensure that the models used in the next phases work correctly and that they can achieve high performance in shorter time. As shown in blue in Figure 3.4, the model now has a more stable and faster training. The network no longer suffers from an exploding gradient, and different training leads to similar results.



(a) GRU model before (black) and after (blue) the fix.

(b) Convolutional model before (black) and after (blue) the fix.

Figure 3.4: Training progress of the models before and after the implemented solutions.

3.2 Model selection

This section will focus on the methodology and results of the model selection process.

3.2.1 Selection process and results

At first a range of possible models was created. The range is divided into three major categories, one for each general architecture:

- **Convolutional:** this type of model uses convolutional layers, pooling layers, dilation, and skip connections.
- **Recurrent:** this type of model uses recurrent unit.
- **Mixed:** this type of model uses elements from both previous techniques.

For the convolutional and recurrent models, a small search for the optimal learning rate was performed. These training parameters are then used in the rest of the search. All models will use dropout layer at 0.2, instance normalization for convolutional layer, and layer normalization for recurrent layers.

Convolutional model range

A first choice is in the number of convolutional layers and the use of additional fully connected layer. The model is divided into three possible subnetworks:

- Input section: this section can use an initial dense layer that has the task of extracting the most useful channels from the input.
- Convolutional section: this section uses convolutional layers and has a probability of using the pooling layer, non-linear activation, dilation, and skip connections. Various layers from 3 to 6 were tested.
- Fully connected section: this section uses fully connected layer and can use non-linear function. A number of layers from 0 to 3 were tested.

The various choices in terms of possible models created are the following:
First, there is an 80% chance that the fully connected layer was added after the input. The range of possible neurons for this layer was uniformly chosen from the range 2-12. Then a number of convolutional layers has been chosen in the range 3-6 with uniform probability. The hyperparameters for each convolutional layer were extracted from the ranges of Table 3.3.

layer	size of filter	# filters	activation	pooling	size of pooling	dilation
1	[20 200] ms	[4 16]	none-relu-elu	none-avg-max	[10 30] ms	1
2	[40 200] ms	[4 16]	none-relu-elu	none-avg-max	[10 30] ms	1
3	[40 240] ms	[2 12]	none-relu-elu	none-avg-max	[10 30] ms	1 or 2
4	[60 240] ms	[1 8]	none-relu-elu	none-avg-max	[10 40] ms	1 or 2
5	[100 300] ms	[1 6]	none-relu-elu	none-avg-max	[10 40] ms	1 or 4
6	[100 300] ms	[1 4]	none-relu-elu	none-avg-max	[10 40] ms	1 or 4

Table 3.3: Range of hyperparameters of the convolutional sub-network.

For the fully connected subnetwork, the range of hyperparameters described in Table 3.4 was used.

layer	number of nodes	type of activation
1	[4 8]	none-relu-elu
2	[2 6]	none-relu-elu
3	[2 4]	none-relu-elu

Table 3.4: Range of hyperparameters of the fully connected sub-network.

The size of the filters, the number of filters and the size of pooling were randomly chosen with a uniform probability. The choice of a pooling type, an activation type, the dilation or the skip connection is global for each model. This means that there is no model that uses both ReLU and ELU or both maximum and average pooling.
For this model, a preliminary search was conducted using *learning rate* in the range 0.0001-0.1 and number of *mini-batch* from 1 to 128.

Convolutional model search results

For the preliminary search, 40 models were trained. For **learning rate** any value found in the range 0.001 to 0.01 performs well. For this reason **0.01** learning rate was used for the rest of the search. The reason why there is little difference between a wide range of learning rates is probably ADAM optimization, which has fewer problems with higher learning rates. In terms of the number of **mini-batches**, dividing the training set into 8 **mini-batches** resulted in good performance and faster training times.
For the final search, 160 models were trained. The **first dense layer** improved the general performance and helped reduce the number of parameters by a lot. As shown in Figure 3.5, the area under the curve value has increased with the addition of this layer.

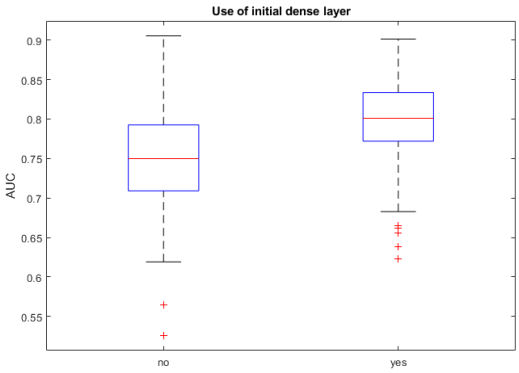


Figure 3.5: Increase in AUC with the addition of an initial dense layer.

In terms of number of nodes for this layer, there does not seem to be much different in the range 6-12, while a small drop in performance was seen with lower values. 6 nodes were chosen.
A clear increase in performance is visible when increasing the number of convolutional layers as shown in Figure 3.6.

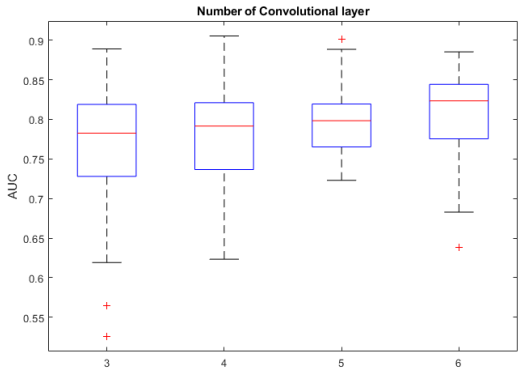


Figure 3.6: Increase in AUC with the use of multiple convolutional layers.

Given the time constraint, more model selection could not have been done, but Figure 3.6 suggests that the 7 or 8 layer could lead to a further increase in the AUC values. The models still have really good efficacy so for this reason 6 are going to be used in the final model. In terms of number of filters, there is little difference between the models. In the best model returned by the search there are many filters in the first layers and less in the deeper layers. This pyramid shape model was chosen for the final one. The best model on the selection all have small filter in the first layers, in the 40-100 ms range, and wider filters in the deeper layers, in the 160-300 ms range. The same filter sizes are going to be used in the final model. The use of skip connection and dilation did not improve the performance but greatly improved the training stability and speed.
The **additional dense network** did not improve the performance. For this reason, it is not going to be used in the final model. In terms of **activation** function, the ReLU activation after each layer outperformed the linear and ELU activation. The **Max pooling** after each layer is also a choice that greatly improves the AUC.

Recurrent model range

For the recurrent model, both the GRU and LSTM models were tested. The model selection in this case had an additional phase where the best recurrent unit among these two is selected. The network used an additional dense layer after input with a range of 2-12 neurons as for the convolutional case.
The possible number of recurrent layers was uniformly selected in the range 1-6. Each layer was extracted from the following ranges of parameters shown in Table 3.5.

layer	type of unit	number of recurrent nodes	type of activation
1	GRU-LSTM	[8 64]	none-relu-elu
2	GRU-LSTM	[4 32]	none-relu-elu
3	GRU-LSTM	[4 32]	none-relu-elu
4	GRU-LSTM	[4 16]	none-relu-elu
5	GRU-LSTM	[2 8]	none-relu-elu
6	GRU-LSTM	[1 4]	none-relu-elu

Table 3.5: Ranges of hyper parameters of the recurrent network.

As for the convolutional model, the choice of the non linear activation function is global, while the other parameters were selected with uniform probability. The same initial search for learning rate and minibatch size is also performed.

Recurrent model search results

A small group of 20 networks, 10 with GRU and 10 with LSTM were trained. For this context the use of **GRU units** returned better performance than the use of LSTM, as show in Figure 3.7.

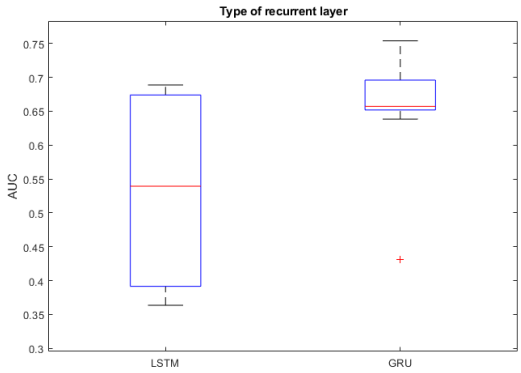


Figure 3.7: AUC of LSTM based model and GRU based model.

To find the best training parameters, 40 models were trained. The best training option that was found for the recurrent model are the same as for the convolutional model, so l**earning rate** of 0.01 and 8 **mini-batches**.
For the final search of the parameters of the models, 160 different models were trained. The use of an initial dense layer in the recurrent model greatly improved the performance, so it was introduced in the final one. 6 nodes were chosen in this case too, same as the convolutional case. For the number of **GRU layers** we see an increase in performance when increasing the number of layers, as shown in Figure 3.8. The use 6 GRU layers returned the highest general performance.

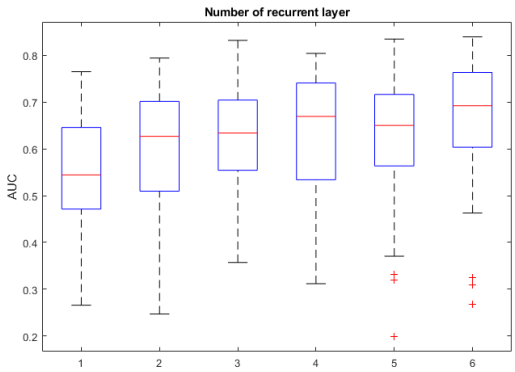


Figure 3.8: Increase in AUC with the addition of more convolutional layers.

Regarding the convolutional model, RELU activation greatly improved the general performance, as shown in Figure 3.9.

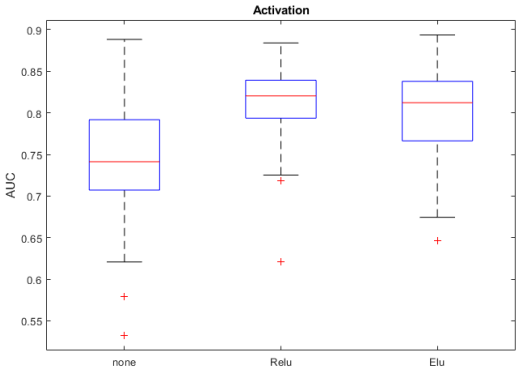


Figure 3.9: AUC on different types of non linear activation.

Mixed model range

The mixed model uses GRU units followed by convolutional layers. Since this step was done after the convolutional and recurrent model search, some of the information from previous searches was used to make this search faster. For example, there was no need to test the initial dense layer, since both of the previous cases have shown that there is an increase in performance in adding it. There is also no need to search for learning rate and mini-batch size.
This model is divided into two subnetworks: the recurrent subnetwork and the convolutional subnetwork. For the first, 1 to 3 recurrent layers will be randomly added with hyperparameters extracted from the values presented in Table 3.6.

layer	number of GRU nodes	type of activation
1	[8 64]	none-relu-elu
2	[4 32]	none-relu-elu
3	[4 32]	none-relu-elu

Table 3.6: Ranges of hyperparameters of the recurrent sub-network.

For the convolutional subnetwork, 1 to 3 layers are selected with ranges extracted from the ranges presented in Table 3.7.

layer	size of filter	number of filters	type of activation	type of pooling	size of pooling
1	[20 100] ms	[4 16]	none-relu-elu	none-avg-max	[10 30] ms
2	[40 120] ms	[4 16]	none-relu-elu	none-avg-max	[10 30] ms
3	[40 160] ms	[2 12]	none-relu-elu	none-avg-max	[10 30] ms

Table 3.7: Ranges of hyperparameters of the convolutional sub-network.

In this case, due to the limitation found while implementing the model, I was unable to add the skip-layer connection and the dilation factor. This should not affect

the performance, as we have discussed in the convolutional model selection, but just the number of parameters.

Mixed model search results

The main aspect of this search is the number of layers and the hyperparameters of the layers. As for the other two previous searches, there is a clear increase in the AUC of validation when adding both more recurrent layer and convolutional layers, as shown in Figures 3.10 and 3.11. The use of ReLU and max pooling was chosen as for the two previous cases.

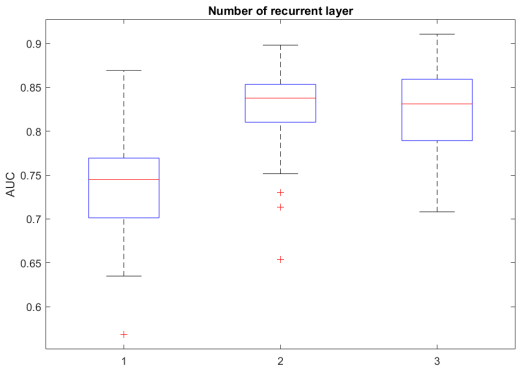


Figure 3.10: AUC on different number of recurrent layers.

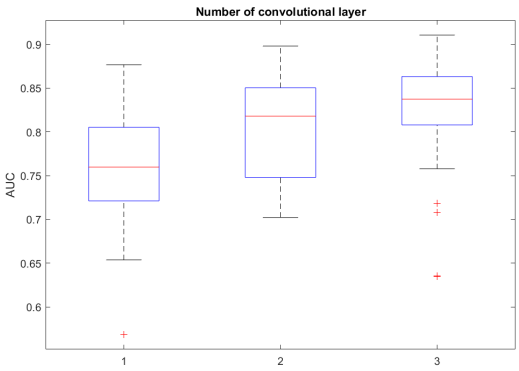


Figure 3.11: AUC on different number of convolutional layers.

3.2.2 Possible improvements

Having limited resources and time I was able to perform only a certain amount of model training. The results of the model selection clearly show a positive trend in performance when using deeper models. The obtained model are all well suited for our task, but they can be improved with a more exhaustive search. Initially more than 6 layers per model were excluded, but given the results a better search with up

to 12 layers could be performed. Another type of improvement, in this case on the mixed model, would be to expand the number of alternatives, such as using firstly the convolutional layer and then the recurrent, or to merge the two. Other than these two improvement that could be applied, this random search was still a major task for my work and was able to extract a set of working final models.

3.3 Selected models

All the models that were selected are going to be presented in this section. The linear model used as baseline are describe followed by the deep learning models.

3.3.1 Baseline models

The linear models can be created without the use of a complex model selection. In this case linear model can be constructed without complex model selection. The only hyper-parameters chosen were the 100 ms filter size for the layer, the learning rate and the batch size. Each of the four models works with one of the features discussed in Section 2.3.

Linear convolutional layer on LFP

This is the simplest model I constructed. This model uses LFP data as input and linearly mixes the channels to find the evoked response. If this model performs well, it would mean that we just need to filter the signal at 1-100Hz, linearly transform it and use a threshold in order to detect the evoked response.

- Input layer
- Data augmentation layer
- 1 convolutional layer with 1 filter of size 200ms and same padding.
- max pooling of 40ms
- sigmoid activation
- weighted binary cross entropy

Linear convolutional layer on MUA

This second model uses MUA features as input. This model will check if there is any spike burst in any given channel in relation to stimuli.

- Input layer
- Data augmentation layer
- 1 convolutional layer with filter size 200ms and same padding.
- max pooling of 40ms
- sigmoid activation
- weighted binary cross entropy

Linear convolutional layer on LFP frequency bands

This model uses a layer that I have implemeted, capable of extracting the 4 frequency bands, Tetha, Alpha, Beta, and Gamma. An example of input and output of this layer is shown in Figure 3.12.

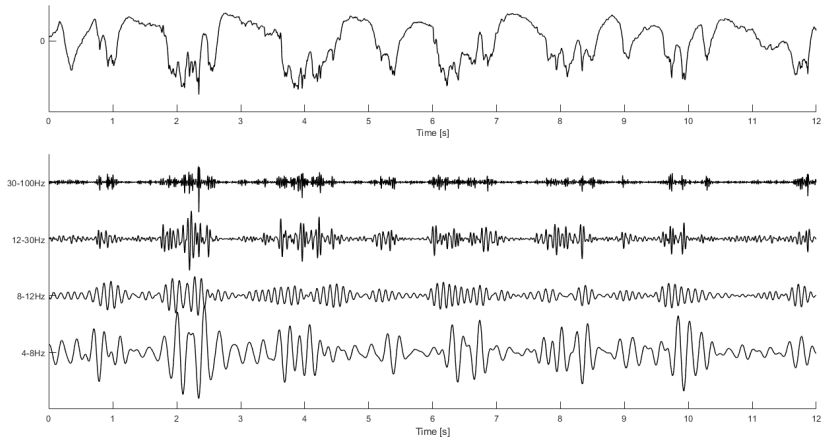


Figure 3.12: Example on how the *filter bank layer* converts the values of a channel in four frequency bands.

The structure of the model of this model:

- Input layer
- Data augmentation layer
- Filter bank layer with 4-8Hz, 8-12Hz, 12-30Hz and 30-100Hz ranges
- 1 convolutional layer with filter size 200ms and same padding
- max pooling of 40ms
- sigmoid activation
- weighted binary cross entropy

Linear convolutional layer on CWT

This model uses the CWT layer. This layer is able to compute the *Continuous Wavelet Transform* of the LFP signal. This model has the ability to extract information from any frequency bands and any channel if needed. If there is a space-frequency correlation between the LFP and the evoked response, then this model will be able to find it. An example on the output of the CWT Layer is shown in Figure 3.13.

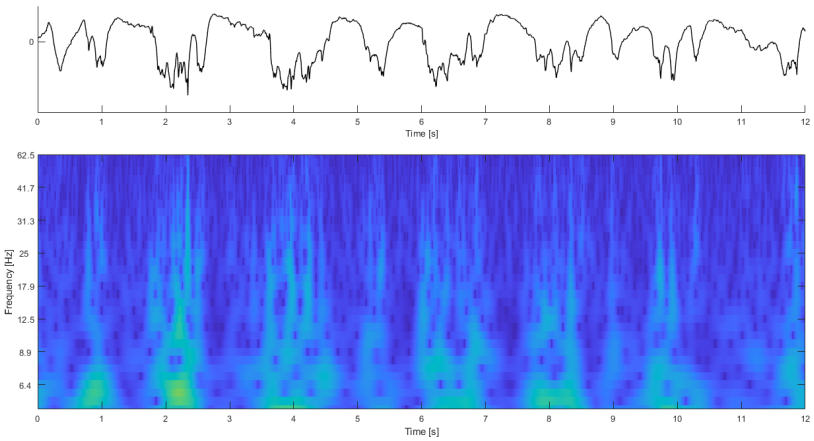


Figure 3.13: Example on how the *CWT layer* converts the values of a channel in a spectrogram.

The structure of the model is:

- Input layer
- Data augmentation layer
- CWT layer with 8 voices per octave and range 0-100Hz.
- 1 convolutional layer with filter size 200ms and same padding.
- sigmoid activation
- weighted binary cross entropy

3.3.2 Convolutional Model

In Figure 3.14 the complete architecture of the model is presented. This is the Temporal Convolutional Model extracted from the model selection. The model uses six layers, each one with a set of causal convolutional filter and a skip connection. The number of filters progressively decreases with deeper layers, while the size and dilation progressively increase. The model has in total 11'600 weights and has a receptive field of 1 second.

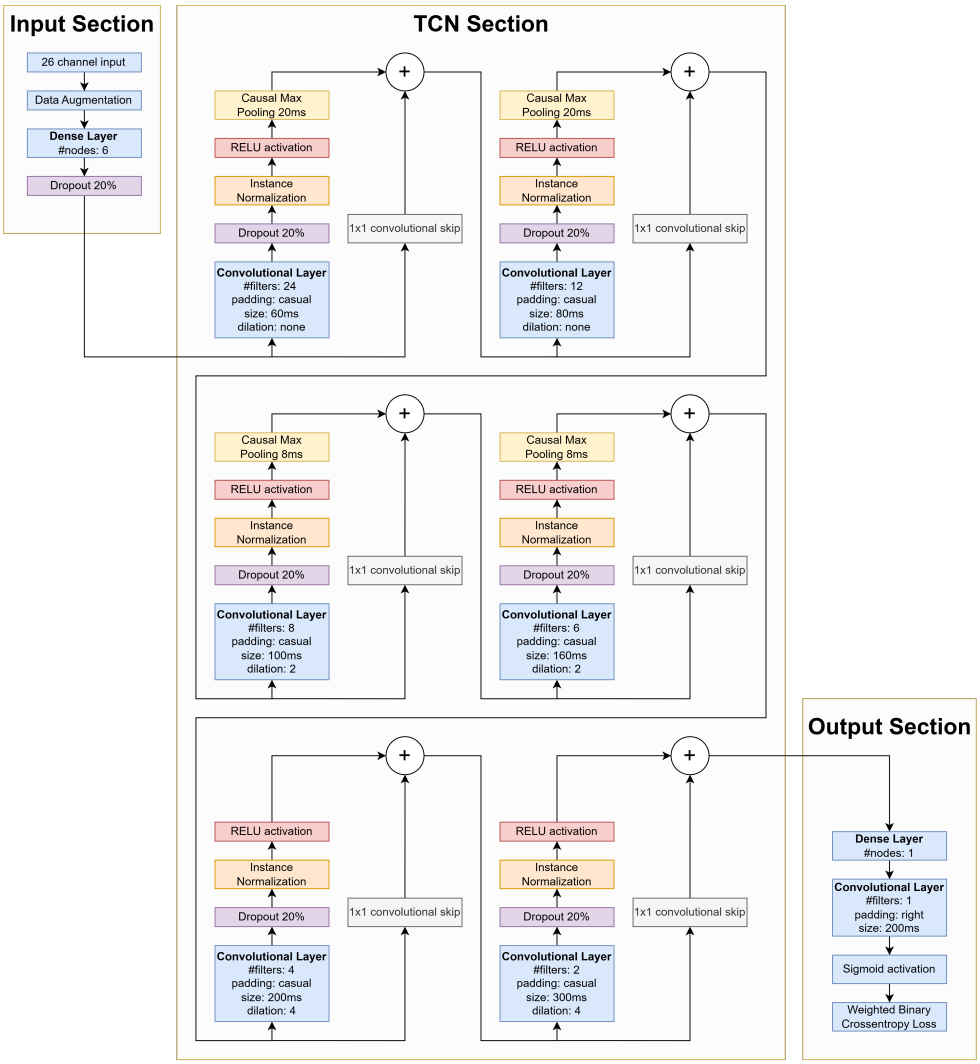


Figure 3.14: Architecture of the temporal convolutional model.

3.3.3 Recurrent Model

In Figure 3.14 the full architecture of the GRU model is presented. The model uses six layers, each one containing a GRU layer and ReLU activation. In total it uses 5'000 learnable parameters.

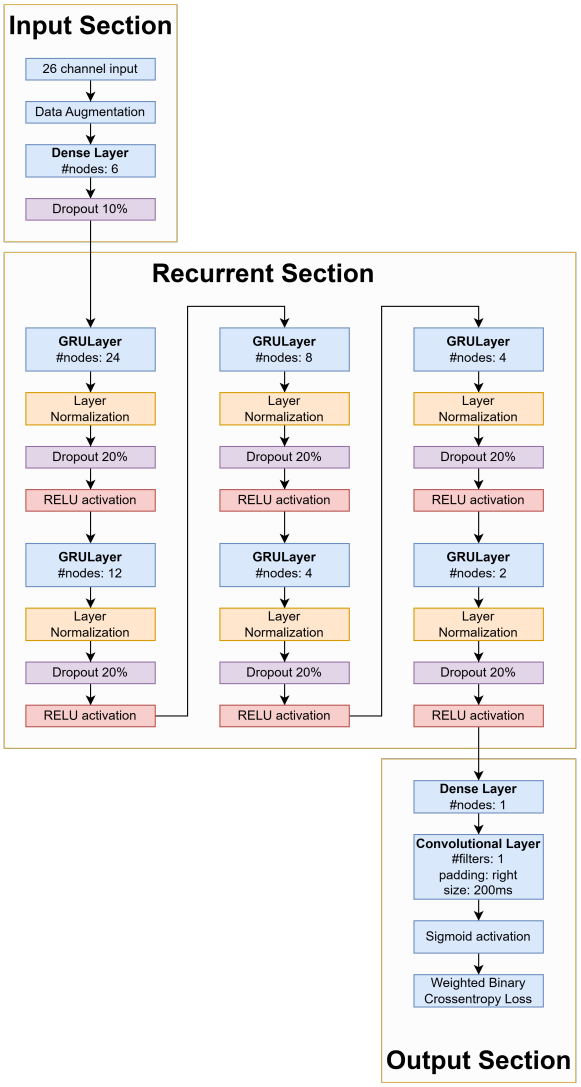


Figure 3.15: Architecture of the recurrent model.

3.3.4 Mixed Model

In Figure 3.14 the complete architecture of the Mixed model is presented. The model uses three GRU layers with ReLU activation and three vanilla causal convolutional layers without dilation and skip connections. In total, it uses 8'000 learnable parameters.

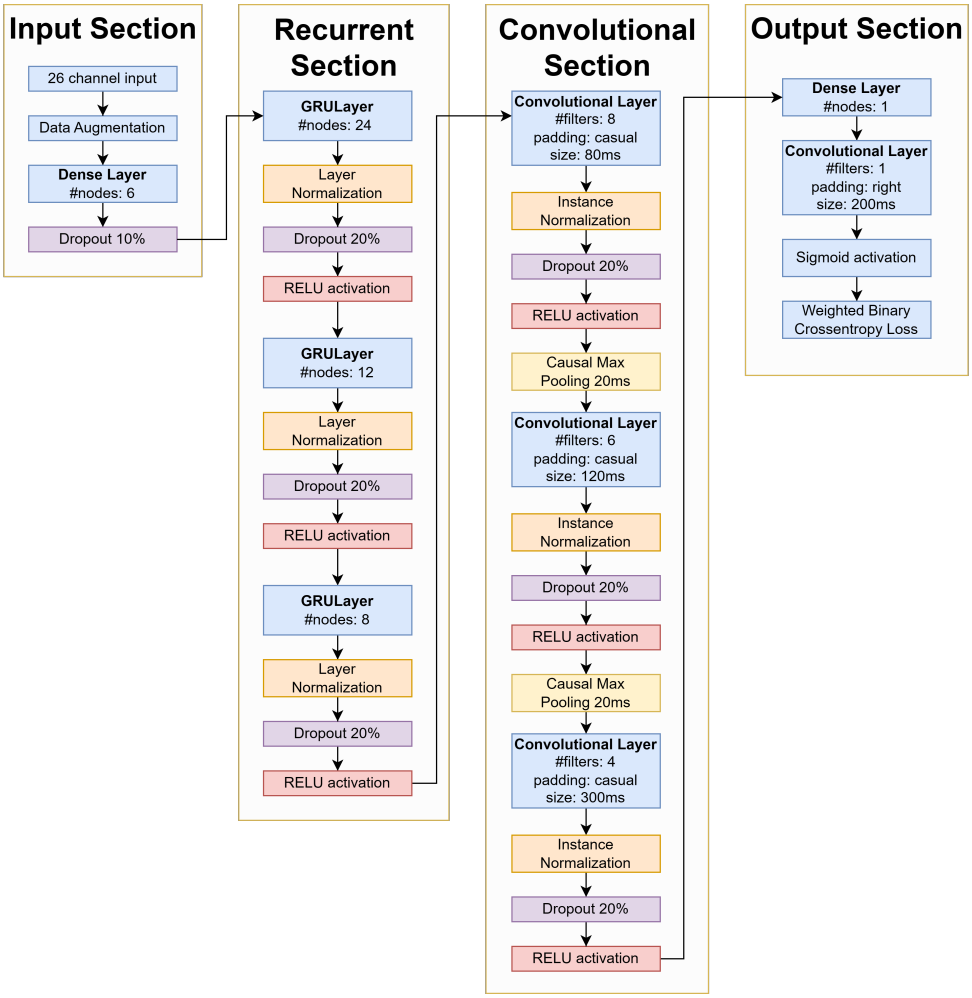


Figure 3.16: Architecture of the mixed model.

3.3.5 Ensemble Model

The ensemble model is fairly simple. First, it normalizes the output of the previous layers and sums them together. This model was added to check if the different deep learning models described above are capable of finding different types of evoked potentials. If this is the case, then the ensemble would achieve the highest efficacy of the three with the lowest efficency.

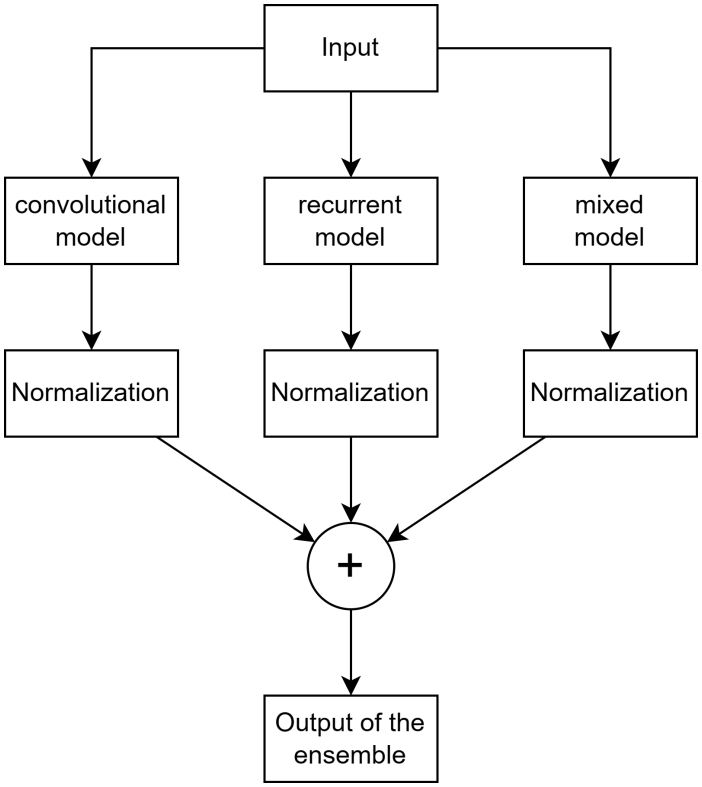


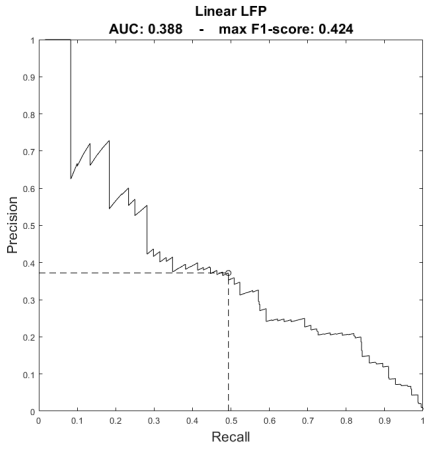
Figure 3.17: Architecture of the ensemble model.

3.4 Testing the models

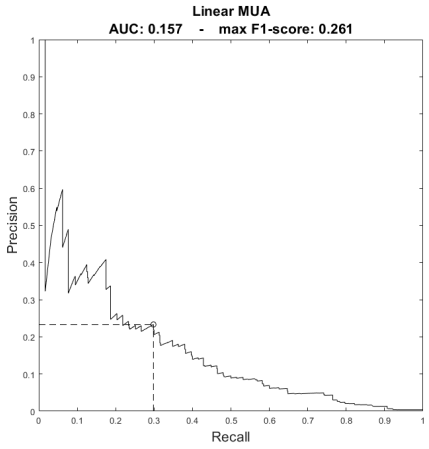
In this section, the results are presented on the test set of the various models. As a reminder, the rat with index 11 was chosen as the test rat, and the AUC and F1 score are computed on full uncut recording extracted from this rat. To compare each model, the precision-recall curve and the output of a full recording are used. To show this output, which is 5 minutes long, it is divided into 30 segments centered on the stimulation of 10 seconds each. This is not to be confused with the segmentation process. In each output Figure, the detected EPs are going to marked in blue.

3.4.1 Baseline models

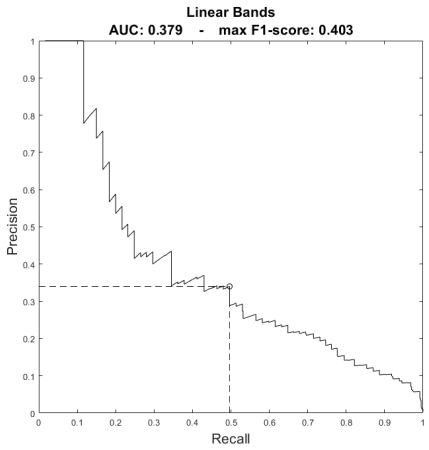
The linear models show decent ability to detect EP. All AUC values are in the range of 0.18 to 0.39. In comparison, a random output has AUC close to 0, given the different distributions of the classes.



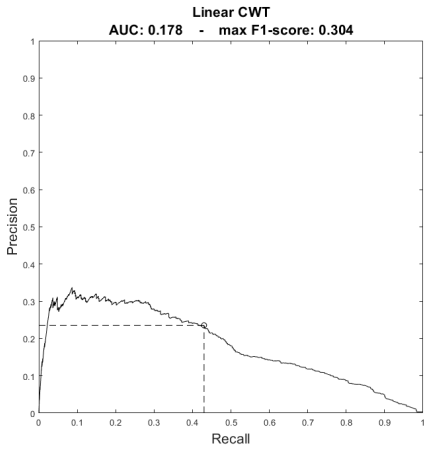
(a) Linear model on LFP.



(b) Linear model on MUA.



(c) Linear model on LFP frequency bands.



(d) Linear model on CWT.

Figure 3.18: Precision-Recall curve of the linear models.

Using LFP signal

This model has shown a good ability to identify the stimulus but cannot filter out any other similar spontaneous activity. As shown in Figure 3.18b, the model has an AUC of 0.388 and a maximum F1 score of 0.424. At that threshold, the model was able to find half of the evoked response, but only 1 out of three prediction is a true positive. We can see from the output of the full recording, Figure 3.19, the main problem of these types of models. In multiple sections the spontaneous activity bursts are recognized as multiple evoked potentials.

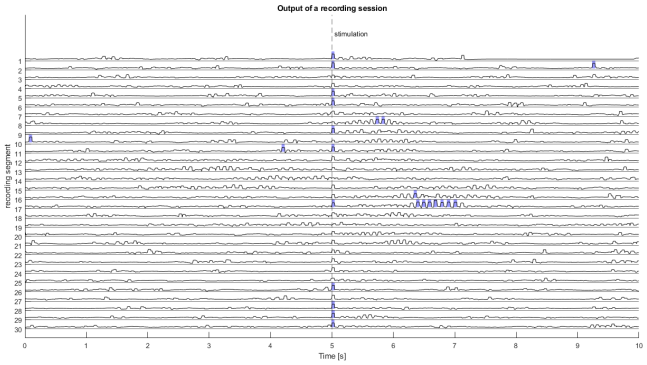


Figure 3.19: Output of a full recording using LFP.

Using MUA signal

The linear model on the MUA signal has the worst performance of all the models tested.As shown in Figure 3.18a, the model has an AUC of 0.157 and a maximum F1 score of 0.267. In this case, the model cannot correctly identify most of the evoked potentials. The reason for this is simple. If the probe is placed in the wrong barrel or in the space between two different barrels, we would not be able to record the correct MUA signal. For the MUA we can only detect activity if the probe is perfectly placed in the correct barrel. From Figure 3.20 it can be seen that we are able to detect just

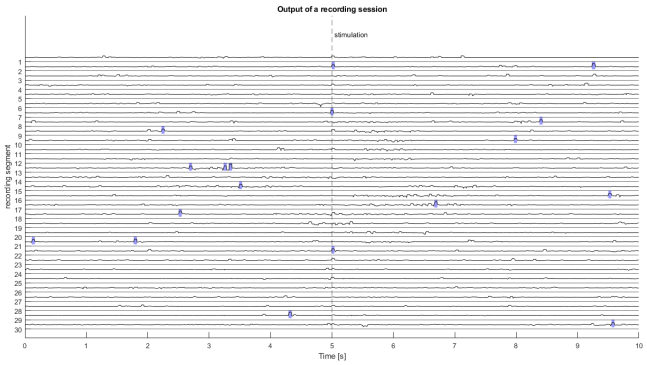


Figure 3.20: Output of a full recording using MUA.

Using LFP frequency bands

Using the four frequency bands, the model has shown slightly worse results than the raw LFP. This would suggest that the spontaneous activity and the evoked potential are found in the same frequency range. As shown in Figure 3.18c, the model has an AUC of 0.379 and a maximum F1 score of 0.403. The output of the model, shown in Figure 3.21, is also similar to the simpler LFP case.

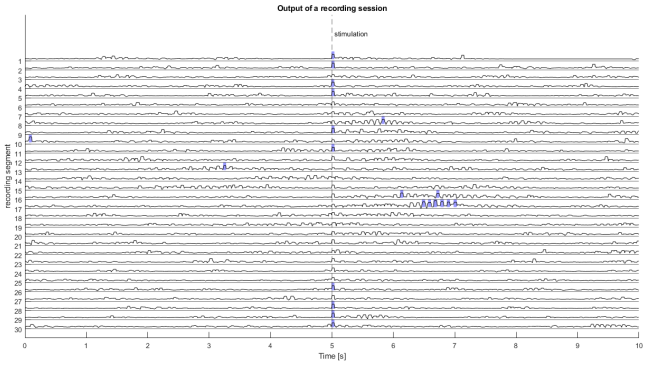


Figure 3.21: Output of a full recording using frequency bands.

Using CWT

The CWT model is the second worst in terms of performance. As shown in Figure 3.18d, this model has an AUC of 0.178 and a maximum F1 score of 0.304. The model has a good ability in terms of detecting the EP, but has problem in discriminating it to other activity. We can see from the output, shown in Figure 3.22, that the model was not able to filter out spontaneous activity.

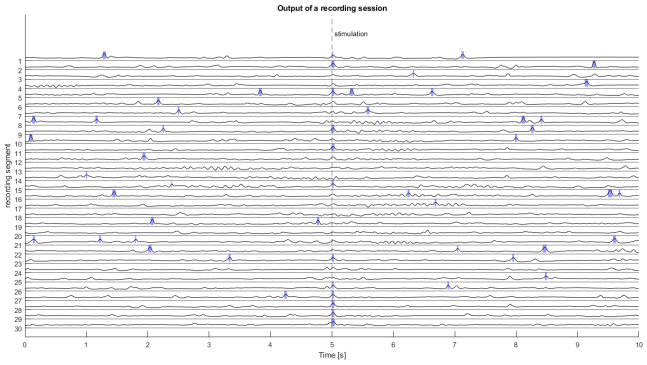
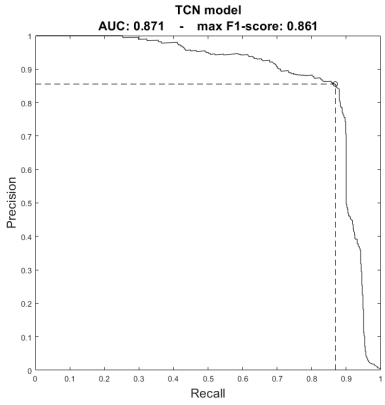


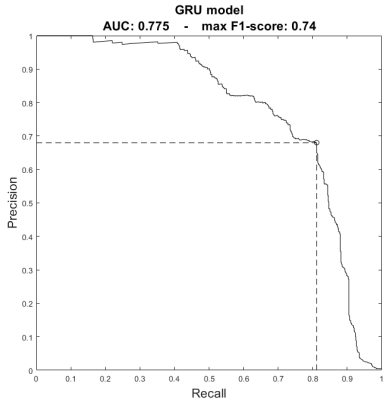
Figure 3.22: Output of a full recording using CWT.

3.4.2 Deep models

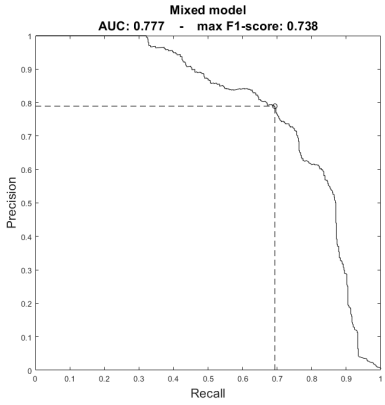
Using deep learning methods, we can greatly increase performance over the baseline. In this case, as shown in Figures 3.23, we have an AUC in the range of 0.77-0.87 and a maximum F1 score in the range of 0.73 to 0.86. These models have the ability to detect up to 86% of stimulation.



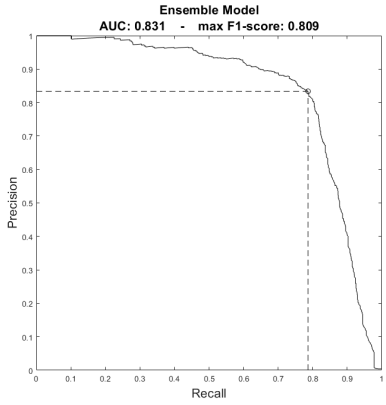
(a) Temporal Convolutional Network.



(b) GRU Network.



(c) Recurrent-Convolutional Network.



(d) Ensemble.

Figure 3.23: Precision-Recall curve of the deep models.

Temporal Convolutional Model

Starting with the temporal convolution model, a large increase in performance can be seen over the baselines. As shown in Figure 3.23a, the model is able to find the majority of the evoked responses, with an AUC of 0.871 and a maximum F1 score of 0.861. In this case, the maximum. The output of a full recording using TCN, shown in Figure 3.24, has only 3 false negatives and 5 false positives.

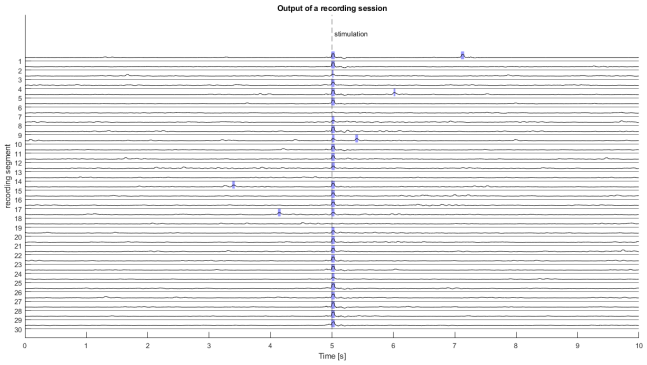


Figure 3.24: Output of the TCN on a full recording.

With the analysis of the average response, shown in Figure 3.25, it is possible to calculate that the delay introduced by the model is 80 ms. The model generates a wavelike pattern when a evoked potential is detected. This type of response has only been seen on the convolution model during this analysis. This output may require some post-processing in order to be correctly used, such as a moving max of the absolute value of the output.

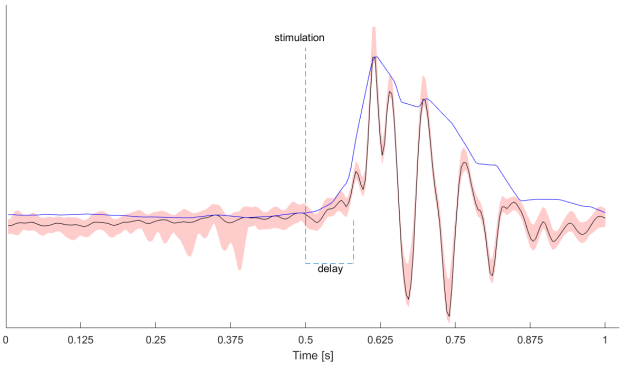


Figure 3.25: In black, the average response of the TCN model. In blue, a possible post-processed signal.

GRU Model

The recurrent model has achieved performance higher than the baseline but lower than the TCN. The model has an AUC of 0.775 and a maximum F1 score of 0.740, as shown in Figure 3.23b. If we compare the output of this model, shown in Figure 3.26, with the TCN, the recurrent model has higher false positive and false negative rates.

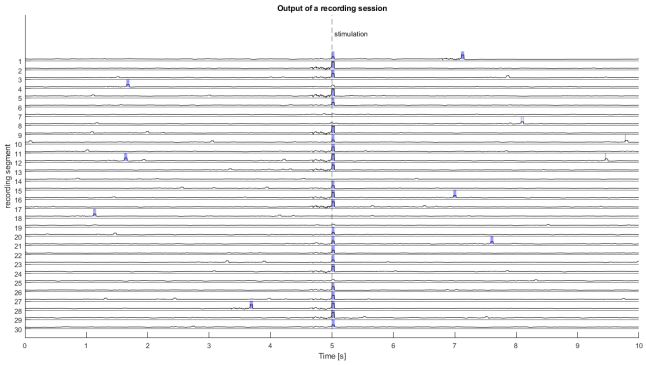


Figure 3.26: Output of the GRU network on a full recording.

The delay is of just 32 ms and the response is the best in terms of quality. It is noticeable from Figure 3.27 that the recurrent model returns a single peak just after detecting the evoked response. This output does not require any post-processing to be used.

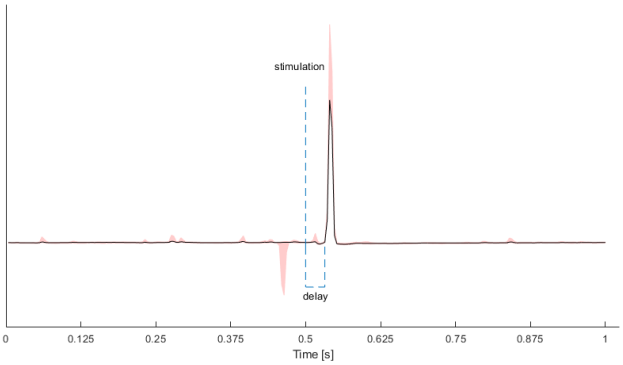


Figure 3.27: Average response of the recurrent model.

Mixed Model

The mixed model has achieved performance that is slightly better than the GRU model. The model has an AUC of 0.777 and a maximum F1 score of 0.738, as shown in Figure 3.23c. In this case, the optimal precision and recall values chosen lead to more false negatives and less false positives, as shown in Figure 3.28.

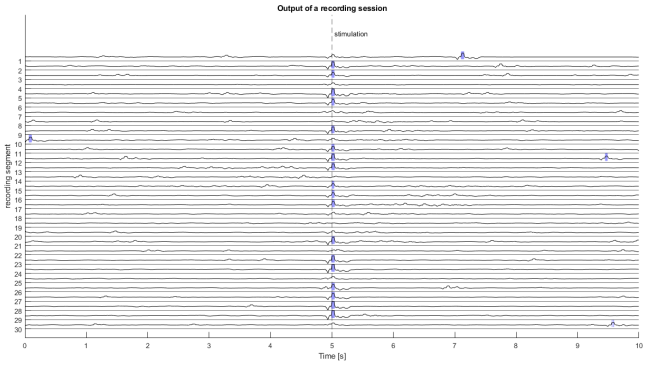


Figure 3.28: Output of the mixed model on a full recording.

The delay calculated using the average response, shown in Figure 3.29, is 90 ms. The model, as for the TCN, generates a wavelike pattern so some post-processing could be required.

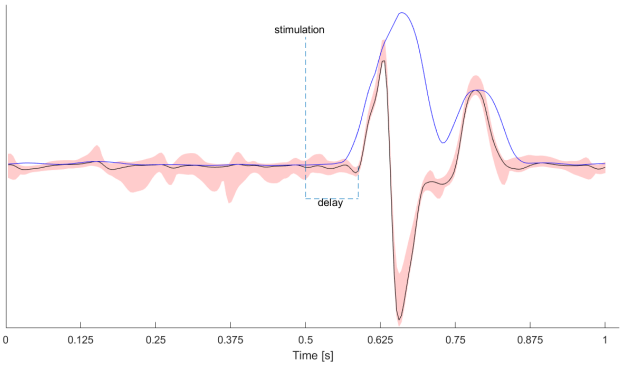


Figure 3.29: In black, the average response of the mixed model. In blue, a possible post-processed signal.

Ensemble Model

The ensemble model returned results similar to the convolutional model with an AUC of 0.831 and a maximum F1 score of 0.809, as shown in Figure 3.23d. This tells us that the GRU and the mixed models did not learn any different feature than the TCN one, and the correct outputs of those models are just a subsets of the correct output of the convolutional models. As we can see from the full output, shown in Figure 3.30, the ensemble has the same false negative as the TCN.

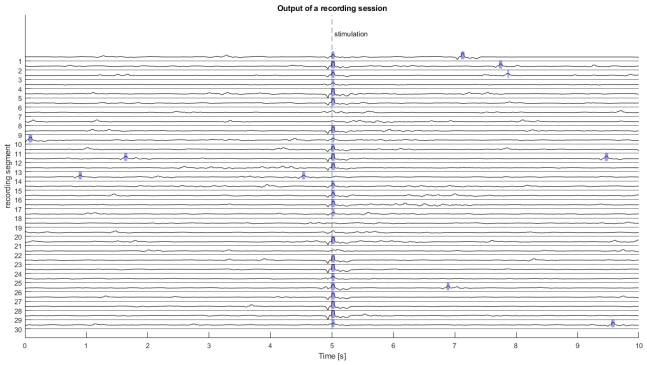
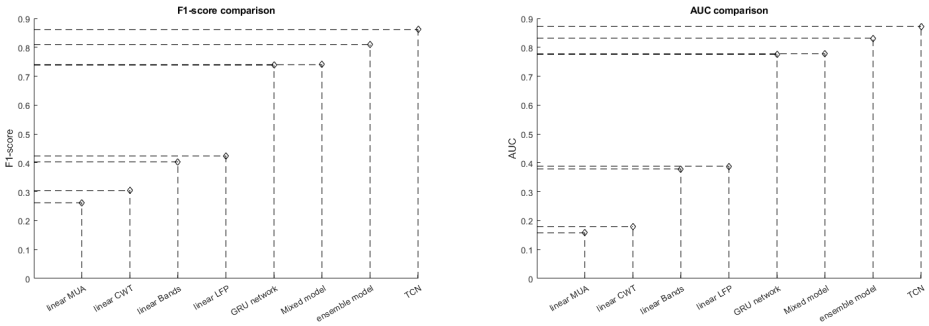


Figure 3.30: Output of the ensemble model on a full recording.

3.4.3 Comparison



(a) Comparison of maximum F1-score. **(b)** Comparison of AUC.

Figure 3.31: Comparison of the efficacy of the models.

When comparing the efficacy, shown in Figures 3.31, the Temporal Convolutional Network outperformed all the other models. Although linear models are capable of detecting a good portion of the evoked responses, they still perform worse than deep neural network models. The GRU network has returned the worst performance of the deep models, followed by the mixed model. In terms of delay and number of parameters required, the TCN is the most expensive computationally and with a delay of 80 ms. The recurrent model is currently the best in terms of efficiency, with the lowest number of parameters, half of the ones required by the TCN, and with a lower delay of only 32ms. A comparison of delay and number of parameters is shown in Table 3.8.

Model	**Delay**	**#Parameters**
TCN	80ms	11'600
GRU	32ms	5'000
Mixed	90ms	8'000

Table 3.8: Efficiency comparison.

The Temporal Convolutional Model was selected for its high effectiveness in detecting the evoked response. The next step is to make this model as efficient as possible for a real-time scenario.

3.5 Optimization of the convolutional model

The Temporal convolutional model is the best performing one in terms of efficacy. The model has in total 11'600 learnable weights. If each weight is used for a sum or a multiplication, we would have an efficiency problem. With a 250Hz signal the model would require at least 3 million operations per second, which is low for a typical computer, but a really expensive task for a neural prosthesis.
In this final section, we analyzed whether it is possible to make this model as efficient as possible without losing too much of its ability. In order to achieve this, we use a technique called pruning. Pruning mainly consists in identifying the least important weights in the model and deleting them. Due to time constraints, I was only able to implement magnitude pruning. Techniques as the *Optimal Brain Surgeon* [29] or the Optimal Brain Damage, were not implemented for time constraints reasons. OBS uses the Hessian of the loss function to remove some of the weights and improve the remaining one; whereas magnitude pruning ranks each weight on a magnitude value and iteratively removes the worst and fine-tunes the remaining ones. Although simple, magnitude pruning is capable of removing up to 90% of the weight of a model. [30].
Multiple approaches of pruning may be adopted for the convolutional model:

- **Weight pruning**: given a kernel, we find the least useful part of the kernel and delete it. For example, if we have a kernel with 100 weight but only 10 are used, then we can delete the other 90 to improve the performance. This is the most complex one to implement.
- **Kernel pruning**: given a kernel, we check if the whole set of weights of the kernel is useful; otherwise, the entire kernel is deleted.
- **Filter pruning**: given a filter, we check if all the kernels of the filter are useful; otherwise, we delete the entire filter.

The best choice is the weight pruning, but the only one I was capable of using is the filter pruning, since it is not possible to simply remove single kernels or weight from a network in MatLab.
Now we need to define a metric to understand if a filter is "useful" to the task. There are multiple ways to do it, but the two analyzed are the following:

- **Weight magnitude**: we compute the sum of the absolute values of all the weights in a filter;
- **Gradient magnitude**: we forward the validation set in the model and compute the gradient for each weight. We then multiply each gradient and each weight and for each filter we compute the sum of the absolute value of this result.[31]

Both methods were tested and compared.

3.5.1 Pruning pipeline

In order to correctly prune the model, I implemented a greedy pruning pipeline, which is schematized in Figure 3.32. The pipeline works as follows. We iteratively checked half of the filters, ordered by magnitude, of each layer. We remove the filter with lowest magnitude and retrain. If the removal of a filter leads to a drop in validation AUC of more than 5%, then we would restore that filter and move to the next layer. Otherwise, we continue until half of the filters of the layer are removed. The process is carried out multiple times, until it is not possible to remove any filter without increasing the error (from now on the decrease in validation accuracy will be referred to as the *error* introduced by pruning). The reason why in each iteration we stop at half of the filters is to prevent the behavior learned by the initial layers from moving into deeper layers, which still have a lot of filters. We can immagine a scenario were the first layer drops from 24 filters down to 1 and all the behavior that was pruned is relearned by filters in layer 2.

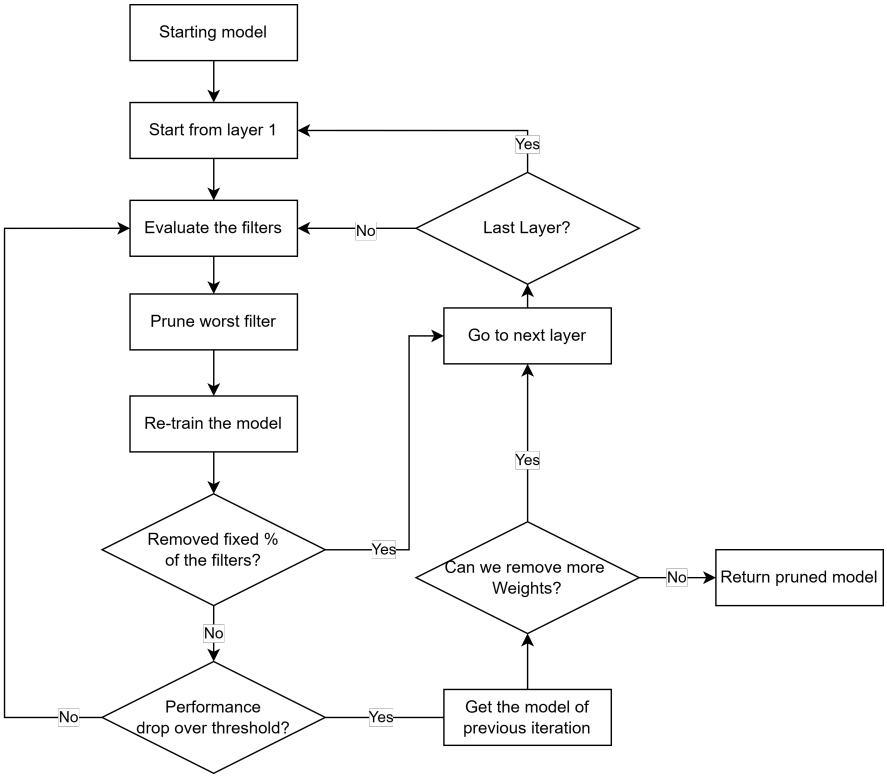


Figure 3.32: Scheme of the pruning pipeline.

3.5.2 Pruning results

Weight magnitude pruning results: Using the weight magnitude, the pruning process was able to remove 85% of the parameters before reaching the 5% error threshold. The most optimized model now has 1,800 parameters compared to 11,600 of the initial TCN. In Table 3.9 the number of filters used by the original model are compared to the pruned one.

Layer	#Filters Before Pruning	#Filters After Pruning
1	24	7
2	12	4
3	8	2
4	6	2
5	4	1
6	2	1

Table 3.9: Number of filters pre- and post- pruning using weight Magnitude.

Figure 3.33, shows us the performance drop, in terms of error, caused by the pruning of different filters. The performance of the model is stable until we pruned around 60% of the parameters. When removing 60-80%, a clear and continuous drop in performance can be detected.

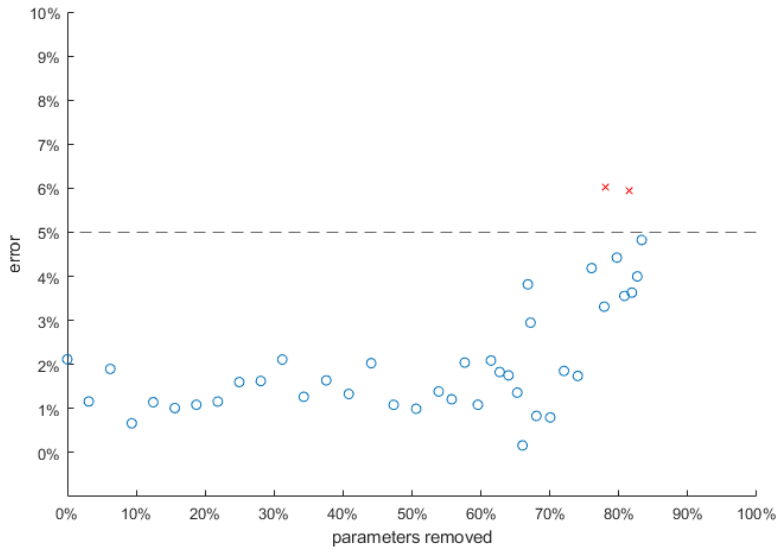


Figure 3.33: Increase in error of the weight magnitude pruning when removing parameters.

Gradient magnitude pruning results: Using the gradient magnitude the results are much better. Using the gradient magnitude, the pruning process was able to remove 92% of the weights before reaching the 5% validation error threshold. The pruned model now has 910 learnable parameters. The number of filters shown in Table 3.10 is low and significantly lower compared to the result of weight pruning.

Layer	#Filters Before Pruning	#Filters After Pruning
1	24	3
2	12	2
3	8	2
4	6	2
5	4	2
6	2	1

Table 3.10: Number of filters pre- and post- pruning using gradient Magnitude.

In Figure 3.34 its possible to see that the pruning is able to remove 80% of the weights before having any drop in efficacy. In the range 80-90% we start to remove weights that have some significance. In this example a 5% validation error threshold was chosen, but with a 2% we would have a model with similar number of parameters and higher performance.

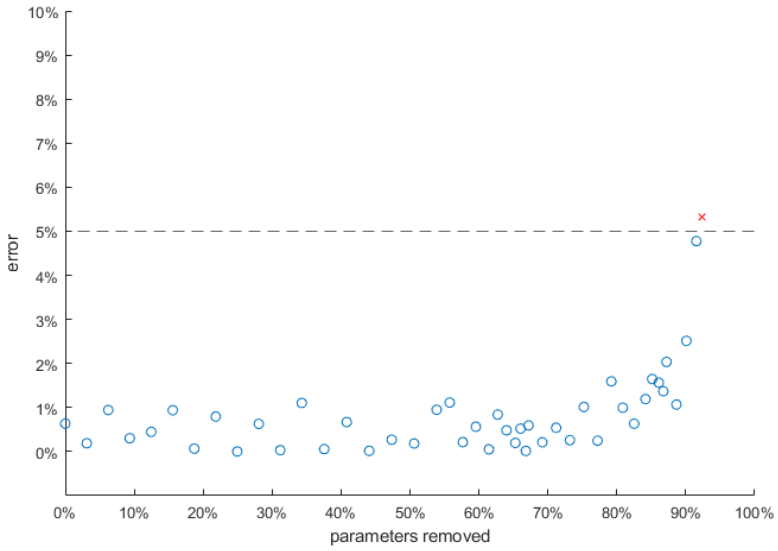


Figure 3.34: Increase in error of the gradient magnitude pruning when removing parameters.

Pruned Model The final model pruned using the gradient magnitude technique is now much smaller than the other deep models proposed in this work. The use of a gradient to compute the importance of a filter had major results. As it can be seen from Figures 3.33 and 3.34 the pruned model is able to use less and more generalized hidden representation.

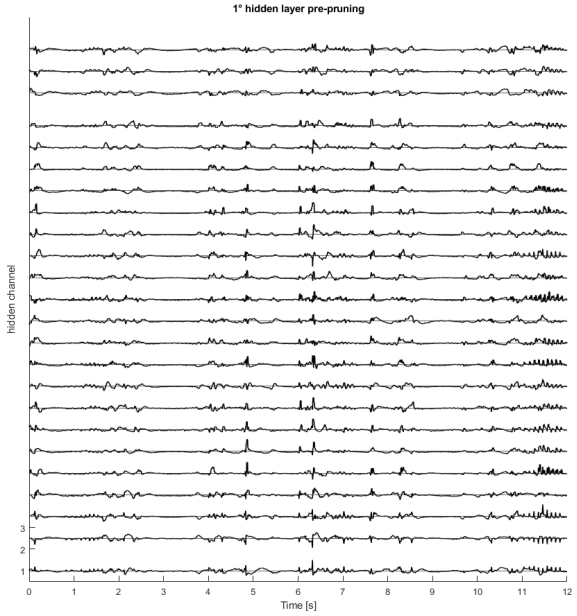


Figure 3.35: Output of layer 1 before pruning.

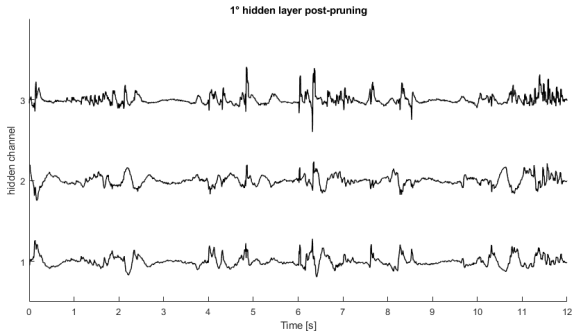
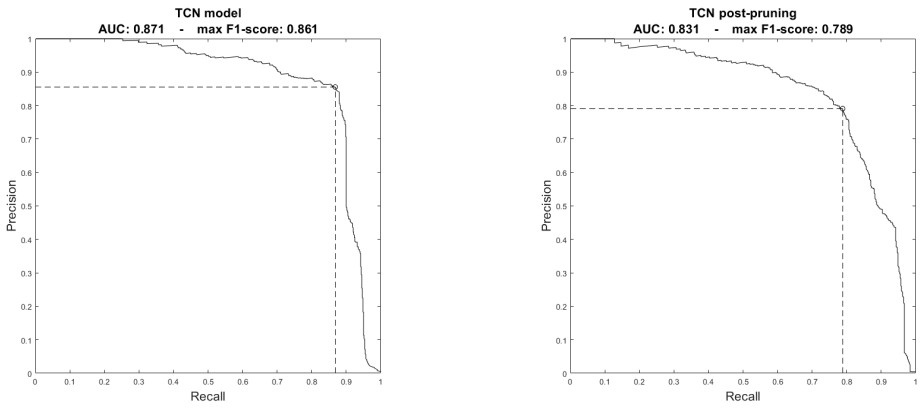


Figure 3.36: Output of layer 1 after pruning.

From Figure 3.37 we can look at the performance drop caused by pruning. By avoiding the deletion of some weights we could have probably obtained higher results, but still the pruned model had higher AUC and Max F1 score values compared to the recurrent and the mixed models.



(a) Original TCN **(b)** Pruned TCN

Figure 3.37: Precision-Recall curve of the pre- and post- pruning models.

3.5.3 Possible improvements

If we analyze the kernel of the first layer, shown in Figure 3.38, we can see that only a section of each kernel is actually important for the task, while the rest has values close to 0.

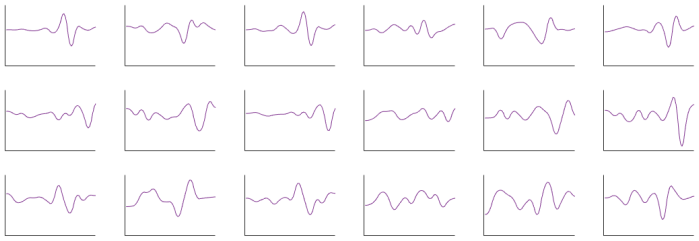


Figure 3.38: Kernels of layer 1 of the pruned model.

Weight pruning applied to subsection of kernel could further improve the general performance. In this context, I was not able to test this implementation since convolutional models in Matlab must be the same in size. If we were to implement this system on a chip, we could extract only sections of kernels that are important, using, for instance, a moving magnitude window.

3.6 Analysis of the models with cross-validation

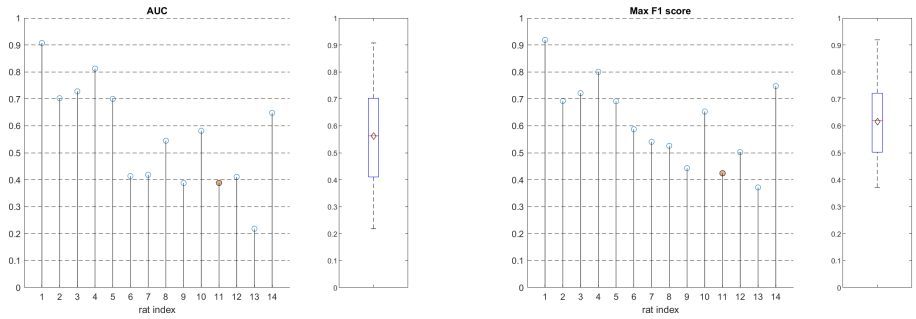
A final cross-validation of the models presented in Section 3.3 was performed. Complete cross-validation should be performed with a different model selection for each fold. Due to the time constraint, the decision was made to always use the architecture found in the model selection presented in Section 3.2. A 14-fold cross-valiation was used, where the data of each rats are used as test set in one fold and for training in the rest of the folds, in a *leave-1-out* approach. This should return the general efficacy

of the models in terms of detecting the EP on the data from each rat. By combining the weights of the different folds of the same linear models, we are also able to understand what channel or frequency component is used. To calculate the importance of a channel or frequency component, we can use the average weight magnitude of each kernel.
As for the previous comparison, only complete un-segmented recordings will be used to compute AUC and F1. The rat 11 used as test set until now is marked in orange in the various Figures present in this section.

3.6.1 Baseline models

Using LFP signal

As for the test set, this model has a good ability to detect EP, but cannot distinguish them from spontaneous activity. The index 1 rat recordings have a strong average response and little spontaneous activity. In this case, the model can find almost all EPs. A clear trend can be seen in figure 3.39 in both AUC and Max F1. Data from rats with a stronger average evoked response are easier to label. This trend is present in the results of all the models. This suggests that the intensity of the average response is a good metric to extract a good dataset as computed in Section 2.5.3.



(a) max F1 crossvalidation. **(b)** AUC crossvalidation.

Figure 3.39: Performance of linear model on LFP.

From the L1 of the weight of the convolutional layer, shown in Figure 3.40, we can see which channels were used by this model. Primarily, the linear model uses the channels from 1 to 12, linearly mixes them, and puts a threshold. The most important channels seem to be the channels from 1 to 7. Here we find the strongest response with less spontaneous activity.

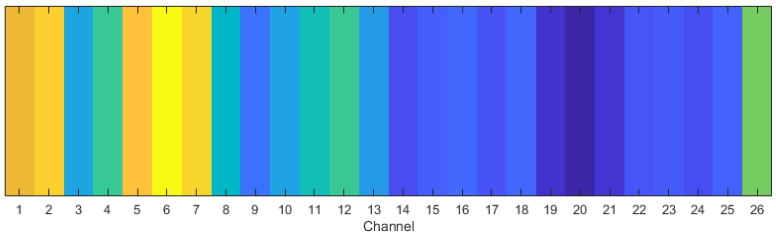
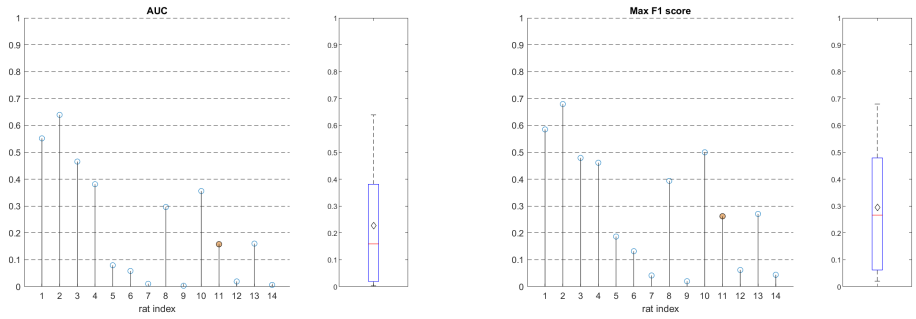


Figure 3.40: Magnitude of the kernels of the LFP linear model.

Using MUA signal

As discussed in the test results, we can only use MUA when we are sure that the needle is perfectly placed. On almost half of the recordings, the model cannot detect 10% of the EP.



(a) max F1 crossvalidation. **(b)** AUC crossvalidation.

Figure 3.41: Performance of linear model on MUA.

The importance of the input channels is shown in Figure 3.42. Differently from other models, this has his most important kernels on the channels 18-20. These channels are expected to touch the Layer 4 of the barrel cortex. Thus, as expected, the model finds a strongest neural activity in that section. The second section with more activation seems to be the Layer 5a, from channel 10 to 17.

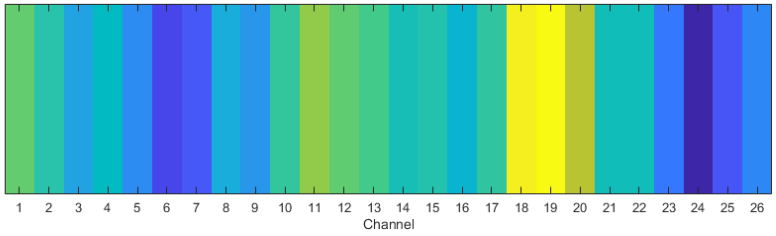
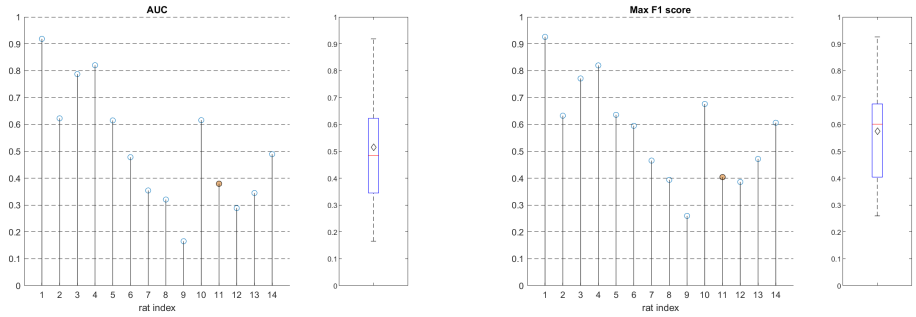


Figure 3.42: Magnitude of the kernels of the MUA linear model.

LFP frequency bands

As for the testing, there is little difference between the results of this model, presented in Figure 3.43, and the results of the linear model on LFP.



(a) max F1 crossvalidation. **(b)** AUC crossvalidation.

Figure 3.43: Performance of linear model on LFP frequency bands.

The model extracts most of the information from the higher frequency bands of channel 1-5. There seems to be no correlation between theta and alpha activity and evoked response.

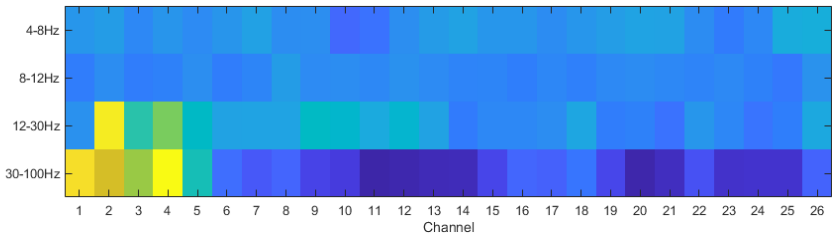
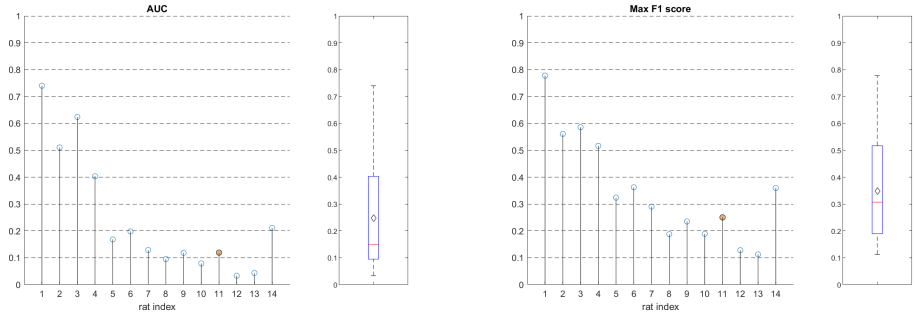


Figure 3.44: Magnitude of the kernels of the LFP frequency bands linear model.

CWT

As for testing, the CWT model is the second worst in terms of performance. From the results, shown in Figure 3.45, we can see that the model has worse AUC and F1 on the data from each rat.



(a) max F1 crossvalidation.

(b) AUC crossvalidation.

Figure 3.45: Performance of linear model on CWT.

From Figure 3.46 we can see a main group of channels and frequencies that is being used. Channels 1-10, which should correspond to layers 6 and 5b, are the most used by the models, with the frequency range 18-100Hz being the most important. There is a strong peak around frequency 30Hz and channels 4-6, which can be read as high beta activity on layer 5b.

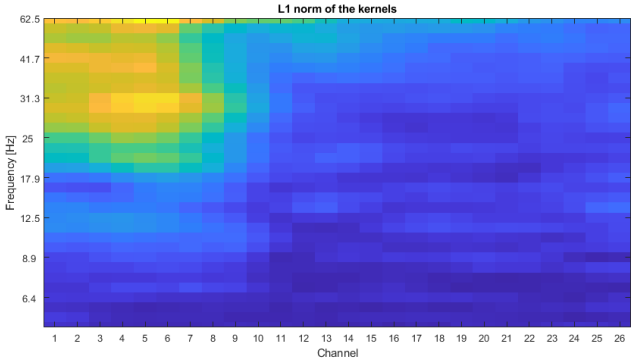
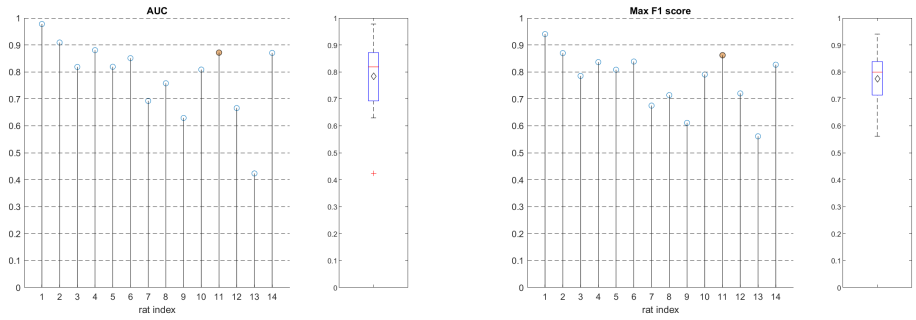


Figure 3.46: Magnitude of the kernels of the CWT linear model.

3.6.2 Deep Models

Temporal Convolutional Model

The performance of TCN is shown in Figure 3.47. The model can find most of the evoked responses even in the worst recordings. On recordings with clear responses, the model has results that are comparable to the baseline. The major improvement comes when comparing the worst recordings. If we take rat of index 11-12-13, we have an average maximum F1 of 0.4 in the best baseline while we have an average of 0.7 with this model.

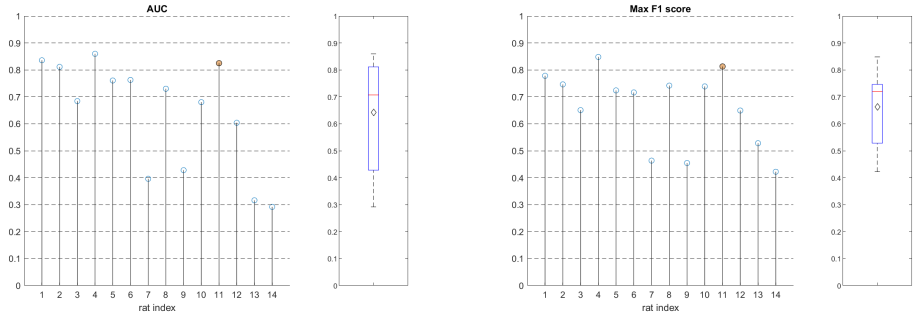


(a) max F1 crossvalidation. **(b)** AUC crossvalidation.

Figure 3.47: TCN model performance.

Recurrent Model

Performance of the recurrent model is shown in Figure 3.48. For some of the rats, the model was unable to find most of the responses. On the TCN, no recording has F1 lower than 0.5, in the recurrent case, we can find 4 that are below this value. In some cases, such as for the rat with strongest response, this model performs worse than the LFP linear model baseline.

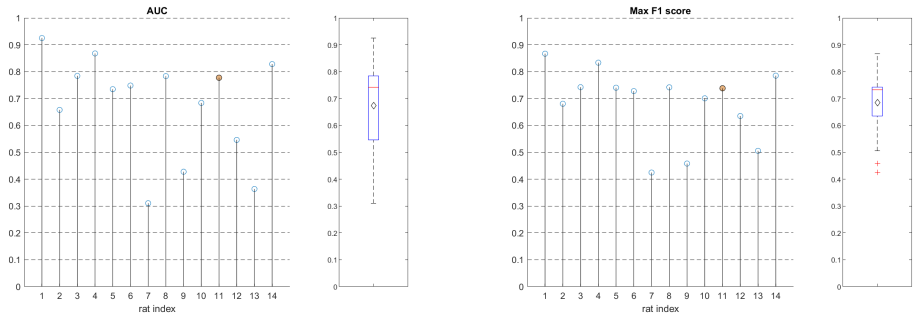


(a) max F1 crossvalidation. **(b)** AUC crossvalidation.

Figure 3.48: Recurrent model performance.

Mixed Model

Performances of the mixed model are shown in Figure 3.49. The mixed model performed slightly better than the recurrent model. The model is still far worse than the TCN, but has better capabilities compared to the recurrent and the baseline models.

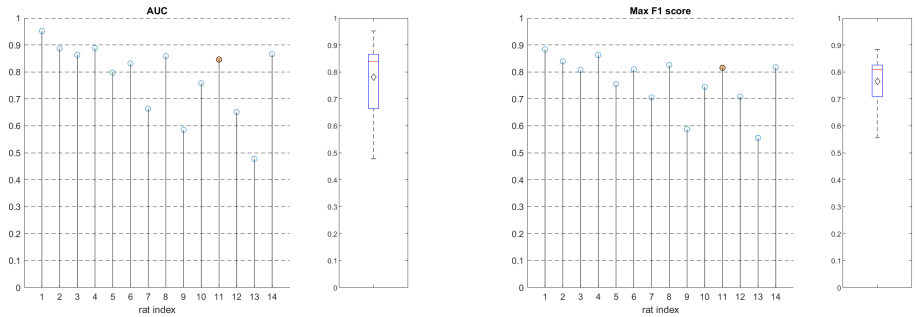


(a) max F1 crossvalidation **(b)** AUC crossvalidation

Figure 3.49: Mixed model performance.

Ensemble Model

Performance of the ensemble model is shown in Figure 3.49. The ensemble model returned results similar to those of the convolutional model. We can make the same consideration as made in the test section that the recurrent and mixed model finds a subset of the EPs found by the TCN.

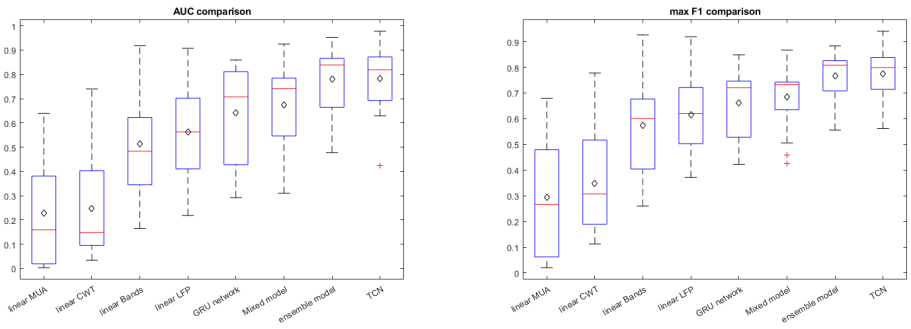


(a) max F1 crossvalidation. **(b)** AUC crossvalidation.

Figure 3.50: Ensemble model performance.

3.6.3 Comparison

A more complete comparison can now be done between all models. From Figure 3.51 we can see how the various models performed. As for the first comparison, the TCN is the best model. TCN has the ability to find 78% of the total stimulation present in the data set, with a precision of 0.8. Follow the ensemble model, the mixed model, and the recurrent model. In this case, the efficacy of both recurrent and mixed models, even if higher, is comparable to the linear model on LFP data.



(a) AUC. **(b)** maximum F1-Score.

Figure 3.51: Comparison of the performance of each model using cross-validaiton

.

The cross-validation confirms what we concluded in the previous comparison made using only index 11 rat.
A more complete final comparison is given in Table 3.11.

Model	AUC	Max F1	Precision	Recall
Linear on LFP	0.56±0.13	0.62±0.10	0.60±0.12	0.66±0.08
Linear on LFP Bands	0.51±0.12	0.57±0.14	0.60±0.12	0.57±0.15
Linear on MUA	0.23±0.17	0.29±0.20	0.32±0.18	0.35±0.22
Linear on CWT	0.24±0.15	0.35±0.15	0.30±0.16	0.50±0.15
TCN	0.78±0.08	0.77±0.05	0.77±0.05	0.78±0.06
GNU	0.64± 0.18	0.66±0.12	0.67±0.10	0.66±0.12
mixed	0.67±0.11	0.68±0.04	0.70±0.05	0.68±0.04
ensemble	0.78±0.09	0.77±0.05	0.78±0.06	0.75±0.06

Table 3.11: Complete comparison of the models.

Conclusion

The last chapter provides a brief account of the work that was performed with an overview of the methods used and the results obtained, as well as their significance for future researches.

Overall summary

Starting from multiple recordings extracted from the barrel cortex of several rats, I was able to pre-process and filter the data to create a data set that can be used in a supervised learning scenario. The data have first been filtered out of 18 bad recordings from 11 different rats, those with weak evoked responses. From the remaining recordings, extracted from 30 experiments on 14 different rats, I proceeded with extracting the LFP and MUA features. The average response of these two features was analyzed to create labels to use as output for the models. A window of 40 ms post-stimulus was labeled as the evoked response. Subsequently, the data was segmented into multiple windows centered around the stimulation. This is the final data set used in the model selection and learning process. For the final evaluation, only complete uncut recordings were taken into account to simulate real-time use.
An initial preliminary analysis was performed to design the model selection process. In this analysis, various papers and similar projects were examined to define the set of deep learning layers that could be used in the model. Therefore, I came to the conclusion that LSTM, GRU, convolutional layers, and temporal convolutional layers, used in different contexts and versions, are well suited to work on LFP data. Furthermore, I concluded that the normalization layer, gradient clipping, and dropout are fundamental components needed for a model to work correctly in this context.
In addition, an output block was designed to be used on each model. This output uses a single convolutional filter to shift the window of the evoked potential and overcome the delay introduced by each model. The sigmoid activation was selected as the output activation and the weighted binary cross-entropy was selected as the loss.
To virtually increase the size of the data set and prevent overfitting, a data augmentation layer was designed, which virtually "shifts" the probe to simulate the offset of different recordings. We cannot be sure that a channel touches the same layer of the cortex in each recording. However, by using data augmentation, the problem is overcome.
The selection of models was carried out using data from 13 rats as training and validation sets. Three models were selected: one that uses temporal convolutional layers, one that uses GRU layers, and one that uses GRU and convolutional layers. The mixed model used convolutional layers and not temporal convolutional layers for a problem encountered in the implementation. This problem did not affect the efficacy

of the model, but only the efficiency. A final ensemble of the three models was added for comparison.
A set of linear models was used as a baseline for analysis. These models use a linear convolutional layer on various features of the input. The features that I selected for this scope were the raw LFP, the MUA, the main frequency bands of the LFP, and the continuous wavelet transform of the LFP.
The test set was used on all models and a comparison of efficacy and efficiency was performed. Deep learning techniques outperformed the linear model baseline in the evoked potential detection task. For this task, the best type of architecture was the temporal convolutional network, followed in order by the ensemble, the mixed model, and the GRU network; in terms of efficiency, the best was the GRU layer followed by the mixed model, the TCN model and the ensemble.
Given the optimal efficacy of the temporal convolutional model, I decided to improve the performance aspect with a pruning pipeline. For this step, I tested both weight and gradient magnitude pruning. The gradient magnitude pruning on the filter of the convolutional network improved the efficiency of the model by a factor of 10x, by removing 92% of the weights with a small loss in efficacy. This final model is fast and accurate and is well suited for a real time usage.
At last, a cross-validation analysis was performed on the full data set using the model found by model selection. Cross-validation results are consistent with the results obtained using the test set previously defined.
To summarize the results, we concluded that all deep learning models that were tested are valid techniques to be used in real-time detection of the evoked potential. The temporal convolutional model performed the best in both efficiency and efficacy. The model is able to detect up to 80% of the evoked responses in the best cases with little to no false positives, using less than 1000 parameters.

Future work

Although this thesis work demonstrates the efficacy of the deep learning models used, possible improvements could be achieved by testing a wider range of models. Due to time and computational constraints, only a small subset of possible models was selected using a model selection process. Both deeper recurrent and deeper convolutional networks would probably increase the efficacy even more. For what regards mixed models, other alternatives could be tested, for instance, the use of convolutional layers first, followed then by recurrent layers.
Further studies could focus on the use of more labels to detect other types of potentials, such as the various forms of spontaneous activity, or to predict the intensity of the stimulations. Having the ability to solve these types of tasks could be helpful in analyzing various types of neural recordings.
In addition, the models created in this work could be helpful to detect the evoked potential of recordings extracted from free-moving rats. In this case it would be hard to detect when a whisker is spontaneously stimulated, however, with the ability of my models to extract this information, more complex experiments could be performed. This would be really useful for close-loop neural prostheses, since they could require real-time detection of the brain response to an event.

Bibliography

[1] S K Saha and B Sikder. Basic neural units of the brain: Neurons, synapses and action potential. *Journal of Neurology & Neuroscience*, 10, 2019.

[2] J M Kowalski and T Rabinowitz. Neuroanatomy, neuron action potential. *StatPearls [Internet]*, 2022.

[3] B Hille. *Ion channels of excitable membranes*. Sinauer Associates Sunderland, MA, 2001.

[4] Rachel Aronoff, Ferenc Matyas, Celine Mateo, Carine Ciron, Bernard Schneider, and Carl C H Petersen. Long-range connectivity of mouse primary somatosensory barrel cortex. *European Journal of Neuroscience*, 31(12):2221–2233, 2010.

[5] Laurens WJ Bosman, Arthur R Houweling, Cullen B Owens, Nouk Tanke, Olesya T Shevchouk, Negah Rahmati, Wouter HT Teunissen, Chiheng Ju, Wei Gong, Sebastiaan KE Koekkoek, et al. Anatomical pathways involved in generating and sensing rhythmic whisker movements. *Frontiers in integrative neuroscience*, 5:53, 2011.

[6] Michael M Morgan, MacDonald J Christie, Thomas Steckler, Ben J Harrison, Christos Pantelis, Christof Baltes, and Malcolm Lader. Multiunit activity. In *Encyclopedia of Psychopharmacology*, pages 809–809. Springer, 2010.

[7] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[9] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[10] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark, 2022.

[11] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[12] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[14] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[15] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.

[16] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[17] Yitian Chen, Yanfei Kang, Yixiong Chen, and Zizhuo Wang. Probabilistic forecasting with temporal convolutional neural network, 2020.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[19] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[20] Jesse M Lasky, Katherine R Kirby, and David N Koons. The area under the precision-recall curve as a performance metric for rare binary events. *Ecological modelling*, 222(13):2199–2207, 2011.

[21] Steven L Bressler and Mingzhou Ding. *Event-related potentials.* Wiley Encyclopedia of Biomedical Engineering, 2006.

[22] Hafeez Ullah Amin, Wajid Mumtaz, Ahmad Rauf Subhani, Mohamad Naufal Mohamad Saad, and Aamir Saeed Malik. Classification of eeg signals based on pattern recognition approach. *Frontiers in Computational Neuroscience*, 11, 2017.

[23] Amjed S. Al-Fahoum, Ausilah A. Al-Fraihat, M. S. Oliveira, A. Grant, and J. A. Hinojosa. Methods of eeg signal features extraction using linear analysis in frequency and time-frequency domains. *ISRN Neuroscience*, 2014:730218, 2014.

[24] Hamid R. Mohseni, A. Maghsoudi, and Mohammad B. Shamsollahi. Seizure detection in eeg signals: A comparison of different approaches. In *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, volume Supplement, pages 6724–6727, 2006.

[25] Saurabh Kumar, Arindam Ghosh, Rajarshi Mondal, Soumya Sarkar, and Suman Chakraborty. Wavelet decomposition of intracortically recorded signals for extracting stable neural features. *Bioelectronic Medicine*, 4(1):11, 2018.

[26] M. Kumar, S. Kumar, and S. Kumar. A review of intrusion detection system based on machine learning techniques. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 1145–1149, 2015.

[27] Jinwon An and Sungzoon Cho. Hand motion identification of grasp-and-lift task from electroencephalography recordings using recurrent neural networks. In *2016 International Conference on Big Data and Smart Computing (BigComp)*, pages 427–429, 2016.

[28] Nur Ahmadi, Timothy G. Constandinou, and Christos-Savvas Bouganis. End-to-end hand kinematic decoding from lfps using temporal convolutional network. In *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4, 2019.

[29] Xin Dong, Shangyu Chen, and Sinno Jialin Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon, 2017.

[30] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015.

[31] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning, 2019.