NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Event Detection and Classification of in-vitro LFP Electrophysiological Signals with Deep Learning

Nikolaos C. Antoniadis

Supervisors: **Irini Skaliora,** Professor, DHPS
Izampo Karali, Assistant Professor, DIT

ATHENS

MARCH 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανίχνευση και Κατηγοριοποίηση Γεγονότων σε στο-γυαλί ΔΤΠ Ηλεκτροφυσιολογικά Σήματα με Βαθιά Μάθηση

Νικόλαος Χ. Αντωνιάδης

Επιβλέποντες: **Ειρήνη Σκαλιώρα,** Καθηγήτρια, ΤΙΦΕ
Ιζαμπώ Καράλη, Επίκουρη Καθηγήτρια, ΤΠΤ

ΑΘΗΝΑ

ΜΆΡΤΙΟΣ 2020

BSc THESIS

Event Detection and Classification of in-vitro LFP Electrophysiological Signals with Deep Learning

Nikolaos C. Antoniadis

S.N.: 1115201100052

SUPERVISORS: **Irini Skaliora,** Professor, DHPS
Izampo Karali, Assistant Professor, DIT

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανίχνευση και Κατηγοριοποίηση Γεγονότων σε στο-γυαλί ΔΤΠ Ηλεκτροφυσιολογικά Σήματα με Βαθιά Μάθηση

Νικόλαος Χ. Αντωνιάδης

Α.Μ.: 1115201100052

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Ειρήνη Σκαλιώρα,** Καθηγήτρια, ΤΙΦΕ
Ιζαμπώ Καράλη, Επίκουρη Καθηγήτρια, ΤΠΤ

ABSTRACT

In Neuroscience, studying complex cognitive functions, such as attentional modulation, motor control or short-term memory, relies on understanding the circuit dynamics that underlie neuronal systems. Researchers need to observe and classify network events in synchronized brain activity, that's characterised by alternating epochs of massive persistent network activity and periods of generalized neural silence, a tedious task when performed by a human expert. A popular method for recording network activity is Local Field Potential (LFP), as it allows for long and stable recordings from multiple sites.

In Computer Science, state-of-the-art hardware combined with the abundance of large scale datasets, in recent years, have given rise to Deep Neural Networks (DNNs), a powerful computational model used by Deep Learning methods, that was able to approach, or even surpass, human level performance on various tasks that rely on pattern recognition, such as speech recognition and medical diagnosis.

In this thesis, we explore if Deep Learning methods can amplify the accuracy of determining the timing and duration of network events in neuronal systems, by processing in-vitro LFP events. Particularly, we implement a variation of Dreem One Shot Event Detector (DOSED), a deep learning approach that jointly predicts locations, durations and types of events in EEG time series. In our approach the input to the network is an LFP time series instead. Then we compare the performance of this method to LFPAnalyzer, which is based on the implementation of established signal processing and non-deep machine learning approaches, a method that has already been shown to perform better than semi-manual analysis. Both methods are fully automated and depend solely on data.

SUBJECT AREA: Signal Processing, Machine Learning, Computational Neuroscience
KEYWORDS: LFP, neural networks, deep learning, classification, object detection.

ΠΕΡΙΛΗΨΗ

Στη Νευροεπιστήμη, η μελέτη σύνθετων γνωστικών λειτουργιών, όπως η προσαρμογή εστίασης, ο έλεγχος των κινήσεων ή η βραχυπρόθεσμη μνήμη, βασίζεται στην κατανόηση της δυναμικής του κυκλώματος που υπόκειται των νευρικών συστημάτων. Οι ερευνητές πρέπει να παρακολουθούν και να ταξινομούν γεγονότα δικτύου σε συγχρονισμένη εγκεφαλική δραστηριότητα, που χαρακτηρίζεται από εναλλασσόμενες εποχές μαζικής επίμονης δραστηριότητας δικτύου και περιόδους γενικευμένης νευρικής σιωπής, το οποίο αποτελεί ένα κουραστικό έργο, όταν εκτελείται από έναν ανθρώπινο εμπειρογνώμονα. Μια δημοφιλής μέθοδος για την καταγραφή δραστηριότητας δικτύου είναι το Δυναμικό Τοπικού Πεδίου (LFP), καθώς επιτρέπει σταθερές εγγραφές, μεγάλης διάρκειας, από πολλαπλές τοποθεσίες.

Στην επιστήμη των υπολογιστών, οι τεχνολογικές εξελίξεις στα εξαρτήματα υπολογιστών σε συνδυασμό με την αφθονία των μεγάλης-κλίμακας βάσεων δεδομένων, τα τελευταία χρόνια έχουν οδηγήσει στη ραγδαία εξέλιξη των Βαθέων Νευρωνικών Δικτύων (DNN), ένα ισχυρό υπολογιστικό μοντέλο που χρησιμοποιείται από τις μεθόδους Βαθιάς Μάθησης, το οποίο έχει προσεγγίσει ή ακόμα και ξεπεράσει την απόδοση του ανθρώπου σε διάφορες εργασίες που βασίζονται στην αναγνώριση προτύπων, όπως η αναγνώριση ομιλίας και η ιατρική διάγνωση.

Σε αυτή τη διατριβή, διερευνούμε αν οι μέθοδοι Βαθιάς Μάθησης μπορούν να ενισχύσουν την ακρίβεια του προσδιορισμού του χρονισμού και της διάρκειας των γεγονότων δικτύου σε νευρικά συστήματα, επεξεργαζόμενοι στο-γυαλί LFP γεγονότα. Συγκεκριμένα, εφαρμόζουμε μια παραλλαγή του ανιχνευτή γεγονότων Dreem One Shot Event (DOSED), μια βαθιάς μάθησης προσέγγιση, που προβλέπει ταυτόχρονα τις θέσεις, τις διάρκειες και τους τύπους συμβάντων στις χρονοσειρές του EEG. Στην προσέγγισή μας, η είσοδος στο δίκτυο είναι μια χρονική σειρά LFP. Στη συνέχεια, συγκρίνουμε την απόδοση αυτής της μεθόδου με τον LFPAnalyzer, ο οποίος βασίζεται στην εφαρμογή καθιερωμένων προσεγγίσεων επεξεργασίας σήματος και μη βαθιάς μηχανικής μάθησης, μιας μεθόδου που έχει ήδη αποδειχθεί ότι έχει καλύτερη απόδοση από την ημι-χειρωνακτική ανάλυση. Και οι δύο μέθοδοι είναι πλήρως αυτοματοποιημένες και εξαρτώνται αποκλειστικά από δεδομένα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Επεξεργασία Σήματος, Μηχανική Μάθηση,
Υπολογιστική Νευροεπιστήμη

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: LFP, νευρωνικά δίκτυα, βαθιά μάθηση, κατηγοριοποίηση, ανίχνευση αντικειμένου

I want to thank every professor from school to university,

who shared their passion with me,

leading me to find my own.

This is for you.

ACKNOWLEDGMENTS

I would like to thank my supervisor Professor Irini Skaliora, for giving me the opportunity, to work on a Computational Neuroscience project and welcoming me to her lab

I would also like to thank PhD candidate Michael Vinos, who was always there to guide me through it and offer his insight.

Lastly, I would like to thank everyone from Skaliora's lab, for providing the essential LFP data I needed for the experimental part of this thesis, as well as Chambon and his team for creating their genuine method, that successfully tackles this difficult problem.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This thesis was written as part of the BSc in Computer Science program at the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens, in collaboration with Skaliora's Electrophysiology lab at the Biomedical Research Foundation of the Academy of Athens.

1. INTRODUCTION

1.1. Background

On this project, we experiment with Deep Learning techniques, particularly of the field of Computer Vision, and apply them on a Signal Processing problem of Electrophysiological signals analysis, particularly LFP signals.

1.1.1. Signal Processing

In order to better understand a phenomenon, we often study signals, i.e. functions that quantify and model the behaviour of a system[1]. In that regard, signals can convey various types of information, such as audio, video, speech, image, sonar or electrochemical information, among others[1]. But the relative information of a signal, is often accompanied by non-relative noise, obscuring the study of a phenomenon.

In that context, we can define noise as random fluctuations of data, that perpetrate the obstruction of observing the original signal and the information it holds [2]. One of the goals of Information Theory, a branch of mathematics that examines the transmission of information, is solving the problem of retrieving the useful information from a noisy observation [3].

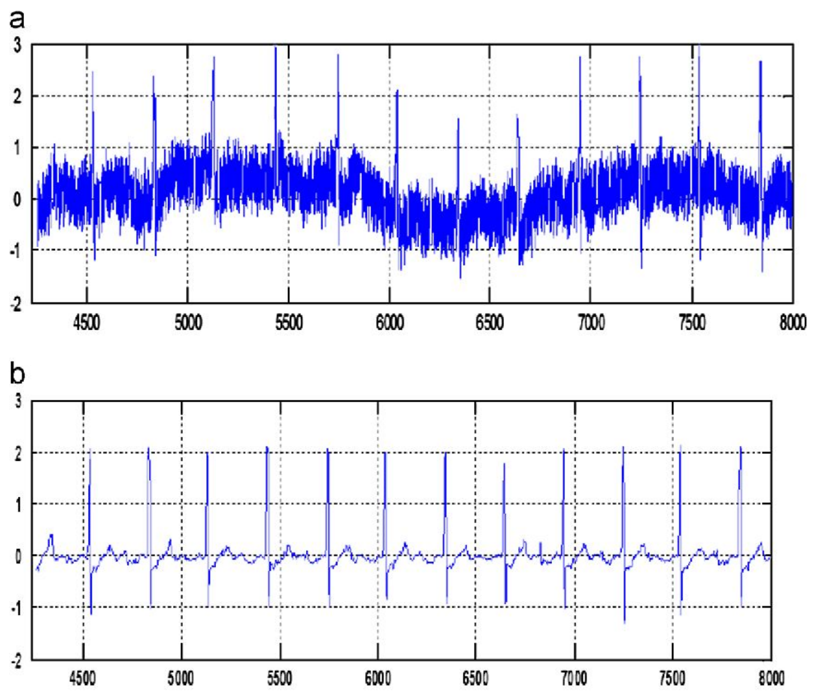


Figure 1.1: Graphical representation of a signal transmitted through a noisy channel (a) and the same signal without noise as a result of signal processing (b)[4]

Over the years, various methods have been established to process signals and extract the necessary compounds to analyze the phenomenon in question. But choosing the best suited method, starts by identifying the properties of the signal.

1.1.2. Electrophysiological Signals and Event-Related Potential

Electrophysiology is a subfield of Physiology that examines the electrical properties of biological tissues and cells. So, in general, electrophysiological signals are functions of voltage fluctuations over a period of time, and in Neuroscience, these signals hold information over the electrical activity of neurons.

In the 1950s, a new approach started coming to prominence, gaining influence over theories and research of human cognition. With this “information processing” approach, cognitive functions can be studied through the patterned neural activities, translated as informational transactions in the brain. These transactions are reflected in “event-related potentials” (ERPs), meaning that an increase in the electric field potential of a neuron, or a group of neurons, can be linked to certain events in the brain, who in turn can be linked to certain cognitive functions.[5]

An ERP waveform contains positive and negative values representing voltage fluctuations that can be further analysed into a set of underlying components.[6] Furthermore, the waveform can be divided into the signal of interest, i.e. a series of ERPs and the accompanying noise. This noise is the sum of random background brain activity, called neural oscillations, bio-signals and even electromagnetic interference caused by external factors, such as equipment used in the lab during the recording of the electrophysiological signal.

In that context, in order to further analyse cognitive functions, one needs an electrophysiological monitoring method that evaluates the electrical activity in the brain to obtain electrophysiological signals and one signal processing method that extracts relative information in the form of ERPs.

1.1.3. Electroencephalogram

One of the most commonly used methods for measuring brain activity, is the Electroencephalogram (EEG), which is recorded through the use of electrodes attached to the scalp, and is used to diagnose various brain disorders, like epilepsy, brain tumors, brain damage, strokes or sleep disorders, among others.

EEG provides a graphic display of voltage changes from different sites across the brain and is mostly used non-invasively (extracranial EEG), while it can also provide in-vivo measurements from surgically implanted electrodes focused at specific regions of the brain (intracranial EEG).

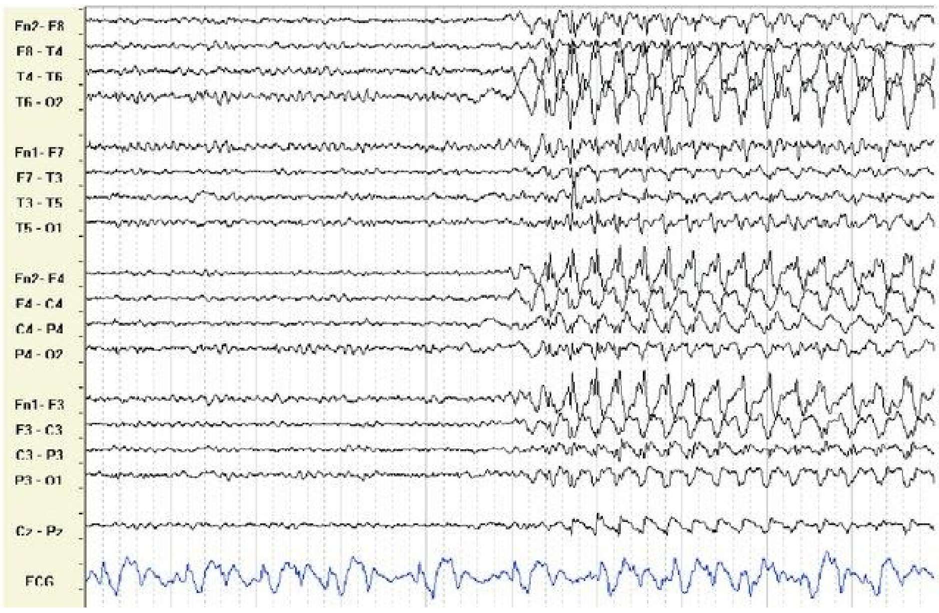


Figure 1.2: EEG data showing onset of an epileptic event.[7]

In the common case, human experts are the ones who visually inspect EEGs in search of ERPs, manually annotating the onset and offset of such events, as well as classifying them. But this raises the following issues: there is a high likelihood that annotations from different experts will vary significantly, and on top of that, the whole process is mundane, time consuming and costly on the experts' end.[8]

1.1.4. Local Field Potential

Another popular method for recording brain activity is the Local Field Potential (LFP; also known as intracranial EEG), which is more invasive than the extracranial EEG. To get an LFP recording, an extracellular microelectrode is set within brain tissue. The placement of the microelectrode should have sufficient distance from individual local neurons, in order to avoid dominance of the signal by a singular neural cell. This way, a great number of neurons can equally contribute to the signal that constitutes the potential generated by the total of all local currents on the surface of the microelectrode.

The rising prevalence of LFPs can be credited to a number of factors. The utilization of advanced microelectrode technology of the last decade, in the form of silicon-based multi-electrodes, has enabled researchers to reach extraordinary spatial coverage and resolution of the processes that amount to the generation of the extracellular field.[9] At a network level, integrative synaptic processes can be easily observed by researchers as LFPs provide different band-limited components and in addition, they provide useful information about the effect of neuro-modulatory pathways, local intra-cortical processing and the state of the cortical network.[10,11] LFPs usually provide a wide spectrum of neural oscillators on the range of 1-100Hz that reflect the activity of various neural processing pathways.[12] In neural processing, discrete and possibly disassociated information channels can be empirically examined through LFP signals[13], and even provide apprehension of neuronal activity generated by circuit dynamics, when combined with spike recordings.[14] Lastly, in-silico models of neuronal activity can be developed from information obtained through LFPs, that can later be compared to in-vivo experiments.[15]

LFP signals consist of alternating epochs of massive persistent network activity and periods of generalized neural silence, so the accurate determination of the timing and duration of those network events, in the form of the events' onset and offset, is crucial but requires great manpower and tedious work for the researchers.[16]

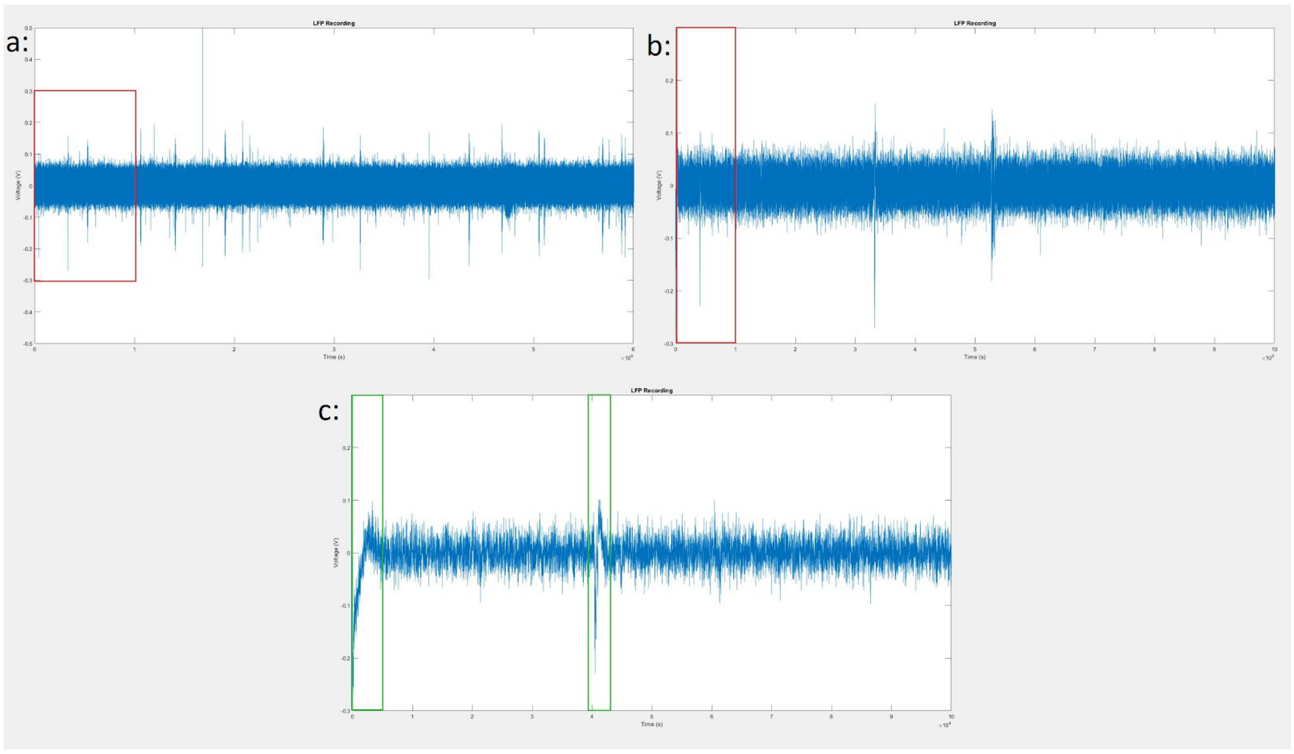


Figure 1.3: A single LFP recording in three different plots, in terms of time :

a) The original complete signal. The red box signifies the part that will be zoomed in, producing plot b.
b) The same signal as plot a, but zoomed in. The red box signifies the part that will be zoomed in, producing plot c.
c) The same signal as plot b, but zoomed in. The two green boxes signify the two events of interest. Anything out of the boxes is considered neural silence/noise.

Notice that the peaks of the events are distinguishable with the eye, even from plot a, but we need to see them closer from plot c, to find their onset and offset.

In conclusion, on top of the above-mentioned reasons for LFPs rising popularity, neuroscience researchers perceive an enormous increase of analysis load, due to the growing need for massively parallel data from multi-electrode arrays. As such, the requirement for reliable, automated and high throughput tools for detecting and quantifying neural events in LFP signals is of grave significance.[16]

1.1.5. Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence that concerns computer algorithms that learn how to solve a certain problem, without the programmer explicitly defining it or the approach to solve it. Instead, the computer learns how to solve the problem mainly from performing random actions at first, then evaluating said actions by comparing the generated results with data provided by the programmer, in order to select better future actions to get better results.

Machine Learning is mostly used on problems whose solution consists of so complex steps that the programmer is unable to clearly define them. In these cases, it is preferable and more effective to guide the machine into developing its own algorithm than actually providing one yourself. The three main categories of approaches of ML are supervised learning, unsupervised learning and reinforcement learning.

In supervised learning, a supervisor/expert provides already labeled data, in the form of example inputs and desired outputs, that will be used to train the algorithm to map every of these inputs to specific outputs. During training, the algorithm processes the training data and makes a prediction for each data instance. Then it compares its prediction, with the ground truth, i.e. the labeled data the expert provided. The evaluation of how wrong the prediction was, sets how the algorithm should shift, in order to produce a prediction that's going to be closer to the correct one in the future. After the end of the training period, the algorithm will attempt to make predictions about previously "unseen" labeled data instances, so that the programmer can infer how well the algorithm can generalize to new, and possibly unlabeled, data. This method of learning is parallel to psychology's concept learning and is widely used for problems of pattern recognition, aka classification and regression, aka function approximation.[17]

In unsupervised learning, the datasets are unlabeled and the supervision by an expert is minimal. The algorithm attempts to find undetected patterns and structure in the data by itself. It's usually applied to general estimation problems, such as clustering, statistical distribution estimation, compression and filtering or feature learning.[17]

Finally, in reinforcement learning, the algorithm deploys intelligent agents that are able to interact with a dynamic environment and the actions they apply either positively or negatively reinforce these actions in the future. The problem is modelled as a Markov decision process environment, and in it, the agents need to balance between exploration, in the sense of performing an action that hasn't been tried yet and exploitation, in the sense of repeating actions that previously yielded positive results. Reinforcement learning algorithms are widely used for control problems and other sequential decision making tasks, such as self driving vehicles, natural resource management, medicine design, as well as playing board games and video games.[17]

1.1.6. Artificial Neural Networks

Artificial Neural Networks (ANNs) are mathematical models capable of learning how to solve problems. Their inspiration comes from biological neural networks. As such, ANNs mirror both their composition, as well as the way they function to learn new complicated tasks. An ANN consists of a collection of artificial neurons, the neurons' connections, the connections' weights and a propagation function, i.e. a function that defines the output of each neuron. The basic idea is that the input to the algorithm is feeded to a subset of these neurons, who based on their connections, weights and propagation functions, each output a number, which then serves an input to the next set of neurons, and so on, until it reaches the final set of neurons, whose output is considered the output of the algorithm and the solution to the problem at hand.[17]

Much like their biological counterparts, each artificial neuron consists of a cell body, a nucleus, an axon, axon terminals and dendrites. The *input* to each artificial neuron, paired with individual *weights* to the incoming connections*,* can be considered as the biological neurons' dendrites. The inputs are combined with the *sum function Σ* in the cell body. Then the *propagation function f* , also called *activation function,* can be thought of as the computations that run also in the cell body, with the *bias*. Finally the computed *output signal* is transferred through the axon to the outcoming connection i.e. the axon terminals.

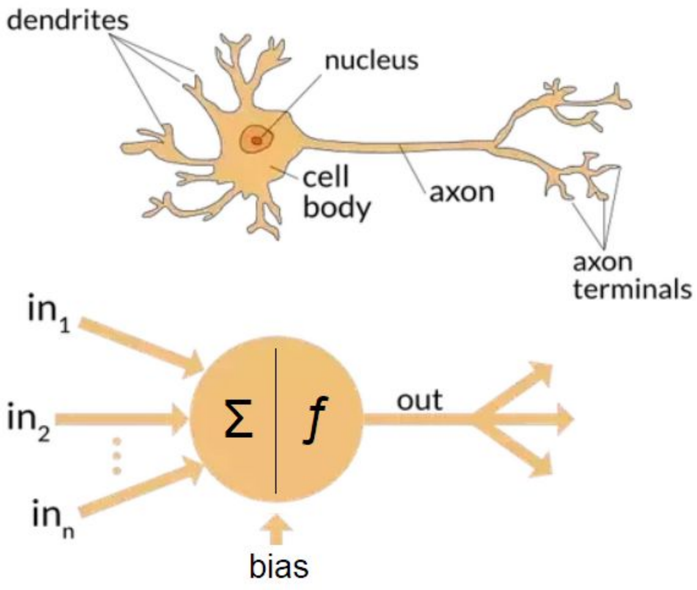


Figure 1.4: Graphical comparison of a biological neuron and an artificial one.[18]

The output of neuron$_j$, that has incoming connections from neuron$_0$ to neuron$_n$ can then be defined as:

$$output_j = f\left(\sum_{i=0}^{n} (weight_{i,j}\ output_i) + bias_j\right)$$

where output$_i$ is the input to neuron$_j$ from neuron$_i$, weight $_{i,j}$ is the weight of the connection of these two neurons, bias$_j$ is the bias of neuron$_j$ and f is the activation function.[17]

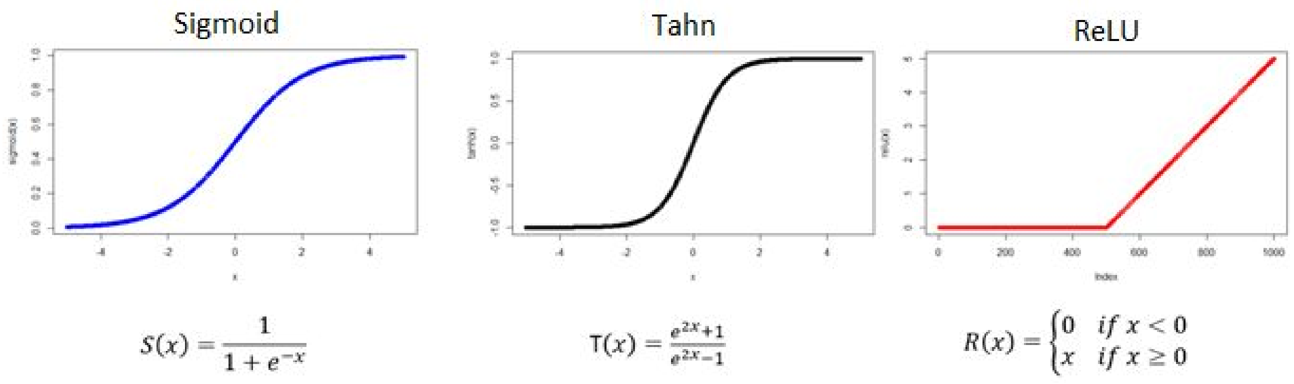


Figure 1.5: The three most commonly used activation functions are the sigmoid, tahn and ReLU functions.[19]

In practice, these artificial neurons are nodes of a computational graph, that can be fully connected or not, pending on the task the ANN is trying to learn. Before training the network, the weights and the biases of each node are randomly assigned. During training, under the machine learning paradigm, the network predicts outputs for the given training data. The network proceeds to evaluate its output and then changes the weights and biases following a learning rule, set by the programmer, attempting to get better predictions in the future. After successfully training, the network is ready to be used to perform the task it learned, by predicting output for new data instances.[17]

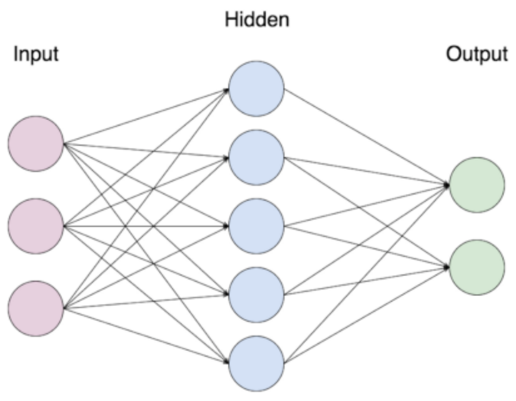


Figure 1.6: An ANN as computational fully-connected graph with three layers, one for input, one for output and a hidden layer.[20]

ANNs can be used with all three aforementioned machine learning approaches and in later years, after decades of not working efficiently, they have been proven to be extremely powerful. The main reasons for ANNs recent success are one side, new computer technology, that enables running computational graphs with millions of artificial neurons and more than one hidden layers, and on the other side the enormous size of datasets available today, that allow the creation of statistical models capable of solving tremendously complex problems. These two factors have led the way to the greatest revolution in computers of this century, that has been named Deep Learning.

1.1.7. Deep Learning and Deep Neural Networks

Deep Learning (DL) is a subfield of ML that uses deep neural networks (DNNs), meaning ANNs that have more than one hidden layers, to learn to perform tasks. In DL a complex task can be divided into simpler subtasks. Then each subtask can be assigned to one of the hidden layers to perform, making the network extremely efficient. This way, each layer is able to progressively extract higher level features from the input. For example, in Image Processing, the input layer of the network will receive the raw image represented as a matrix, where each element of the matrix corresponds to a pixel in the image. Then as the data flows towards the output layer, passing through the hidden layers, the process can be thought of as trying to reconstruct the contents of the image. As such, the first hidden layer could detect the dots in the picture and encode them as edges, the second could construct arrangements of edges, so that the third could compose more complex shapes and these shapes could be connected on the fourth layer to provide the form of an object, e.g. a dog, that would be detected and classified as such on the final output layer. This constitutes the greatest advantage over classic ML algorithms, as the programmer doesn’t need to define what a dog is, or how a dog looks. Instead, the hard choice of which features in the data are relevant to the task and which are not, is conducted by the network itself.[17]

Apart from Image Processing[21], DNNs have been used, yielding great results, on a number of fields, such as speech recognition[22], audio recognition[23], natural language processing[24], language translation[25], drug design[26], medical image analysis[27], bioinformatics[28], board[29] and video games[30], while in most of these areas they have even surpassed human experts.

Every DNN can be defined by its parameters, hyperparameters and strategies.

The number of the parameters, or coefficients of the model, of a DNN, are defined by its number of layers and the number of neurons each layer has. These are selected by the programmer based on the task the network attempts to learn. So the selection of the architecture of the network by the programmer, defines the selection of the number of parameters by the network itself. These parameters can be divided into the tr*ainable parameters*, meaning that their value changes during the training period, and the *non-trainable parameters*. Usually the trainable parameters are the weights of the connections between neurons and the neurons' biases.[17]

The hyperparameters of a DNN, are the parameters whose value is assigned by the programmer, before training, and they remain unchanged throughout. These include the *number of hidden layers*, the *learning rate*, i.e. how much do the trainable parameters change at every step of the training, the *momentum*, i.e. a technique that increases the speed of convergence by embedding memory of past changes on the learnable parameters, the *activation function* of the neurons on each layer, the *batch size*, i.e. the number of data instances the network processes before adjusting the trainable parameters, the *number of epochs*, i.e. the number of times the network is going to process the whole training dataset, before ending the training period, and more.[17]

The strategies used by a DNN, involve the *parameter initialization*, i.e. how the trainable parameters are going to be initialized before training, the *data normalization*, i.e. normalization or standardization of the input data before training, the *optimization algorithm*, i.e. the algorithm that computes how the trainable parameters should be adjusted in order for the the desired output to be produced, the *loss function*, i.e. a function that evaluates how far from the desired output, the network's actual output is, or in other words the error of the network, and more. [17]

To sum up, when a programmer wants to build a DNN that learns how to solve a certain problem, he needs to carefully select the hyperparameters and strategies of the DNN for it to be successful. These can be selected either by a priori knowledge of the problem or by trial and error.

1.1.8. Computer Vision

Computer Vision is the interdisciplinary field that pursues to automate various visual tasks the human visual system can perform and even surpass it. These tasks include acquiring, processing, analyzing, detecting and extracting useful information from digital images, so that they can be encoded into symbolic information that a computer can "understand", in order to select and perform appropriate actions.[31]

Computer Vision algorithms can be applied to numerous problems, for instance, event detection[32], video tracking[33], scene reconstruction[34], image restoration[35], object recognition[36], motion estimation[37], and many more. In these fields, Computer Vision has seen a drastic rise in efficiency, due to the utilization of DL and DNNs.

1.1.9. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of DNNs that are used for Computer Vision algorithms. Like the DNNs, they are also inspired by biological processes of animals, specifically by copying the organization of the connectivity of neurons in the visual cortex. In the visual cortex, specific neurons only respond to specific overlapping regions of the visual field, also called the receptive field. In this way, in a CNN each artificial neuron gets input from a specific area of the input image, while these overlapping areas can reconstruct the original input to the network, should they be combined. This enables each neuron to focus on a specific part of the image, making it easier to perform complex tasks on large images or videos. These small areas will be combined to larger areas going through the many hidden layers of a CNN, in order to capture spatial and temporal dependencies between the elements of the input image. Doing so, enables CNNs to learn which filters to apply to get the desired output, without the need of a human expert to hand-engineer them, as with classical machine learning algorithms.[38]

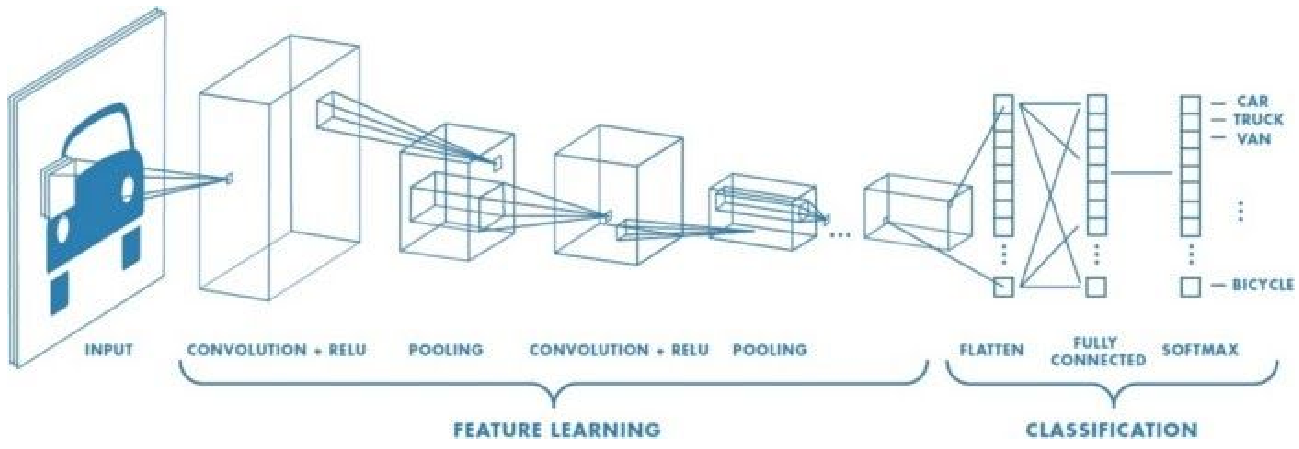


Figure 1.7: A CNN that learns how to detect and classify various types of vehicles.[39]

CNNs take their name from the application of convolution, the mathematical operation, on the areas of the input image. Like the DNNs they consist of one input layer, one output layer and more than one hidden layer. These hidden layers are divided into the convolutional layers, the ones that perform the convolution on the image and the non-convolutional layers, whose properties and number vary and are chosen based on the task the CNN attempts to learn to perform. Typically, the convolutional layers use the ReLU activation function, and are followed by pooling layers, fully-connected layers and normalization layers.[38]

The input of a CNN is in the form of a matrix, or otherwise called tensor, with size: number of data instances x image height x image width x image properties. The image properties can either be the different channels of a recording, the RGB colour channels, number of frames that will be processed together when the input is in video, or some other property. Passing through a convolutional layer, the input gets abstracted to a feature map, whose size would be : number of data instances x feature map height x feature map width x feature map properties. [38]

The feature map height and width depend on the individual kernel size of the individual convolutional layer, which are hyperparameters of the CNN. The kernel of a convolutional layer, is the filter that will be applied on the layer's input and has the form of a matrix. As the kernel matrix, "slides" on the image input matrix of the convolutional layer, it produces the aforementioned feature matrix as the output to the layer, that will serve as the image input matrix to the next convolutional layer. [38]

The way the kernel matrix "slides" is defined by the stride number. This is the number of pixels the kernel matrix will shift over the input matrix. The combination of kernel size and stride define the size of the output matrix.[38]

These can be fully demonstrated and further explained on the image below:

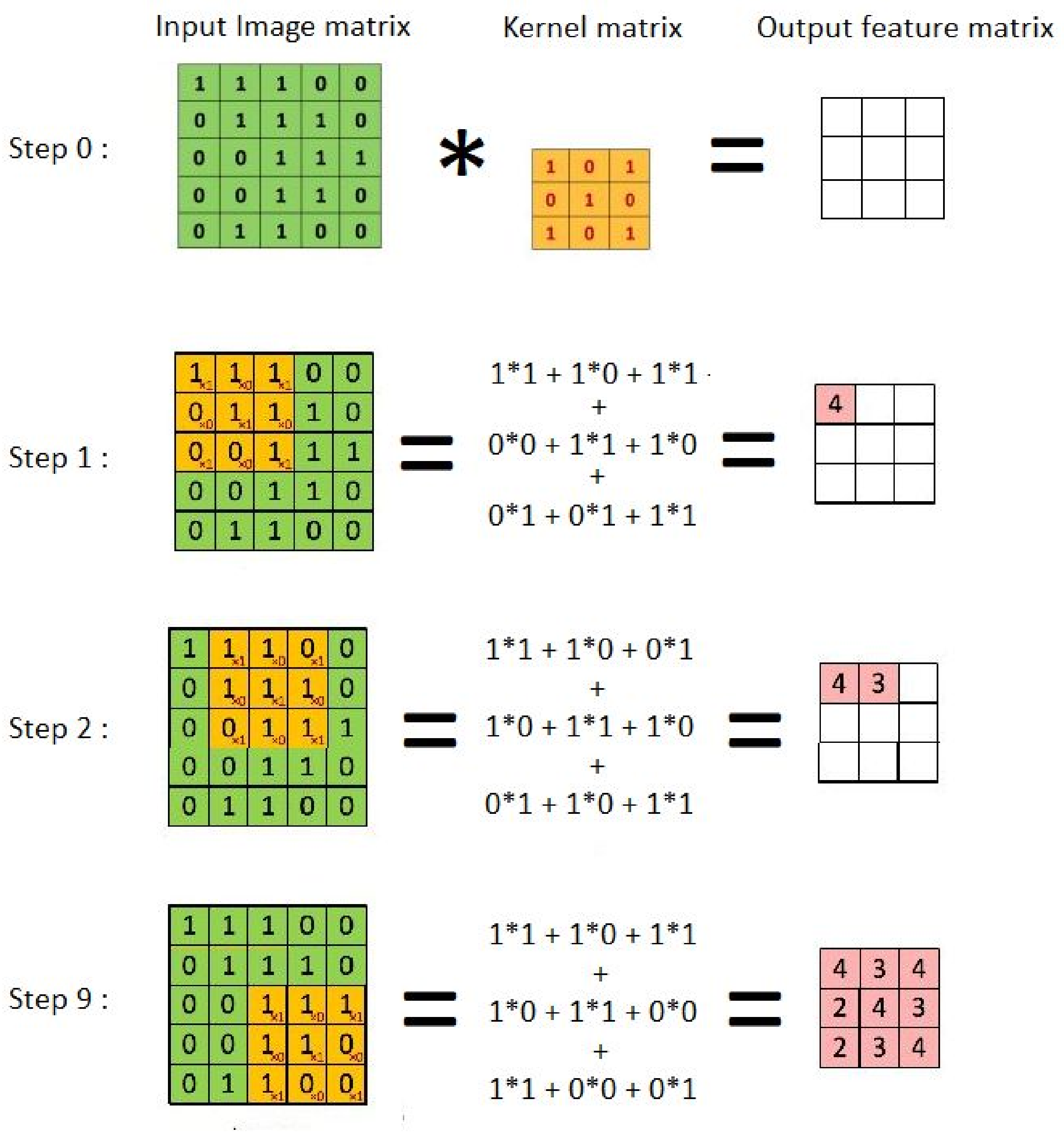


Figure 1.8: The steps of a convolution of an input image matrix of size (5x5x1) at a convolutional layer with kernel size (3x3x1), producing the output feature matrix of size (3x3x1), with strides=1.

The output of the convolutional layer goes through a ReLU layer, which passes forward only the positive values to the pooling layer. The pooling layer's purpose is two-fold: to simplify the output of the previous layer and to reduce its size. There are three main pooling layers, the max pooling, the average pooling and the sum pooling layer, all of which perform to their input the operation in their name to produce their output. So a max pooling layer, with kernel size (height x width) will perform a max operation that will produce an output of size (height x width).[38] The effects of the pooling layer are better showcased in the image below:

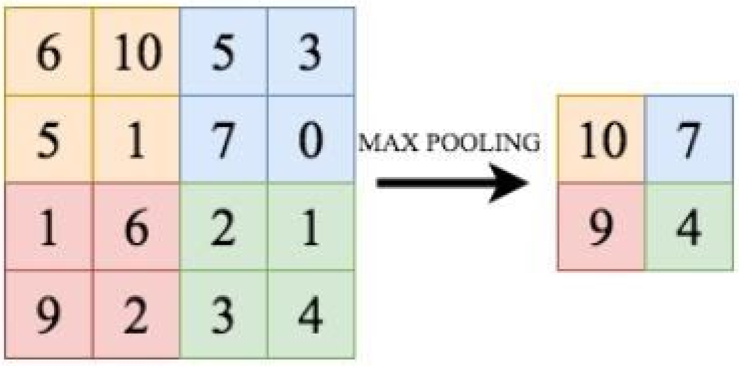


Figure 1.9: Max pooling with kernel size (2x2) and stride=2, applied to a (4x4) matrix to produce a (2x2) output matrix.[38]

Following the last convolutional and pooling layers pair is a set of one or more fully-connected layers, i.e. layers whose input and output are fully-connected with their nodes. The purpose of these are to reduce the size of the input matrix to a vector with elements equal to the number of classes available for classification.

Finally, on the last layer, the softmax activation function is applied, which belongs to the sigmoid activation functions family mentioned before. The softmax returns a vector of probabilities that all sum to one, with ‘x’ element of the vector representing the probability of the input to the network to belong to the class numbered ‘x’.[38]

Now we have fully explained how a feed-forward of one input data instance at a CNN used for detection and classification works. As explained before, after getting the prediction for the class from the output layer for all data instances on the batch, the loss function is called to evaluate the error of the predictions. Then the optimization policy adjusts the trainable parameters so that the next predictions of the next batch of data will hopefully be more accurate. When we reach the final epoch, having made predictions for the whole dataset multiple times, we can end the training period and start using the network for new and “unseen” data.

1.2. Related Work

In this chapter, we are going to overview some of the methods previously used for processing Electrophysiological Signals, including an ML approach for in-vitro LFP signals, as well as a DL approach for in-vivo EEG signals, the latter being the inspiration for our approach in detecting and classifying events of in-vitro LFP signals, and showcase the advantages and limitations of each method.

1.2.1. Analysing Electrophysiological signals

In recent years, there have been various methods and approaches applied to Electrophysiological signal processing.[16] Software packages like EEGLAB[40], CHRONUX[41], OSort[42], Spike2[43], have been proven to be really effective for other types of neuronal signals, but not as much when they are used for detection and quantification of events on LFP signals. This can be easily credited to the very nature of field potential recordings that are characterized by inconsistent and fluctuating voltage bursts.

Some researchers attempted to combat these issues by combining intracellular with extracellular recordings, thus extracting event-related information from intracellular recordings and applying them on LFP recordings[44,45]. Others based their approaches on multiunit activity[46,47], that is established as having higher acquisition frequencies, while they use neuronal spikes for event detection, but these are properties not found on LFP recordings. Other solutions include the utilization of user-defined inconsistent global thresholds for the voltage, so when you get values surpassing said threshold, the software marks it as an event.[48,49] The main problem with these kind of approaches is that they are not data driven, i.e. the applied threshold is global and selected once before the processing starts, not taking into account the differences that may occur between diverse recordings across labs, animals, drugs or tasks that the animals may perform during the recordings, leading to either, or both, overestimation and underestimation of the number of events in a recording or of the onset and offset values of these events. Aside from that, they rely on trial and error to select that threshold, making them extremely time consuming, and the fact that even on the same experimental models, a different threshold may yield better results, proves that the threshold is subjective.[16]

Another group of researchers managed to overcome these problems, by supplying a data driven method for automatically defining the threshold on pre-processed in-vivo LFP recordings.[50] In this approach, the threshold is selected after a series of assumptions, who may be logically set, but remain user-defined. The downsides of this approach are that it includes an essential pre-processing step to eliminate frequencies under 20Hz and also, that the threshold is applied to the whole recording, both of which can be proven to be crucial, because the lower frequencies can be important to the research at hand or because the signal may contain events with low periodically or dynamic baseline fluctuations.[16]

To summarize, previous work on the subject of LFP signal processing, either relies on detection on other-than-LFP electrophysiological signals or relies on global thresholds that may not always perform well on dissimilar datasets.

1.2.2. LFPAnalyzer

The aforementioned reasons led to the development of LFPAnalyzer, a software package that allows automatic detection and quantification of events in LFP signals, without the need for assumptions on the signal on the user's end, by applying data driven methods that are locally adaptive, leading to a more precise estimation of the threshold and in turn, more precise estimation of the onsets and offsets of the events. Additionally, there is no need for a pre-processing step, as the method is applied to unfiltered recordings, the approach is fully automated and successful in detecting recurrent events, as well as estimating their various properties.[16]

The techniques this method employs come from Signal Processing, Speech Analysis and Machine Learning. An overview and an example can be seen in Figure 1.9, while the steps will be explained in detail below.

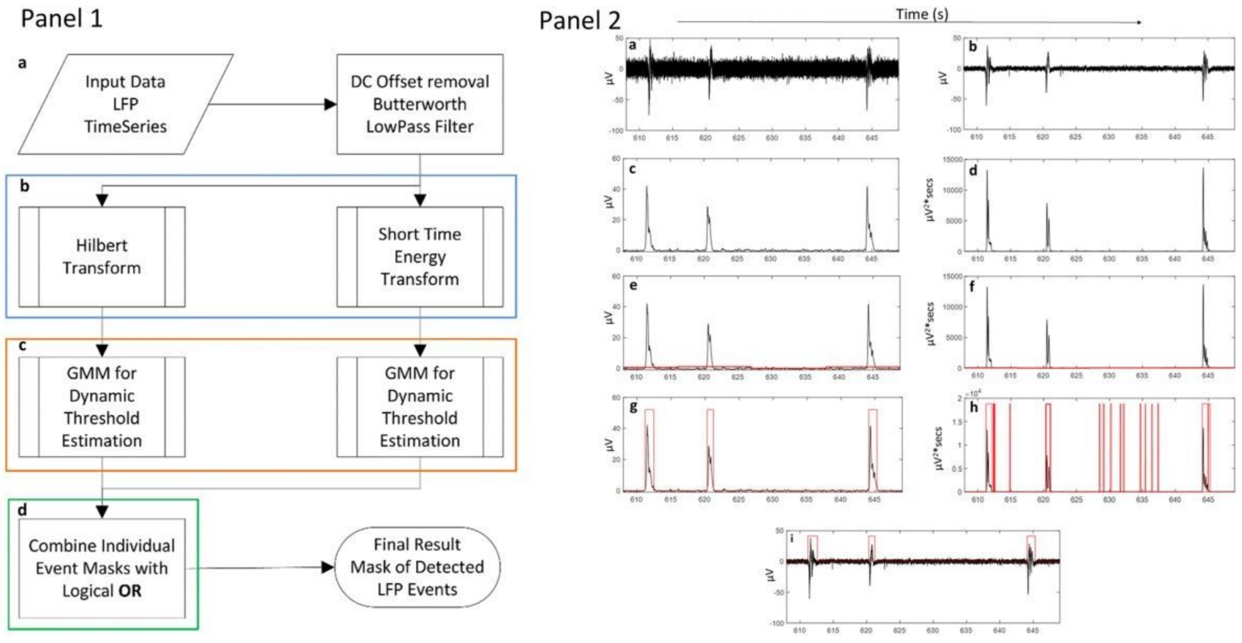

Panel 1
a
Input Data LFP TimeSeries
DC Offset removal Butterworth LowPass Filter
b
Hilbert Transform
Short Time Energy Transform
c
GMM for Dynamic Threshold Estimation
GMM for Dynamic Threshold Estimation
d
Combine Individual Event Masks with Logical OR
Final Result Mask of Detected LFP Events
Panel 2
Time (s)
μV
μV²*secs

Figure 1.10: Schematic overview of LFPAnalyzer: Panel 1 shows the steps that are applied to detect the events and Panel 2 shows the same steps applied on a frame of an LFP recording.[16]

Panel 1-a: First of all, the DC offset created by the acquisition process and the amplifier gets subtracted from the LFP recording, transforming the signal to analog. Then, because higher frequencies than 200Hz do not carry any relevant information, a third order Butterworth low-pass filter is applied to remove them. The preprocessed signal gets segmented to frames of 11 seconds, in order for the method to be locally adaptive and avoid applying a global threshold. These frames are non-overlapping, and the selection of their length is based on the fact that for the threshold estimation to be successful, both an event (or more), that usually last no more than 8 seconds, and some baseline information (noise) need to be present. The results of the first step applied on Panel 2-a can be shown in Panel2-b.[16]

Panel 1-b: The next step of this method is to apply two complementary transformations to all extracted frames, to generate two feature signals. The first one is the Hilbert Transform, a fundamental tool in Fourier analysis[51], that has been used in neurophysiological signals for latency analysis.[52,53] Hilbert Transform is very suitable for LFP signals, as it can be regarded as a method that represents narrow-band signals with reference to amplitude and frequency modulation and will provide the signal’s envelope. The second is the Short TIme Energy Transformation, which is widely used in speech analysis and is very effective for silence period detection, as well as distinguishing audio classes.[54] Given that speech and LFP signals share a lot of similarities, such as that they are both generated from a time varying vocal tract and electrical burst respectively, with time varying excitation, and thus both of them are non-stationary. These two transformations will be used to evaluate two distinct thresholds that will be combined to provide the final threshold that will be used for detecting the events. The need for both of them stems from the fact that Hilbert Transform excludes parts of the events, as low amplitude periods are not passed to the transformed signal, while Short Time Energy Transform tends to divide individual events, due to its hypersensitivity to small silent periods detected in the middle of individual events. The two transformed signals can be shown in Panel 2-c and Panel 2-d.[16]

Panel 1-c: Then, the Gaussian Mixture Model (GMM), a data driven method from ML is applied to dynamically define the thresholds for both feature signals.[55] The method is adjusted to be used unsupervised[56] and will dictate the number of classes of events in the data. The algorithm produces one histogram of the voltage values for each frame of the transformed signal. We define as the two available classes that could be detected in a frame as one being an event, when it includes high voltage values, positive or negative, and the other being the noise, when it doesn’t include high values. Each frame may include either both noise and events, or just noise. Then, we can assume that the histogram will contain two peaks, should there be noise and events, and just one peak, should there be no events. The histogram is produced by the combination of GMM and Expectation-Maximization algorithm[56,57], while the threshold is estimated through the incorporation of Minimum Message Length criterion into the Expectation-Maximization algorithm, providing flexibility for precise evaluation of the threshold on LFP signals with diverse properties.[58] If the frame contains only noise, then no threshold is selected. Otherwise, the threshold is selected under the Bayes rule.[58] On Panel 2-e and Panel 2-f, the red line representing the threshold can be shown for the two transformed signals. Anything surpassing this line is considered an event, while anything under it is considered noise, i.e. baseline.[16]

Panel 1-d: The last steps of the algorithm is to extract the corresponding masks for each transformed signal and combine them through the use of a Logical OR operation. The two thresholds are applied on the two transformed signals, and the generated masks consist of periods of ones, depicting the events, and periods of zeros, depicting the noise. A post-processing step is applied on the combined mask, so that events whose standard deviation (SD) is lower than the complete signal’s SD can be excluded, given that events will show higher SD than the complete signal, because it includes long periods of noise with small amplitude variability. The extracted masks can be shown in red in Panel 2-g and Panel 2-h, while the combined and final mask on Panel 2-i.[16]

This software has been proven to be more efficient than manual analysis and superior to other methods, as it overcomes the arbitrariness of global thresholds and succeeds in detecting all events that show clear differences from noise, even when the events’ periods are relatively so short that they might have remained undetected from other methods.[16]

1.2.3. Dreem One Shot Event Detector

Dreem One Shot Event Detector (DOSED) is a DL algorithm for detecting and classifying events in EEG recordings, specifically ones recorded during sleep.[8] This approach is influenced by computer vision algorithms used for object detection, e.g. YOLO[59] and SSD[60]. As such, DOSED uses a CNN to construct a feature map from raw EEG recordings, with two parallel modules on the CNN's output layer, one used for localization, i.e. detecting the events' centers and durations, to extract their onsets and offsets and one used for classification, distinguishing between 3 classes: spindles, K-complexes and arousals. The CNN is trained with supervised learning and back-propagation.[8]

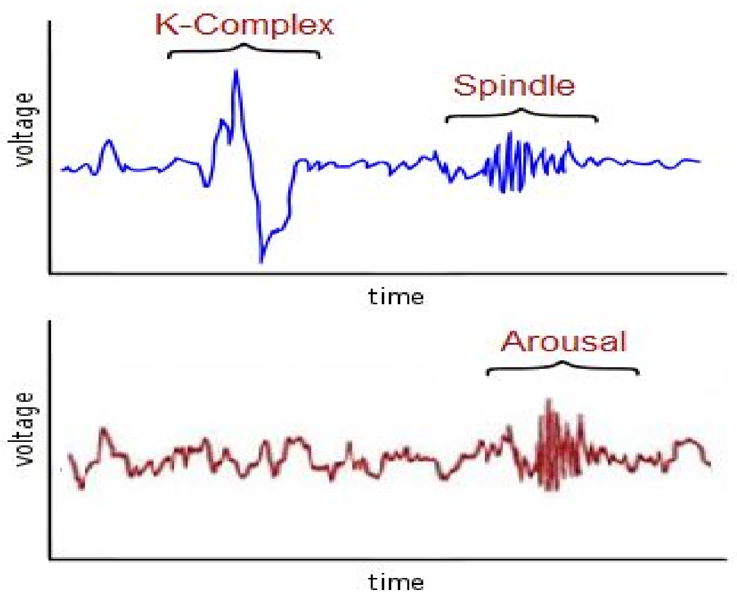


Figure 1.11: Examples of EEG micro-events: a K-Complex, a Spindle and an Arousal.

The first step of the algorithm is to divide every EEG recording of the provided labeled dataset, into EEG samples of 20 seconds duration. Then, N_d default overlapping events are generated, covering the entire sample. Each default event is characterised by its center t^c and duration t^d. The initial values for the default events' durations depend on the typical duration of a true event that the CNN is trying to detect, as well as the overlapping factor, in order for the whole sample to be covered.[8]

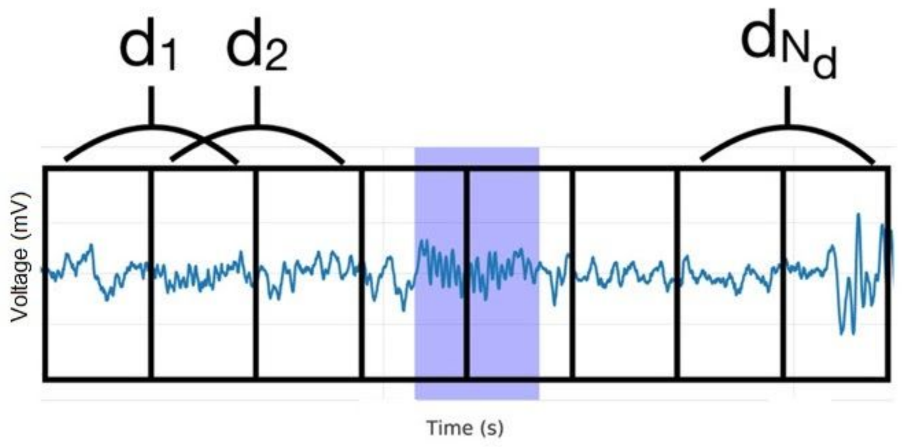


Figure 1.12: N_d default overlapping events are generated, covering the whole sample. Inside the purple box lies an event (yet undetected).[8]

Then, the CNN is feeded with all the generated samples, and it generates the predicted events, i.e. the adjusted centers and durations for each of the N_d default events in each sample, as well as the probability p_L of each of them containing a true event. If this probability, of any predicted event, is higher than a certain cross-validated threshold θ_L, then it is considered as containing a true event.

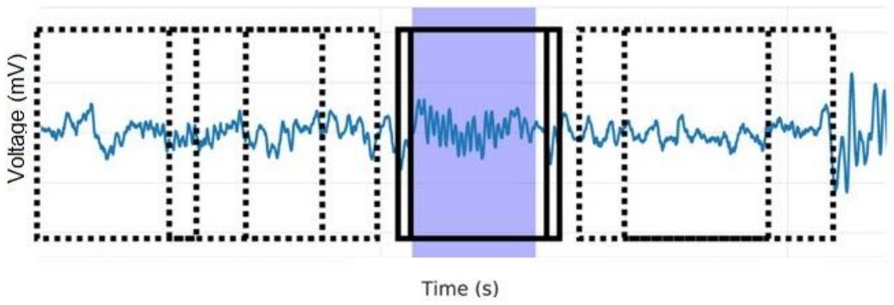


Figure 1.13: The CNN generates the predicted events, whose centers and durations are the adjusted coordinates of the default events from Figure 1.11.
The dotted boxes contain predicted events, with $p_L < \theta_L$, while the other two overlapping boxes in the middle, over the purple box, contain predicted events with $p_L \geq \theta_L$. [8]

Lastly, non-maximum suppression (NMS)[58,59] is used to exclude overlapping events, by grouping predictions with the same detected class, based on their IoU score[59], keeping the ones with the highest p_L.[8]

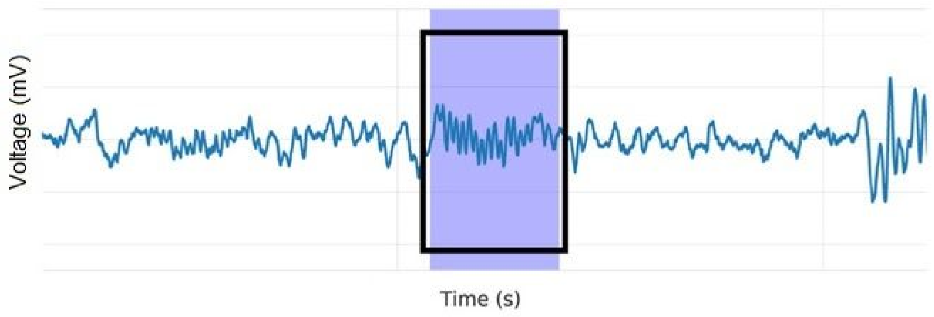


Figure 1.14: NMS applied on the predicted events with $p_L \geq \theta_L$ from Figure 1.12 generates the final output of the algorithm that contains the true event. [8]

The first part of the CNN consists of a Convolutional 2D layer, with Linear activation function, Stride=1 and C number of kernels of size=(C,1), where C is the number of EEG channels of the recording, followed by a transpose layer. These two layers act as spatial filters of the input multivariate signal with the purpose of increasing the signal to noise ratio[61,62,63] Should there be only one channel on the EEG recording, these two layers remain unused.[8]

The second part, consists of a series of K Convolutional 2D layers with ReLU activation function, Stride=1 and $4x2^k$ number of kernels of size=(1,3), where k=1..K, each one followed by a Max Pooling 2D layer with Stride=(1,2) and kernel size=(1,2). These layers act as temporal feature extractors, processing the codependence of the input sample in terms of time. The number K was selected to be equal to 8.[8]

The last part of the CNN, consists of the localization and classification modules, the output of the CNN. Localization is performed through the use of a Convolutional 2D layer, with Linear activation function, no stride and $2xN_d$ number of kernels of size=$(C,T/2^K)$, while classification through the use of a Convolutional 2D layer, with softmax activation function applied every L+1 kernels, with no stride and $(L+1)xN_d$ kernels of size=$(C,T/2^K)$. The number T represents the time series of the sample, e.g. if the sample is 20 seconds long and the recording is sampled at 256 Hz then T = 5120 and L is the number of classes to be detected in the data. The reason that the kernels are equal to $(L+1)xN_d$ is because the algorithm assigns the label ‘0’, to predicted events containing no true event, then the label ‘1’ for events that belong to class 1 and label ‘L’ for the events of class L. These two convolutional layers are parallel to each other and connected to the last Max Pooling 2D layer of the precious block of the CNN.[8]

For the training of the CNN, a custom loss function is attempted to be minimized, that utilizes the IoU score between the default events and the true events to match them, in order to compare the predicted events with the corresponding true events. Stochastic gradient descent was used as an optimizing policy, with learning rate = 10^{-4} or 10^{-3} (depending on the dataset), momentum = 0.9, batch size = 32 and epochs = 200. The batches fed to the network contained 50% samples with at least one true event and 50% with no true event. Three metrics were used to evaluate its performance: precision, recall and f1 score[8]

Before using the network for training, the programmer needs only to select the default events number N_d, their duration and overlapping factor, based on the properties of the events that need to be detected, and the learning rate and detection threshold θ_L, that should be evaluated by experimental knowledge.[8]

DOSED has been proven to be extremely versatile and efficient in both detecting and classifying micro-events in EEG recordings, surpassing the three other state-of-the-art detection approaches it was compared with, as it has been tested on four datasets.

1.3. Our Objective

While the LFPAnalyzer excels in detecting events, their classification still remains a manual and extremely time consuming task, given that a researcher may have to process hours long recordings. On top of that, on the programmer’s side, a vast background on electrophysiological signals and their properties is mandatory in order to develop such algorithms, and generalizing LFPAnalyzer to detect events in signals other than LFP can be tremendously complex.

As previously mentioned, DL algorithms offer automatic feature extraction, so the programmers need to know the bare minimum of the properties of the signal they are going to analyze. Moreover, DL algorithms generalize really well on similar datasets, making it possible for the programmer to use the same DNN architecture, with minimal adjustments, to learn to perform the same task on different input data.

So, in order to develop an algorithm for event detection and classification of in-vitro LFP electrophysiological signals with Deep Learning, we can simply adjust the DOSED algorithm to receive as input data samples from LFP recordings, instead of EEG and train it accordingly.

In this thesis, we will explain every part of DOSED in full detail. Then we will analyse the necessary changes that need to be made for it to train with LFPs, as well as the datasets that we are going to use, how to extract the samples and what statistics we can infer. Finally, we will show some of the libraries, hardware and software that can be used to develop, train and use a DL algorithm.

2. SYSTEM OVERVIEW

In this section, we will give an in depth look of the system we developed, that is inspired by DOSED. We will explain in full detail how a CNN, a model that is used for computer vision problems, can be modified to analyze electrophysiological signals, as well as how we can modify DOSED to analyze LFP recordings, instead of EEG. The code was fully developed in Python 3, as well as Keras and Tensorflow.

2.1. Electrophysiological Signal Processing as a Computer Vision task.

Before the development of software that automatically processes electrophysiological signals, the labor of event detection and classification had to be performed manually by an expert. The expert would have to create a visualization of the signal, manually or automatically, creating a function of its voltage fluctuations over time and then visually inspect it, to manually annotate the events in the signal. Since the human brain is able to perform such a task, with the help of human vision, and Computer Vision algorithms have been proven to be better detectors, we can safely assume that a computer can also perform visual signal analysis.

So, we can build a CNN that instead of 2-dimensional images, it receives 1-dimensional “images” of an electrophysiological signal, since the time variant has a constant rate and only the voltage fluctuates at a changing rate. Thus, we can represent the input signal as a vector, where the i-th element gives the voltage at the i-th time step. The reason we avoid including time as a 2nd dimension is that the more dimensions the algorithm will need to process, the greater the algorithm’s complexity will be. We will further explain all the preprocessing steps to transform our dataset to the desired form later.

The CNN will be trained with supervised learning. As such, we will need to develop a loss function that will estimate the error of the network’s predictions, one that the network will attempt to minimize, to learn to perform the task at hand. Moreover, the loss function needs to be differentiable with respect to the trainable parameters, so that stochastic gradient descent methods and back-propagation can be applied as the optimization policy. To achieve this, we are going to develop a mathematical model that solves the problem of event detection and classification and transform it into a CNN.

2.2. Mathematical Model

The aim of the algorithm is to learn a prediction function $\widehat{f} : X \rightarrow Y$.

2.2.1. Event, Input and Output Definitions

We start with some definitions:

- Let $X = \Re^{CxT}$ be the set of input LFP signals in our dataset, where C stands for the number of channels of the recorded signal and T stands for the number of time steps.
- Let $L \in \aleph$ be the number of available labels or classes of events to be detected.

- We define as $\mathcal{L} = \{1, 2, ..., L\}$ the set of labels, and 0 the label that represents the absence of an event, i.e. baseline.
- Let $\mathcal{E} = \Re^2 x\ \mathcal{L} \cup \{0\}$ be the set of events, with an event $e = \{t^c, t^d, l\} \in \mathcal{E}$ be characterized by its center in time $t^c \in \Re$, its duration $t^d \in \Re$ and its label $l \in \mathcal{L} \cup \{0\}$.
- We define as a *true event*, an event with label $l \neq 0$, that has been annotated by a human expert and as a *predicted event*, an event with label $l \neq 0$, that has been annotated by an algorithm.
- Let $x \in X$ be an LFP sample of 20 s duration, extracted from our dataset.
- Given that $N_d \in \aleph$, we generate N_d *default events* over sample x.
- Let $D(x) = \{d_i, i \in \{1, 2, ..., N_d\}\}$ be the set of *default events*, generated over sample x, with $d_i = (t_i^c, t_i^d)$ being the i-th *default event*. t_i^c is the default event's center, while t_i^d is its duration.
- Let $E(x) = \{e_j, j \in \{1, 2, ..., N_e\}\}$ be the list of the N_e *true events* annotated over sample x.
- Let $Y \subseteq \mathcal{E}$ and $\widehat{f}(x) \in Y = \{(\widehat{t}_i^c, \widehat{t}_i^d, \widehat{l}_i) \in \mathcal{E}, i \in \{1, 2, ..., N_d\}\}$ be the prediction made by the model $\widehat{f}$ over the sample x, where $\widehat{r_i} = (\widehat{t}_i^c, \widehat{t}_i^d)$ are the predicted adjustions to the coordinates of default event d_i.
- Actually, the network, instead of $\widehat{l}_i$, will output the probabilities of each label for every event, so let $\widehat{\pi}_i \in [0, 1]^{|L|+1}$, be a probability vector, where $\sum_{l \in L \cup \{0\}} \widehat{\pi}_i^l = 1$.
- Finally, $\widehat{f}(x) \in Y = \{(\widehat{t}_i^c, \widehat{t}_i^d, \widehat{\pi}_i) \in \mathcal{E}, i \in \{1, 2, ..., N_d\}\}$ is the actual prediction made by the model $\widehat{f}$ over the sample x.

So the network takes as input a sample x and provides as output, not the actual coordinates of the predicted event, nor the predicted label, but the predicted adjustions to the coordinates, i.e. centers and durations, of the default events that were generated over sample x, as well as the predicted probabilities for each event of belonging to every available class.

The network's goal is for these predicted adjustions to result in a default event's coordinates perfectly matching a true event's coordinates in time, as well as the label of that particular default event that has the highest probability to be the label annotated for the matching true event.

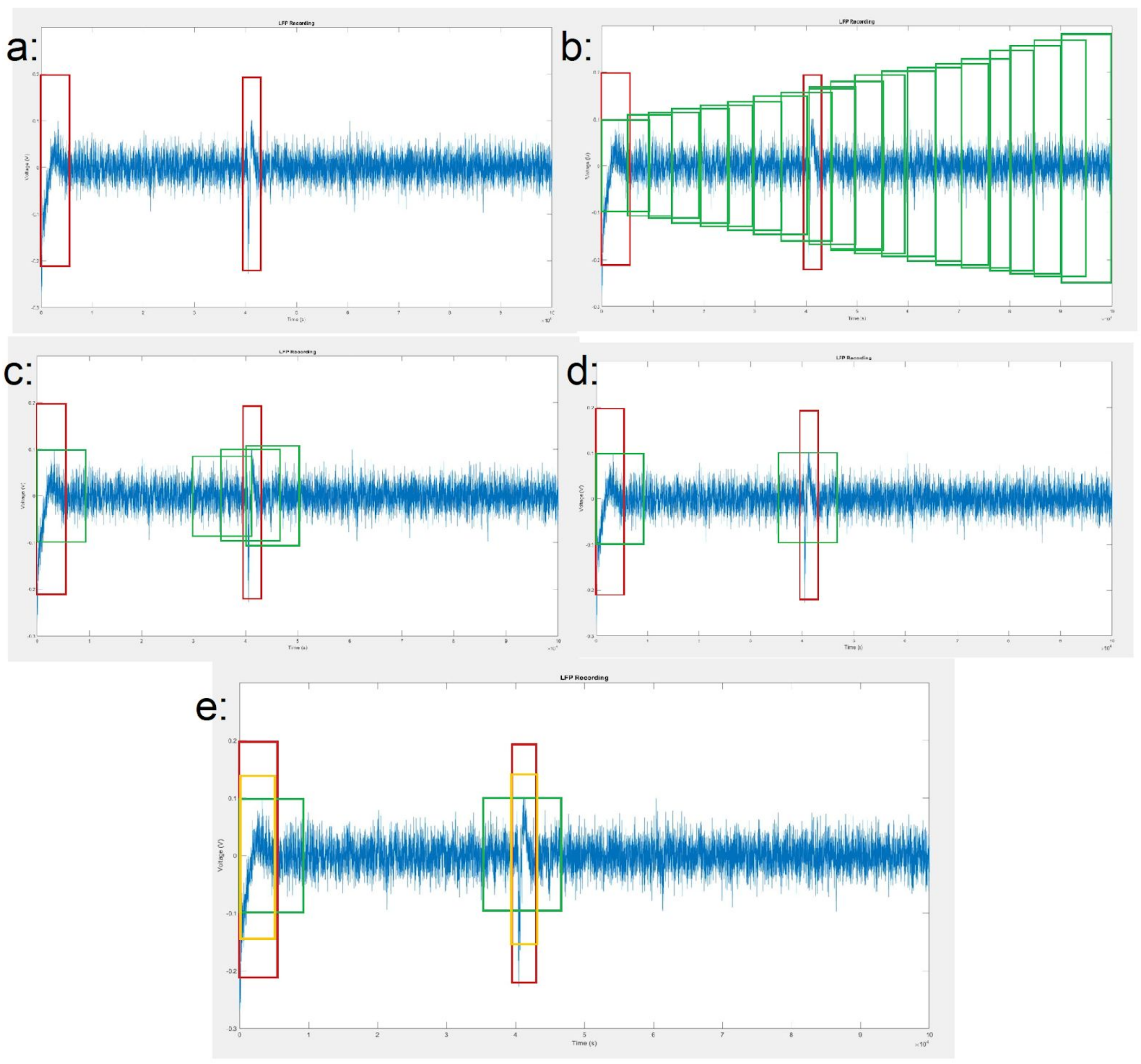


Figure 2.1: Default events generated over a sample x and the trained network's expected output:

a) The sample x, with two annotated true events in red: e_1=(0.25,0.5,1),e_2=(4.1,0.4,2).

b) N_d default events generated over sample x, with duration=1s and overlap factor=0.5 in green. Note that the difference in the size of the boxes at the y-axis is used only for the overlapping of the default events to be more visually comprehensible.

c) Only four of the N_d default events intersect with the true events: d_1,d_7,d_8 and d_9.

d) The two default events that exhibit the highest matching factor with the true events: d_1=(t_1^c=0.5,t_1^d=1) and d_8=(t_8^c=4,t_8^d=1)

e) The network's expected output for the N_d correctly predicted events: (Boxes in yellow are the ones that have been predicted to contain a true event):

if $i \neq 1, 8$ **:** $\widehat{f}(x) = (\widehat{t}_i^c, \widehat{t}_i^d, \widehat{\pi}_i)$ **, where** $argmax(\widehat{\pi}_i) = 0$

if $i = 1$ **:** $\widehat{f}(x) = (\widehat{t}_1^c, \widehat{t}_1^d, \widehat{\pi}_1)$ **, where** $\widehat{t}_1^c + t_1^c = 0.25, \widehat{t}_1^d + t_1^d = 0.5$ **and** $argmax(\widehat{\pi}_1) = 1$

if $i = 8$ **:** $\widehat{f}(x) = (\widehat{t}_8^c, \widehat{t}_8^d, \widehat{\pi}_8)$ **, where** $\widehat{t}_8^c + t_8^c = 4.1, \widehat{t}_8^d + t_8^d = 0.4$ **and** $argmax(\widehat{\pi}_8) = 2$

2.2.2. Loss function

With these in mind, we will define a loss function that quantifies the error of the prediction. We aim for the loss function to be minimized by the optimization policy. As such, we need the loss function to be equal to zero when we have the correct prediction, and higher than zero when it's incorrect. The difficulty in defining the loss function stems from the fact that we need to quantify the error in a way, that the furthest we are from the correct prediction the higher the value of the loss function should be.

To do so, we will use the Intersection over Union (IoU) criterion[59] that quantifies the overlap between two areas:

$$IoU(area_a, area_b) = \frac{area_a \cap area_b}{area_a \cup area_b} \in [0,1],$$

where $area_a, area_b$ are two areas with the same number of dimensions.

The IoU will be used to estimate the time overlap between a default and a true event. So if $IoU(d_i, e_j) = 0$, then the two events don't overlap in time, and if $IoU(d_i, e_j) = 1$, then the two events perfectly overlap.

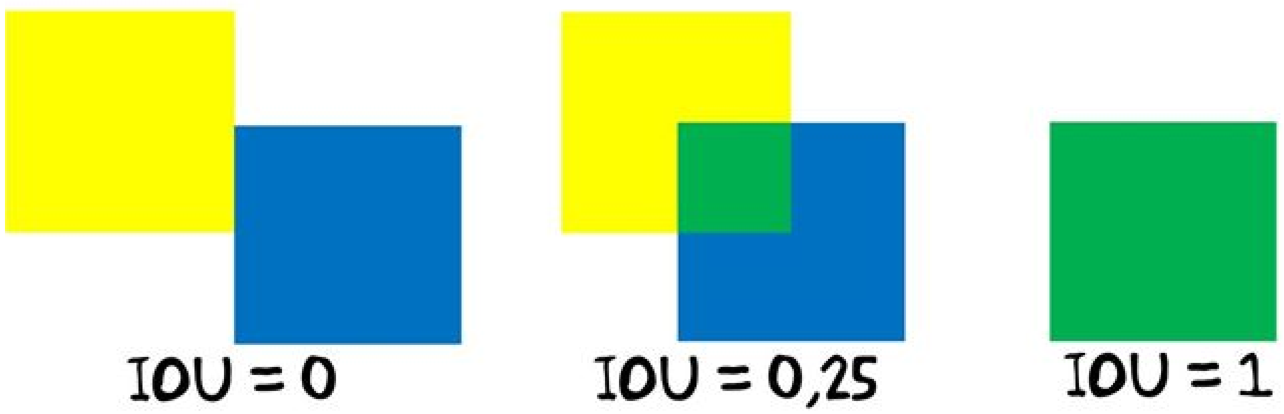


Figure 2.2: Three examples of the IoU between two squares in 2 dimensions.[67]

We are going to apply per-prediction matching[60], thus we will apply bipartite matching, to match every true event with the default event that exhibits the highest IoU value. Then compare the coordinates of the predicted events that correspond to the default events, with the true events the latter were matched with, as well as their predicted label, the one with the highest probability.

In order to define the loss function as such, we will define a series of functions that will be used to estimate the final error of the prediction:

- The matching function $\gamma(i) : \{1, 2, ..., N_d\} \rightarrow \{1, 2, ..., N_e\} \cup \{\varnothing\}$ returns, if it exists, the index j of the true event that exhibits the highest IoU with the default event with index i ,that is over a set IoU threshold $\eta_{Loss} \in [0,1]$:
    - if $IoU(d_i, e_j) - \eta_{Loss} \geq 0$, for at least one $j \in \{1, 2, ..., N_e\}$, then :
      $\gamma(i) = argmax_{j \in \{1,2,...,N_e\}} (IoU(d_i, e_j) - \eta_{Loss}) \in \{1, 2, ..., N_e\},$
    - if $IoU(d_i, e_j) - \eta_{Loss} < 0$, then:
      $\gamma(i) = \varnothing$

- The encoding function $\phi_{e_j}(d_i)$: $\Re^2 \to \Re^2$ returns the encoded coordinates of a default event d_i, in respect to a true event e_j, in order to quantify the relative variations in centers and durations between d_i and e_j[64] :
  - $\phi_{e_j}(d_i = (t_i^c, t_i^d)) = (\frac{t_j^c - t_i^c}{t_i^d}, log\frac{t_j^d}{t_i^c})$
- The loss function $L1_{smooth} : \Re^2 \to \Re$ applies coordinate-wise the Huber loss function[64] and is used to greatly penalize the network when the error is small, so it will keep on looking for a better solution, but also to not discourage the network when the error is big:
  - if $|x| < 1,\ then : L1_{smooth}(x) = \frac{x^2}{2}$
  - if $|x| \geq 1,\ then : L1_{smooth}(x) = |x| - \frac{1}{2}$
- The event-matched loss function $l^+ \to \Re$, returns the sum of the quantified accuracy of both the localization and classification for every predicted event, whose corresponding default events matched to a true event, through the matching function:
  - $l^+ = \sum_{i \in \{1,2,...,N_d\},\ \gamma(i) \neq \varnothing} L1_{smooth}(\phi_{e_{\gamma(i)}}(d_i) - \widehat{r}_i) - log(\widehat{\pi}_i^{l_{\gamma(i)}})$
- The event-unmatched loss function $l^- \to \Re$, returns the sum of the quantified accuracy of classification for every predicted event, whose corresponding default event did not get matched to a true event, through the matching function:
  - $l^- = -\sum_{i \in \{1,2,...,N_d\},\ \gamma(i) = \varnothing} log(\widehat{\pi}_i^0)$
- The final loss function $l : Y \times Y \to \Re_+$, between the true annotation $E(x)$ and the model prediction $\widehat{f}(x)$ over the signal x is defined as:
  - $l(E(x), \widehat{f}(x)) = l_{norm}^+ + l_{norm}^-$

    where l_{norm}^+, l_{norm}^- are obtained by dividing l^+, l^- with the number of their terms respectively.

In conclusion, training the network is reduced to solving the minimization problem below:

- $$\widehat{f} \in argmin_{f \in \mathcal{F}} \mathbb{E}_{x \in X}\left[l(\ E(\ x)\ , f\ (\ x)\ )\ \right]$$

2.2.3. Function f - The architecture of the network

Now that we defined the loss function along with the training of the network in mathematical terms, we can presume to define the function f.

We are going to thoroughly explain the DOSED, and by extension our own, CNN architecture, through its mathematical model. We can think of the network as being a set of three distinct parts. The spatial filtering part, the temporal processing part and the predictor/output part.

We will use for f a CNN, that outputs N_d potential events, given a set of default events $D(x) = \{d_i,\ i \in \{1, 2, ..., N_d\}\}$, over a signal x.

- Let $K, F, C, T \in ℵ$. Then we can define the network as the composition of three function:

$$f(x) = \psi(\varphi_T(\varphi_C(x)))$$

2.2.3.1. The function φ_C - Spatial Filtering:

The multivariate signals of the input are spatially filtered, in order to increase the signal to noise ratio, as it corresponds to a matrix multiplication.[61-63] This can be used to increase the algorithm's robustness in cases where there have been bad channels or electrode removal.[8]

The function $\varphi_C : X \to X$ performs C linear combinations of the C input signals and returns a new signal $\bar{x} \in X$ in the form of a tensor. It is implemented as a two dimensional convolutional layer, with C kernels of size= $(C, 1)$ that correspond to space and time. The output of the convolutional layer acts as the input to a two dimensional convolutional transpose layer that, as its name implicates, performs a transpose operation. In the case where the input signal x has only one channel ($C = 1$), this function translates to the identity function and therefore $\varphi_C = Id$.

2.2.3.2. The function φ_T - Temporal Feature Extraction:

The purpose of this function is to extract temporal dependencies in the signal x. Starting from the first layer, it relates the neighbouring voltage values to each other, creating small groups. From then on, going deeper into the network, the function relates the small groups to bigger and bigger, until it passes through every layer. The output of this function is a tensor whose values correlate the whole original signal in the temporal dimension.

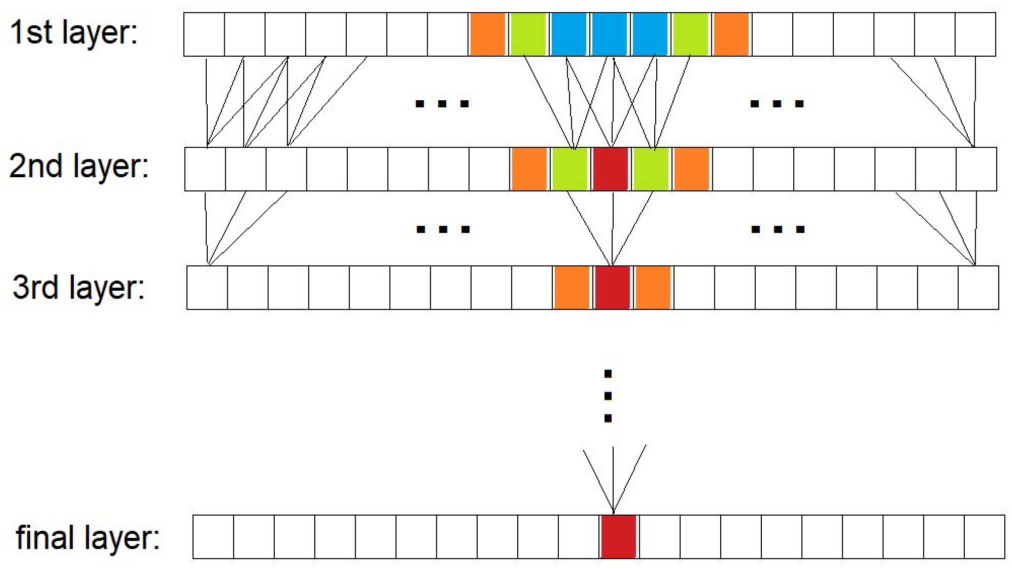


Figure 2.3: A visualization of a simplified Temporal Feature Extraction with an input of 20 elements, where each square represents the voltage value of the i-th time step, and the kernel has size=3:

<u>1st to 2nd layer:</u> The voltage information from the three blue squares of the 1st layer gets correlated in the red square of the 2nd layer.
<u>2nd to 3rd layer</u>: The voltage information from the two green squares of the 1st layer gets correlated with the information of the three blue, resulting in the red square of the 3rd layer.
<u>final layer:</u> All coloured squares from the first layer, along with some other neighboring squares, have been correlated in the red square of the final layer.

The function $\varphi_T : X \to \Re^{F \times C \times \overline{T}}$ consists of K blocks, where each block k consists of a two dimensional convolutional layer, with batch normalization[65], and ReLU activation function[66] and a max-pooling layer. The convolutional layer of block k will convolve the previous feature maps x_{k-1} with 4×2^k kernels of size=(1,3) that correspond to space and time, with stride=1 and the use of zero padding to maintain the dimension of the input tensor of the layer. The temporal max-pooling has kernel size=(1,2), with stride=2 and divides the temporal dimension by 2. None of these blocks process the spatial dimension. The output of this function is a tensor with shape = $(F, C, \overline{T})$, where $F = 4 \times 2^K$ and $\overline{T} = \frac{T}{2^K}$.

2.2.3.3. The function ψ - The predictor:

The aim of this function is to perform the final prediction, using the temporal feature maps extracted by φ_T. It will predict both the adjustions to the coordinates and the probabilities of the labels for all N_d potential events, based on the N_d default events generated over sample x.

The function $\psi : \Re^{F \times C \times \overline{T}} \to (\Re^2 \times \mathcal{L} \cup \{0\})^{N_d}$ consists of two parallel to each other two dimensional convolutional layers. The first one, used for localization, has $2 \times N_d$ kernels of size=$(C, \overline{T})$, with linear activation function, while the second, used for classification, has $(L+1) \times N_d$ kernels of size=$(C, \overline{T})$, with softmax activation function applied on every $L+1$ output feature maps, in order to end up with $L+1$ probabilities (one for every label plus one for noise).

2.3. Our Approach

We are going to use the exact same network architecture as DOSED, since we can infer that it will be sufficient for LFP as well. For example, a CNN that was built for learning to detect dogs, can be used to detect cats, or other animals, or maybe objects. As long as the network receives the same type of input data, to learn to perform the same task, the only thing that differs between learning to detect different things is the datasets that will be used to train it.

Now that we have set the mathematical foundations we can build the network itself, that will receive samples from preprocessed recordings, in the form of tensors and will output tensors representing the coordinates and probabilities of labels for the N_d potential events. The final step that is applied to the output of the network is Non-Maximum Suppression, and its purpose is to exclude overlapping potential events with labels other than 0, by selecting the one with the highest confidence.

Table 2.1: The CNN architecture, where C is the number of channels of the input sample/tensor x, T is the number of time steps and K is a hyperparameter set before the start of the training period, that defines how many blocks will perform temporal feature extraction.

Module	Layer Type	Number of Kernels	Size of Kernels	Output Dimension	Activation Function	Stride	Padding
Input	Input	-	-	$(1, C, T)$	-	-	-
φ_c	Conv2D	C	$(C, 1)$	$(C, 1, T)$	linear	1	same
	Conv2D Transpose	-	-	$(1, C, T)$	-	-	-
φ_T k *blocks* $k \in \{1, 2, ..., K\}$	Conv2D	4×2^k	$(1, 3)$	$(4 \times 2^k, C, T/2^{k-1})$	ReLU	1	same
	Batch Normalization	-	-	$(4 \times 2^k, C, T/2^{k-1})$	-	-	-
	MaxPooling 2D	-	$(1, 2)$	$(4 \times 2^k, C, T/2^k)$	-	(1,2)	-
ψ - localization	Conv2D	$2 \times N_d$	$(C, T/2^K)$	$(2 \times N_d, 1, 1)$	Linear	-	valid
ψ - classification	Conv2D	$(L+1) \times N_d$	$(C, T/2^K)$	$((L+1) \times N_d, 1, 1)$	Softmax every $(L+1)$ kernels	-	valid

2.3.1. Preprocessing

We are going to apply two preprocessing steps, one used to increase the signal to noise ratio and one used to visually magnify the signal.

In order to increase the signal to noise ratio, we will perform the preprocessing steps of LFPAnalyzer. As such, we will subtract the DC offset from the LFP recordings transforming the signal to analog and will apply a Butterworth low-pass filter to remove frequencies higher than 200Hz.[16] Although this step is optional, as DOSED has been proven to be sufficiently applied to unfiltered raw EEG recordings[8], we have observed an enormous speed-up to the training process.

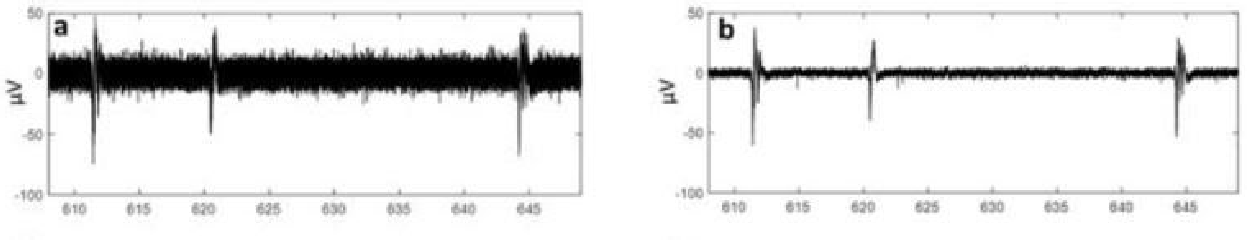


Figure 2.4: DC offset removal and application of a low-pass Butterworth filter:
a) A raw LFP recording.
b) The preprocessed LFP recording.

The second preprocessing step is to "visually" magnify the recordings. The intuition comes from the fact that the problem is treated as a Computer Vision task. Therefore we can assume that whatever action helps a human expert detect and classify events, will also help a computer perform the same task. Indeed, after experimentation, we observed that multiplying the LFP recordings by a factor of 100,000, greatly increased the convergence of the algorithm, as it visually magnifies the differences of the events, who exhibit voltage values away from zero, and the baseline, who exhibit voltage values close to zero.

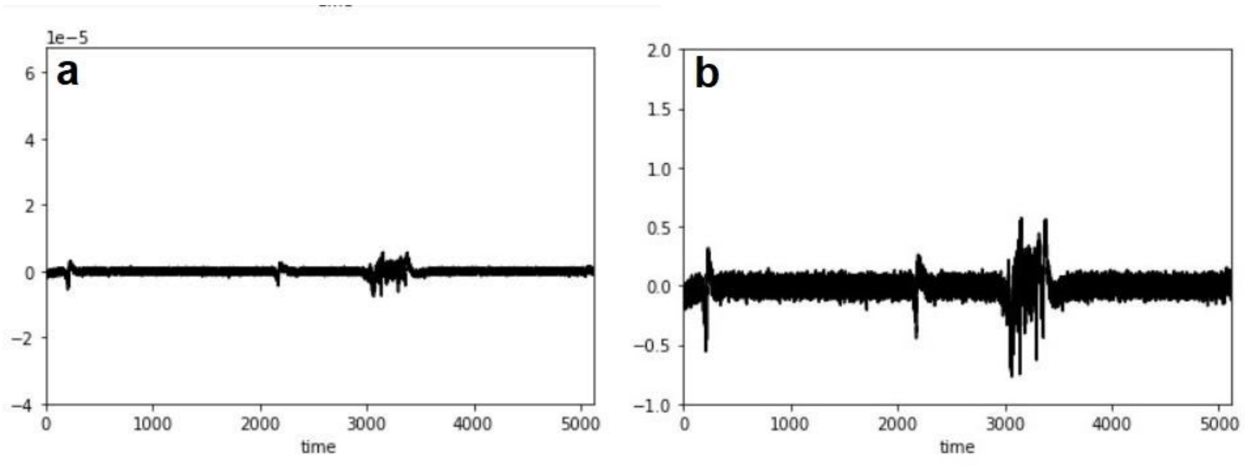


Figure 2.5: Magnification of a preprocessed LFP recording:
a) A preprocessed LFP recording.
b) The magnified LFP recording.

2.3.2. Input

The recordings are then divided into samples. The samples are extracted in two ways, depending on their use.

For training data, we want to keep a balanced dataset of samples. That means that we want 50% of the samples to contain at least one true event, while the other 50% to contain no true events at all.[8] The motive is that the network needs to learn to detect and classify both true events and baseline, so we need to make sure that none of them is favoured against the other. Since the samples that will be used for training are already annotated, we developed an algorithm that extracts one sample for every annotated event in the full recording. First, we make a list of every event in the recording. Then for every event in that list, we extract a sample whose coordinates in the original recording are selected randomly. As such, the position of the event inside the sample is random, and a lot of samples will end up containing more than one event, since the sample duration that we have chosen is large enough to allow for samples that are relatively close to each other to end up in the same sample. The events that randomly end up in a sample, are not excluded from the list, so there is a chance that the same neighboring events will be present in more than one sample. Additionally, every time we extract a sample with an event, by taking it off the event list, we randomly select a sample from the recording that contains only noise, whose coordinates are somewhere in between the previously selected event and the event that will be selected afterwards. If these two events are so close to each other, that a sample with sufficient duration can't be extracted, then we randomly select a sample with baseline from the full recording. The same action is also performed in the case where the last event of the recording is too close to its end.

When the network is already trained and we want to use it to make accurate predictions, we divide the recordings into sequential overlapping samples of the same duration as used in training. The overlapping factor of the samples is determined by the average duration of events used in training, to avoid on one hand over-segmentation that will lead to slower execution time and on the other hand to avoid missing parts of events that were divided when the samples were extracted.

Finally, in both cases, each sample is formatted in the input shape of the network, which is defined by C, the number of channels of the recordings, and by extent the samples, and T the number of timesteps. The latter is set based on the selected duration for the samples that we will feed to the network, as well as the sampling frequency that we chose to sample on the original LFP signals. So every sample is transformed into a tensor of shape (1,C,T) and feeded into the network in batches of the selected batch size.

2.3.3. Default Events

Before the start of the training process and the build of the network, we define the coordinates of the N_d default events that are generated over each sample x. These coordinates are the same for every sample and consist of the default events' centers and duration. The duration of the default events is selected based on the average duration of the already annotated LFP events of our dataset. The centers of the default events are initialized based, apart from their duration, on the their number: N_d and their overlapping factor. We will further talk about our selection for these numbers in the third section of this thesis.

2.3.4. Output

The output of the network for each sample x is two tensors of shape $(2 \times N_d, 1, 1)$ and $((L+1) \times N_d, 1, 1)$, which will be reshaped into two arrays of shape $(2 \times N_d)$ and $((L+1) \times N_d)$, representing the N_d predicted potential events' adjustions to the default events' coordinates and probabilities of labels respectively. The predicted adjustions to the default events' coordinates will be added to the actual coordinates of the N_d default events to provide the actual coordinates of the predicted potential events. Then the argmax operation will be performed on the predicted probabilities to provide the actual predicted labels of the potential events. These will be combined to construct a list of potential events on which we will apply the post-processing step.

2.3.5. Post-Processing

We will apply on the list of potential events Non-Maximum Suppression[59-60] (NMS), to exclude events with labels other than zero and suppress the potential events that overlap with each other, thus eliminating duplicate detections.

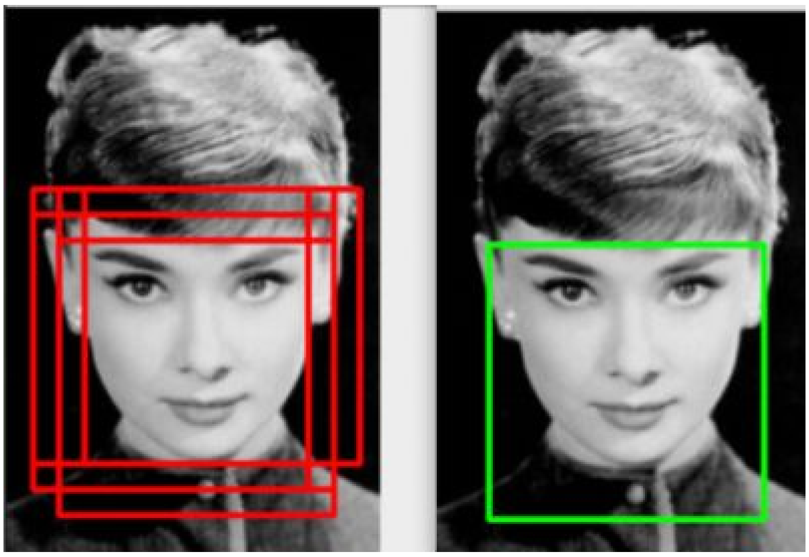


Figure 2.6: NMS applied on detection boxes used for face detection on an image.

Left: The detection boxes that consist the output of the CNN.

Right: The final selected detection box that contains the detected face, as the output of NMS.[68]

NMS is an algorithm that also uses the IoU criterion, in order to group overlapping detection boxes and proceeds to select the one with the highest confidence score on each group. Specifically, we will use the potential events' coordinates to group each two consecutive events, when they exhibit IoU higher than a set threshold η_{NMS}. Afterwards, we will use the potential events' probabilities for each label as the confidence score, and the algorithm will only keep the one with the highest score. If an event, that has label other than 0, doesn't exhibit IoU with any other potential event higher than the threshold, then it is also selected as an actual event.

Finally, we check if each of the events selected by NMS exhibit a confidence score, i.e. the probabilities of containing a true event, higher than a set threshold $\theta_{confidence}$, thus providing the final output of the system.

3. EXPERIMENTS

3.1. Dataset

The experiments were performed on a dataset provided by Skaliora’s Electrophysiology lab at the Biomedical Research Foundation of the Academy of Athens. The dataset contained 756 LFP recordings of 20-23 mins duration, with varying sampling frequencies: 5000-10000 Hz. The recordings were obtained from mice of both sexes, from different age groups, of two types: Wild Type (WT) and b-2 Knockout (b2KO), while various drugs were administered to more than half of them, such as GCP, Gabazine, DHbE, MLA, or a combination of the last two, on varying doses. Most of the recordings were originally obtained for three other experiments, one studying the effects of Gabazine and GCP, one studying the effects of DHbE and MLA and the last examining sex differences. All of the recordings were recorded on the same recording site, on room temperature, through a single channel, so they are not multivariate signals as the ones used in DOSED[8].

The events on the records were detected using the LFPAnalyzer software package. In some rare occasions, their onset and offset were manually edited by human experts, if they observed that they weren’t accurate enough. Additionally, they were fully labeled by the same experts.

From these 756 LFP recordings we extracted 62438 samples of 20 s duration, with sampling frequency of 256Hz. The reasons for the selection of the samples’ duration and sampling frequency are explained in full detail in the 3.3.1 T time step: sample duration and sampling frequency section below. The average samples per LFP recording were found to be 82.58. Of the 62438 samples, 31219 contain at least 1 event, while the rest 31219 samples contained no events, keeping the balance between events and baseline as discussed in section 2.3.2 Input.

The 31219 samples with events, contain a total of 53600 events, with three different labels: ‘0’,’1’,’2’’, which correspond to three classes: Upstates, Biphasics or Blimps and Unknown/Unclassified, with the number of events per label shown at the pie chart below. To clarify, the label ‘0’ used here is for a true event and not to be confused with the label ‘0’ mentioned in the previous sections, that annotated the absence of an event.

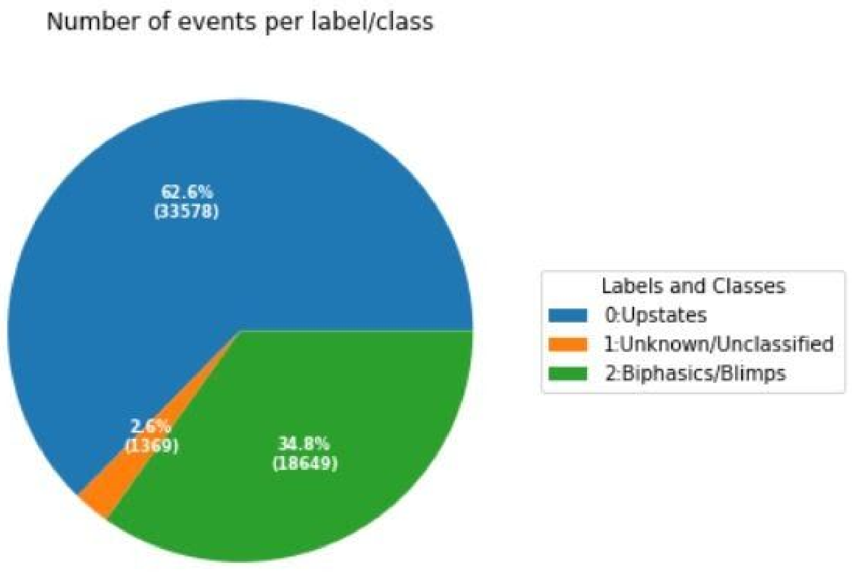


Figure 3.1: Class distribution of the 53600 events from the 62438 samples.

Moreover we calculated the following statistics, since we will need them to set some Hyperparameters later. The statistics for the number of events per sample, were calculated over the samples that contained at least one event. On the other hand, all of the recordings contained at least one event, so we calculated these statistics over all of the recordings.

Table 3.1: Various events' statistics.

	min	average	max
duration (s)	0.10	1.21	9.83
number of events per sample	1	1.71	11
number of events per record	1	70.89	220

Additionally we counted the number of recordings and samples by sex, age and group. As we can see below in figure 3.2, there is an observable asymmetry in the distribution of both recordings and samples, in the case of sex and type. There are 7.3 times more recordings from male mice than female, 6.2 times more samples from male than female, 3.1 times more recordings from WT mice than b2KO and 2.7 times more samples from WT than b2KO. Also, we can clearly see that we have more older mice in both recordings and samples. But it is noteworthy, that the number of samples has a better distribution in the age groups, than the number of the recordings. This, as well as the fact that we have a slightly less difference in sample number than recording number by sex and type, can be credited to the fact that we extract from every recording, samples in twice the amount of the events in the recording. So we can infer that pups and adolescents, as well as females and b2KO mice, exhibit more events per recording, as opposed to adults and old mice, or males and WT correspondingly, at least in our dataset.

Lastly, we can see in figure 3.3 the number of records and samples by drug and by drug and dose. Again, we can observe that mice that have been given Gabazine exhibit more events per sample, compared to other drugs or the non-administration of a drug. This is easily noticeable in both pairs of bar graphs, but especially in the pair of bar graphs about drug and dose. The greater the dose of Gabazine administered to mice, the more the samples, hence the more the events per recording. On the other hand, we can observe that the exact opposite happens with CGP, as the greater the dose, the less the samples per recording. Also, there is a slight increase with descending order in samples per recording for DHbE, DHbe+MLA and MLA. Finally, the dataset is balanced in drug administration, as it contains 31094 samples from mice on which some drug was administered and 31344 samples from mice on which no drug was administered.

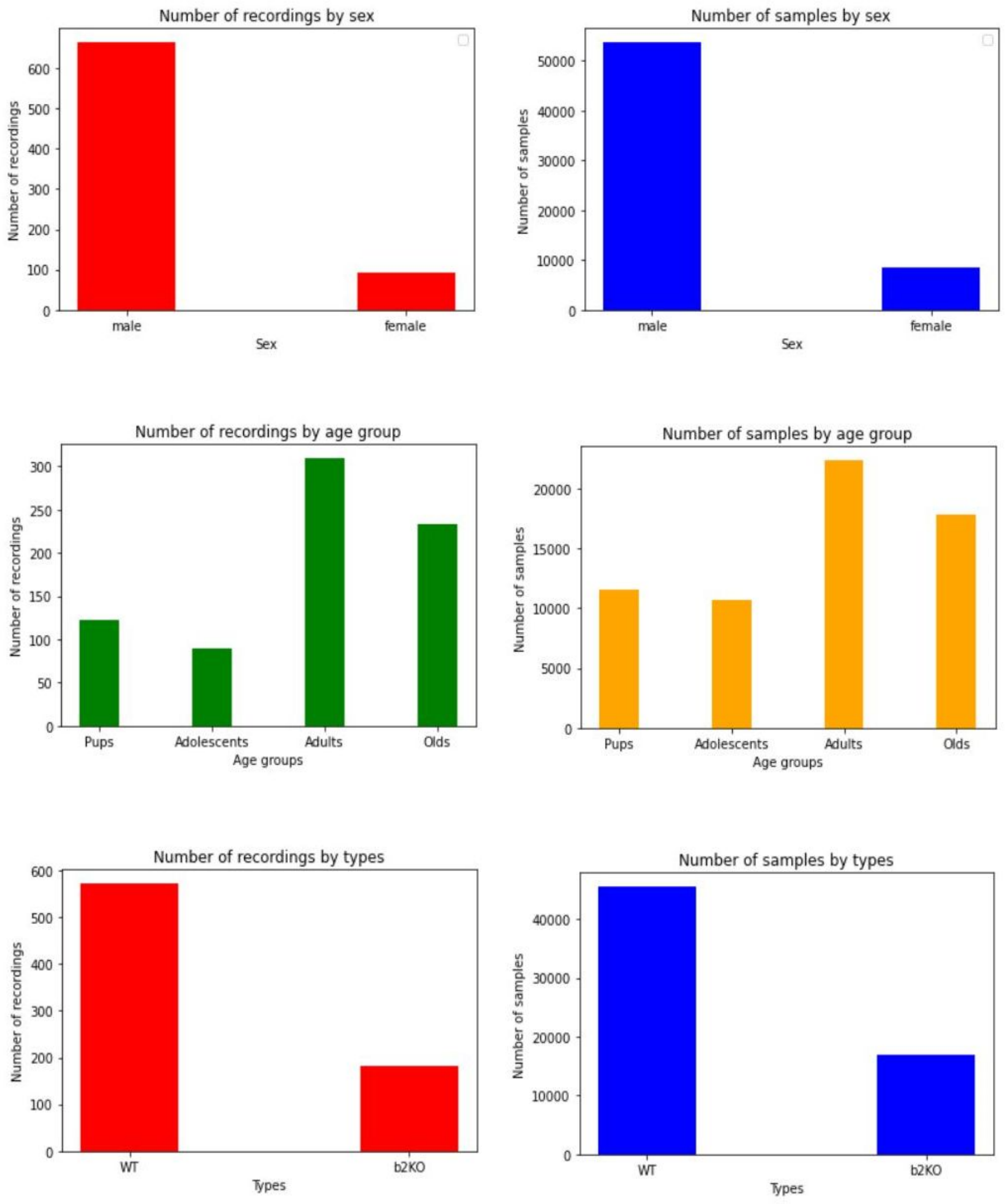


Figure 3.2: Number of recordings and samples by sex, age and type.

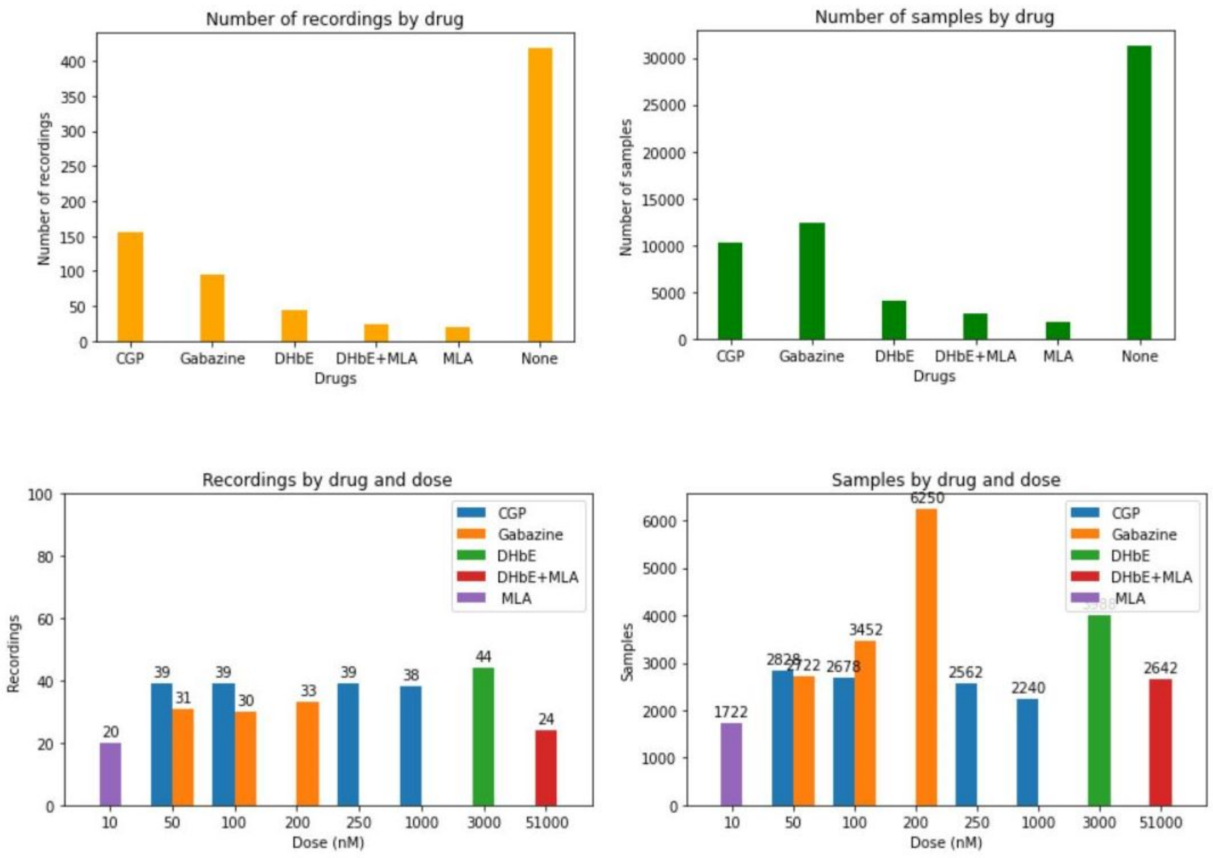


Figure 3.3: Number of recordings and samples by drug and by drug and dose.

Out of the 62438 extracted samples in our dataset, 53072 (85%) were used for training the network, while the other 9366 (15%) were used for validation. Both the training and validation data were balanced between samples with at least one event and samples with no events. The labels of the events on the samples were updated, in respect to the label ‘0’ being reserved from the network to annotate the absence of an event, in the predicted events. Additionally, since LFPAnalyzer was used to detect the onset and offset of the events, that generates a mask for every recording to annotate them, as discussed in section 1.2.1 LFPAnalyzer, we extracted said masks for every sample and stored them along with the actual sample and labels of the events of the sample. The already computed samples’ masks are going to make the computations of the evaluation metrics faster, as discussed in section 3.5. Evaluation Metrics.

3.2. Software and Hardware

The algorithm was fully developed in Python 3. The CNN was built and trained with Tensorflow 2, an end-to-end open source platform for machine learning development and the Tensorflow-integrated Keras deep learning API. The samples were extracted from MATLAB files and stored in csv files. The training of the network was executed on Google Colab, a cloud service used for Python developing, that allows for free access to GPUs, a necessary hardware to accelerate CNN training, since the computations performed by the CNN on its parallel neurons mimics the GPU computation pattern.[69]. Specifically we performed the training on the Google Colab Pro platform, that provides better resources and longer runtime duration for a small monthly fee. The runtime duration is the number of hours you can continuously be logged in the cloud service and execute a program, before the service stops your execution, in order to stop you from abusing the resources and make sure that everyone can have access to them. The difference in resources between the free Colab and Colab Pro can be seen below, in Table 3.2.

Table 3.2: Comparison of the resources available with Google Colab and Google Colab Pro.

	Google Colab	Google Colab Pro
CPU	Intel(R) Xeon(R) CPU @ 2.20GHz	Intel(R) Xeon(R) CPU @ 2.30GHz
GPU	Nvidia Tesla K80	Nvidia Tesla P100 - PCIE - 16GB
RAM	12.8 GB	26.3 GB
Runtime Duration	12 hours	24 hours

3.3. Hyperparameters

The hyperparameters of the algorithm were carefully selected, either by a priori knowledge of the nature of the hyperparameter, or by suggestion of the DOSED paper[8] or by experimental testing.

3.3.1. T timesteps: sample duration and sampling frequency:

The T variable is the number of timesteps of each sample, as explained in the previous section, that is defined as $T = sample\ duration * sampling\ frequency$.

The samples' duration was selected to be 20 s, due to the fact that in our dataset we have observed events of maximum duration: 9.83 s, and we needed to make sure that even in the most extreme case, e.g. a sample with two events of max duration positioned right next to each other, there will be sufficient baseline in each sample. Moreover, increasing the samples' duration more than 20 s, would increase the network's learnable parameters by a huge factor, since the network's size depends on T. That would result in slowing down the network's convergence time.

Additionally, the same would occur if we raised the sampling frequency. Due to the network's architecture, T should be a number that can be expressed as a power of 2. In addition, the lowest sampling frequency observed on our dataset was 5000 Hz, so the only candidates for the sampling frequency were 256, 512, 1024, 2048 and 4096 Hz. Building the network and training it with these sampling frequencies resulted in the following numbers of network parameters and training times. The training time was calculated for 1 epoch with 100 samples.

Table 3.3: Numbers of parameters and training time by Sampling frequency.

Sampling Frequency (Hz)	Non-Trainable Parameters (#)	Trainable Parameters (#)	Total parameters (#)	Training Time (s)
256	20,400	2,303,868	2,324,268	57.83
512	40,804	8,777,516	8,818,320	68.12
1024	81,612	34,233,996	34,315,608	85.26
2048	163,228	135,282,000	135,445,228	107.86
4096	326,460	538,999,536	539,325,996	169.67

The exponential increase in all these properties can be observed in the bar charts below:

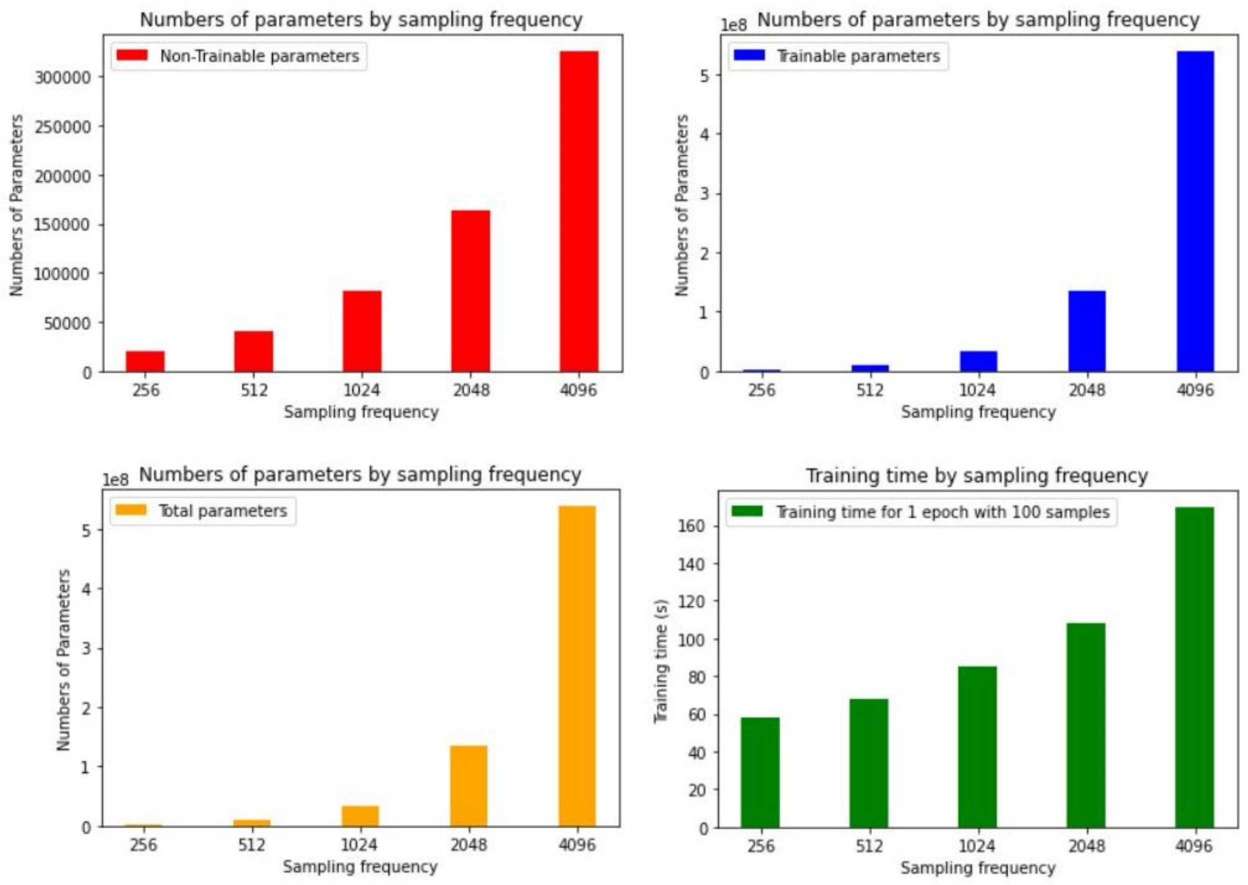


Figure 3.4: Bar charts of the parameters' number and training time by sampling frequency.

The training time's exponential increase, due to the sampling frequency, may not seem extreme on the above bar, but this can be credited to the low samples' number. If we were to train on the whole dataset, we would observe a tremendous increase in training time. Also, the exponential growth of the parameters' number leads to the same growth in memory and computational resources needed both for training and using the network for predictions post-training.

On the other hand, a low sampling frequency can lead to minor inaccuracies on the onset and offset of the detected events. For example, if we choose a sampling frequency of 256

Hz, and we apply it to downsample a recording that had an original sampling frequency of 10000, then the onset and offset detection may be off by a factor of 0 to 39.06.

However, since the inaccuracy factor isn't that great, while the memory and computational resources, as well as the training time differ greatly between different sampling frequencies we selected to run the experiments with a sampling frequency of 256 Hz, given our available hardware. In the future, should someone have more powerful computing systems, with multiple powerful GPUs that allow parallel computation for acceleration, we suggest to train and run the network with a sampling frequency of 1024 Hz, since it has the greatest trade-off between accuracy in onset and offset detection and resource consumption.

3.3.2. Default events: N_d, default event duration and overlap factor:

The default events are characterized by 3 hyperparameters: their number N_d, their duration and their overlap factor.

We define the following function to evaluate the number of default events that will be generated over a sample x:

$$N_{d_{max}} = int\,(\,(\frac{sample\ duration}{default\ event\ duration} - 1) / (1 - overlap\,factor))$$

The intuition behind the above function, comes from the worst case scenario, in which a sample could be populated with as many events it can fit, if they were positioned right next to each other. We want to be able to detect all of them, so we define as such the maximum number of default events needed. The $int(x)$ function returns the closer integer to the float number x.

The default event duration was selected to be equal to 1.21 seconds, the average event duration detected on our dataset, since the sweeping majority of event durations fall around that number. If the default event duration was closer to the maximum event duration of our dataset, then the CNN would struggle greatly to learn to predict the correct duration of true events that exhibit really short durations and vice versa. I.e. it would need more training time to learn to make so big adjustions to the durations of the predicted events. So we think that the selection of the average duration is the best, as it allows for small numeric adjustions on the default duration, with the sign of the adjustion dictating whether the potential event is shorter, when it's minus, or longer, when it's plus, than the default event duration.

The overlap factor has been chosen to be equal to 0.5, as suggested in DOSED[8], meaning that two subsequent default events will share half of their duration. The reason for this is that there could be two different events, or more, right next to each other and we want to avoid one of them remaining undetected. Also, because the events in the recordings, and by extent in the samples, are sparse, a greater overlap factor would result in slower convergence time, as it would affect the default events' number N_d, which would affect the size of the CCN and the computations needed for the evaluation of the loss function on every training and validation step. So the number 0.5, gave us the best trade-off between accuracy in detection and execution time.

Moreover, as mentioned above in section 3.1 Dataset, the average number of true events in a sample on our dataset is 1.71, with the maximum being 11. So the combined selection of the default event duration and overlap factor, gives us more than enough default events to detect any potential events present in a sample x.

Finally, we have:

- $default\ event\ duration = 1.21\ s$
- $overlap\ factor = 0.5$

  and

- $sample\ duration = 20\ s$

hence:

- $N_d = 30$

We selected N_d to be equal to 30, while $N_{d_{max}} = 32$, so that the CNN will have an output layer with a smoother output shape $(2 \times N_d, 1, 1), ((L+1) \times N_d, 1, 1)$:

- $N_d = 30 \Rightarrow$ classification: $((60, 1, 1)$ and localization: $(240, 1, 1))$.

  over

- $N_d = 32 \Rightarrow$ classification: $((64, 1, 1)$ and localization: $(256, 1, 1))$.

where $L = 3$.

3.3.3. C channels and L labels:

For the current experiments, we selected $C = 1$, since our dataset doesn't contain multivariate signals. However, the network was developed as detailed in section 2.3 Our Approach, enabling the use of multivariate signals for training, should they be available in the future. Moreover, as explained in section 2.2.3.1 The function φ_C - Spatial Filtering. when $C = 1$, then the function φ_C becomes the Identity function.

For the purpose of this thesis, we selected to only train the CNN, with $L = 1$. That means that the network will only have to classify among two labels: 0 for baseline and 1 for events of any type. This was chosen as such, due to the fact that training on such a big dataset, with that big a network, requires an enormous amount of time, given our available hardware through Google Colab Pro, and that's more than we have available for an undergrad thesis. However, the CNN was developed to enable classification with more labels, a task that we will continue in the future, as described in section 4. Conclusions and Future Work.

3.3.4. Thresholds for IoU, Loss function and Non-Maximum Suppression:

The algorithm utilizes three thresholds: η_{Loss} the IoU overlap threshold used in the Loss function, η_{NMS} the IoU overlap threshold used in NMS and $\theta_{confidence}$ the confidence threshold used after NMS.

As discussed in section 2.2.2 Loss function, the η_{Loss} hyperparameter is used to perform bipartite matching between the default and true events. It was set to be $\eta_{Loss} = 0.5$, as suggested in DOSED[8].

The hyperparameter η_{NMS}, is used to group potential events, when they exhibit IoU higher than η_{NMS}, in order to later eliminate the ones of the same group, with the lowest confidence scores, i.e. lowest predicted probabilities to contain a true event, as discussed in section 2.3.5 Post-Processing. It was experimentally set as $\eta_{NMS} = 0.9$.

Lastly, the hyperparameter $\theta_{confidence}$ is used right after NMS, so that the algorithm will output only the predicted events, whose probability to contain a true event is greater than $\theta_{confidence}$, as discussed in section 2.3.5 Post-Processing. It was experimentally set as $\theta_{confidence} = 0.2$, as the network excelled at detecting baseline, so anything above 0.2 was indeed a true event.

3.3.5. Batch size and number of Epochs:

The batch size of the CNN, is a crucial hyperparameter that affects the convergence of the algorithm. The batch size dictates how many samples the network will process until it adjusts its learnable parameters, in order to make better predictions in the future. Usually, most DNNs, and by extent most CNNs, have a large batch size, but we observed that larger batch sizes were slowing down the convergence time of the algorithm. This can be credited to the fact that while processing the samples in the large batch, the optimization policy would try to calculate the necessary adjustments to its parameters, but before it could apply them, processing the rest of the samples in the batch would result in unlearning what it learned, so by the time the adjustments were applied they made no significant difference in the efficiency of the network. So we observed that the best three candidates for the batch size were 16, 32 and 64, since the batch size conventionally has to be a power of two. We trained the network with all three and selected 64 as the best choice. The reasons behind our selection are explained in detail in section 3.6.1. Batch size comparison.

As for the number of epochs, i.e. the number of times that the network will process the whole dataset, during training, we selected $epochs = 30$, due to the great amount of time each epoch needs to finish on our available hardware and our lack of available time. We can infer given our results, that for $L = 1$, we would need about 50-60 epochs, for our system to reach the desired efficiency. For $L = 3$, if we were to classify the events with all the available labels in our dataset, we assume that we would need 200 epochs, as suggested in DOSED[8]. We explain in full details, the training time constraints and limitations, in section 3.4 Training.

3.3.6. Learning rate lr and Momentum μ

Learning rate and Momentum are two hyperparameters used by the Stochastic Gradient Descent optimization policy. The first one dictates how much do the trainable parameters change at every step of the training, i.e. after the processing of each batch of samples, while the second is used by a technique that increases the speed of convergence by embedding memory of past changes on the learnable parameters, so the number of Momentum dictates the percentage of these past changes. They were selected to be $lr = 10^{-3}$ and $\mu = 0.9$ as suggested by DOSED[8].

3.4. Training

We trained the CNN with the aforementioned hyperparameters, with three different batch sizes: 16, 32 and 64, whose output is detailed in 3.6. Results. Both the training and validation datasets were randomly shuffled. Optimization techniques were used to speed up the process, such as special operations for computing on GPU and using mixed precision policy to represent the float variables on the calculations, i.e. using float16 instead of float32 or float64, where the numbers 16, 32 and 64 represent the number of bytes available to store the numbers beyond the floating point. We also created a special callback function to store into log files the training loss and validation loss at the end of each batch, and the validation precision, recall and $f1$ metrics, that are detailed below, at the end of every epoch.

The training time for one epoch was about 11 hrs for batch size = 64, about 8 hrs for batch size = 32 and about 6 hrs for batch size = 16. Since Google Colab Pro enforces a random timeout almost every 22-24 hours, with a waiting time to log back in of about 4-6 hours, we saved the parameters of the network to a log file every 10000 samples and we loaded them every time we had to restart the training process.

3.5. Evaluation Metrics

For the evaluation of the algorithm's efficiency we are going to use precision, recall, and f1 score, which can be given by the following functions:

- $precision = \frac{true\ positive}{true\ positive + false\ positive}$
- $recall = \frac{true\ positive}{true\ positive + false\ negative}$
- $f1 = 2 * \frac{precision * recall}{precision + recall}$

Since, in most cases, when precision increases then recall decreases and vice versa, the f1 score is the most objective metric to evaluate the algorithm's efficiency.

Based on the nature of our input data, we have developed the following evaluation metric to compute the $f1$ score, that is accelerated through the use of masks.

As described in section 1.2.2 LFPAnalyzer, we define as a mask, a signal $m : \Re \rightarrow \{0, 1\}$ that is generated over an LFP sample x, to annotate its events:

- $m(t) = 1$, if t is a timestep in x, where $x(t)$ is annotated as an event.
- $m(t) = 0$, if t is a timestep in x, where $x(t)$ is annotated as baseline.

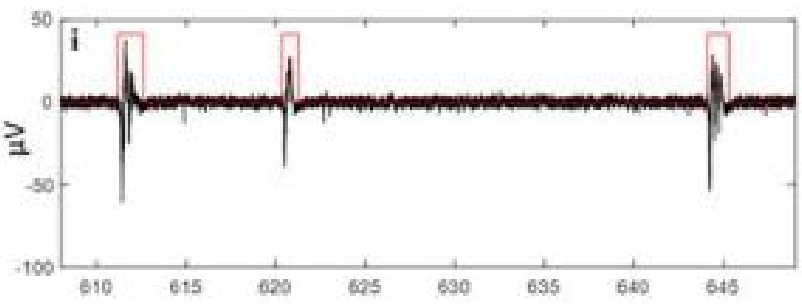


Figure 3.5: A mask (in red) generated over an LFP sample (in black).

Since the masks on our dataset are already computed for the true events, at runtime we need only to compute the $predicted\,mask : \mathfrak{R} \rightarrow \{0, 1\}$ for the predicted events, as described above, and compare it with the $true\,mask : \mathfrak{R} \rightarrow \{0, 1\}$.

- $true\,mask(t) = m(t)$, when the annotated events of x are true events.
- $predicted\,mask(t) = m(t)$, when the annotated events of x are predicted events

To compare the two masks, we will create a new mask that we will call $metric\,mask : \mathfrak{R} \rightarrow \{-1, 0, 1, 2\}$:

- $metric\,mask(t) = 2 * predicted\,mask(t) - true\,mask(t)$

Now we can assign to each of the four numbers that the $metric\,mask$ can output: {-1,0,1,2}, one of the four characterizations for our predictions: {false negative, true negative, true positive, false positive} that we need in order to compute the precision and recall and finally the $f1$ score, with the following algorithm:

```
● $false\,negatives = 0$
● $true\,negatives = 0$
● $true\,positives = 0$
● $false\,positives = 0$
● for $t$ in sample $x$:
    ● if $metric\,mask(t) = -1$
        ● then $false\,negatives = false\,negatives + 1$
    ● if $metric\,mask(t) = 0$
        ● then $true\,negatives = true\,negatives + 1$
    ● if $metric\,mask(t) = 1$
        ● then $true\,positives = true\,positives + 1$
    ● if $metric\,mask(t) = 2$
        ● then $false\,positives = false\,positives + 1$
```

Thus, for every t timestep in a sample x, we can annotate our prediction on t to be either a false negative, a true negative, a true positive or a false positive predicted timestep. Summing each of these timesteps will give us the final number of false negatives, and so on, for the prediction on the sample x, and we can proceed to compute precision, recall and $f1$ score with the functions described above.

Below on figures 3.6-9, we show some examples of applying the aforementioned evaluation metric on different predictions on a pair of samples, one that contains true events, and one that contains only baseline and we compute the combined precision, recall and $f1$ score for the predictions on both samples.

The main benefits of this method are mainly computational. As we mentioned, the $true\,mask$ is precomputed. The complexity of computing the $prediction\,mask$ is $O(T)$ comparisons, since we need to create T timesteps. The complexity of computing the $metric\,mask$ is $O(T)$ multiplications between integers, which is really faster than multiplying float numbers, and $O(T)$ subtractions. The calculation of the false positives, etc, is $O(T)$ comparisons between integers.

So the total complexity of computing the false positives, etc, for one sample is:

- $O(2 * T)$ *comparisons*
- $O(T)$ *integer multiplications*
- $O(T)$ *integer subtractions*

The complexity of the evaluation metric over a set of n validation samples:

- $O(n * 2 * T)$ comparisons
- $O(n * T)$ multiplications between integers + $O(2)$ multiplications between floats (for the computation of $f1$)

  = $O(n * T)$ multiplications between integers
- $O(n * T)$ subtractions between integers + $O(4 * (n - 1))$ additions between integers (for the computation of the sum of false positives etc) + $O(3)$ additions between floats (for the computation of the divisors of precision, recall and $f1$)

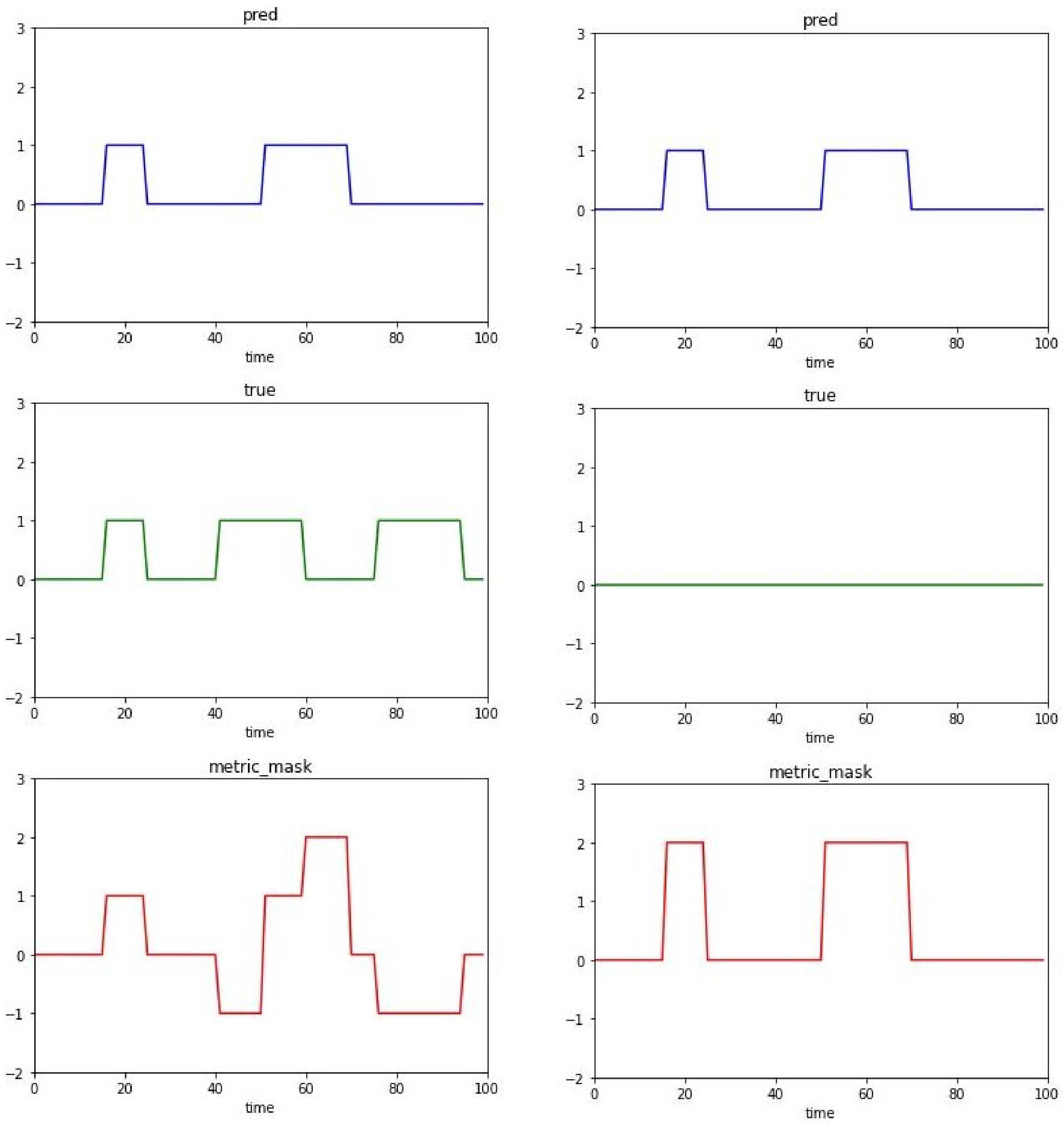
  = $O(n * T)$ subtractions between integers
- $O(2)$ divisions between integers (for the computation of precision and recall) + $O(1)$ divisions between floats (for the computation of $f1$)

  = $O(3)$ divisions

And finally, the total complexity of the evaluation metric over a set of n validation samples:

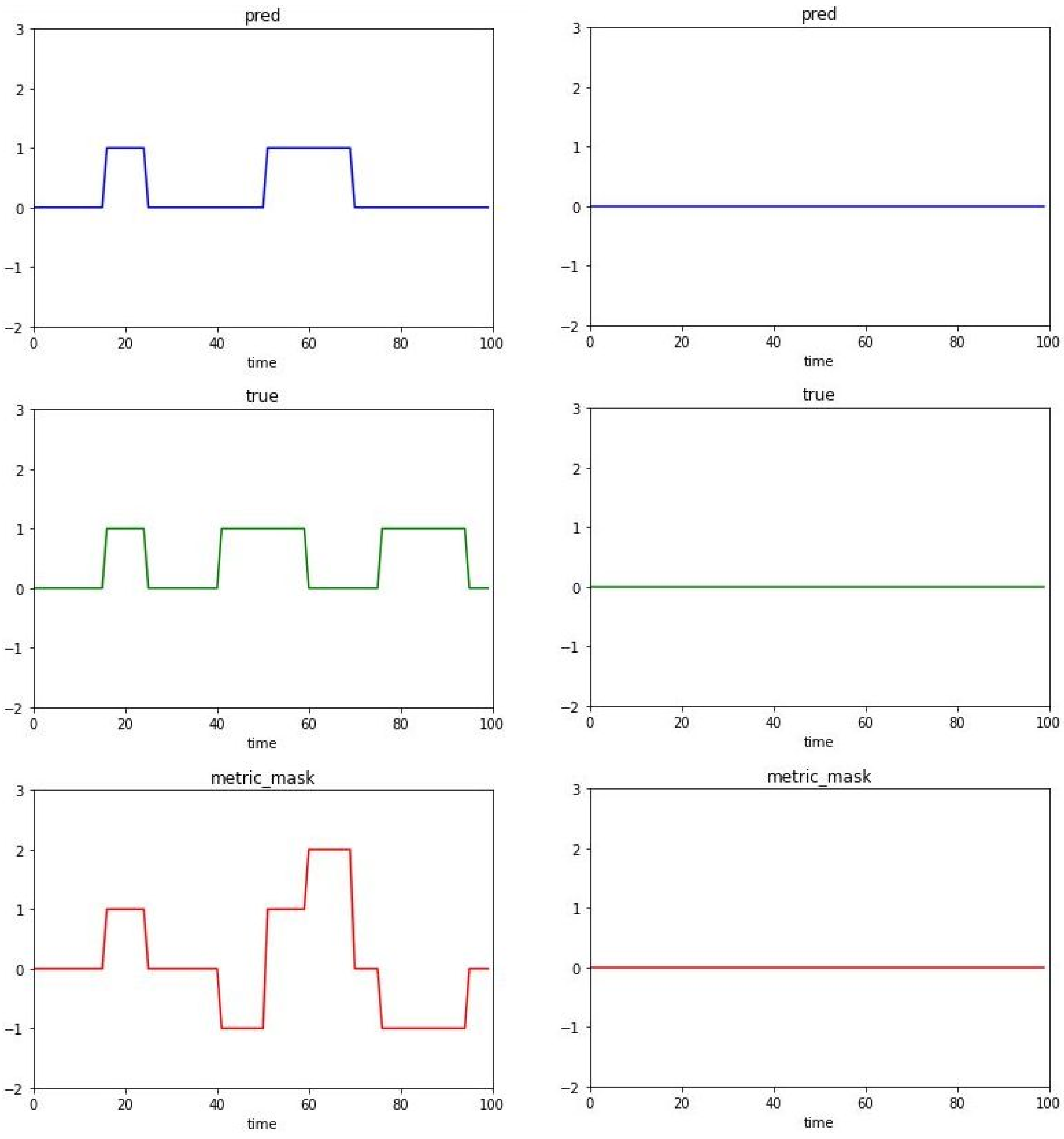
- $O(2 * n * T) + O(n * T) + O(n * T) + O(3) = O(4 * n * T)$

Additionally, this method utilizes the least possible memory resources, since all the calculations except 6 are between integers.



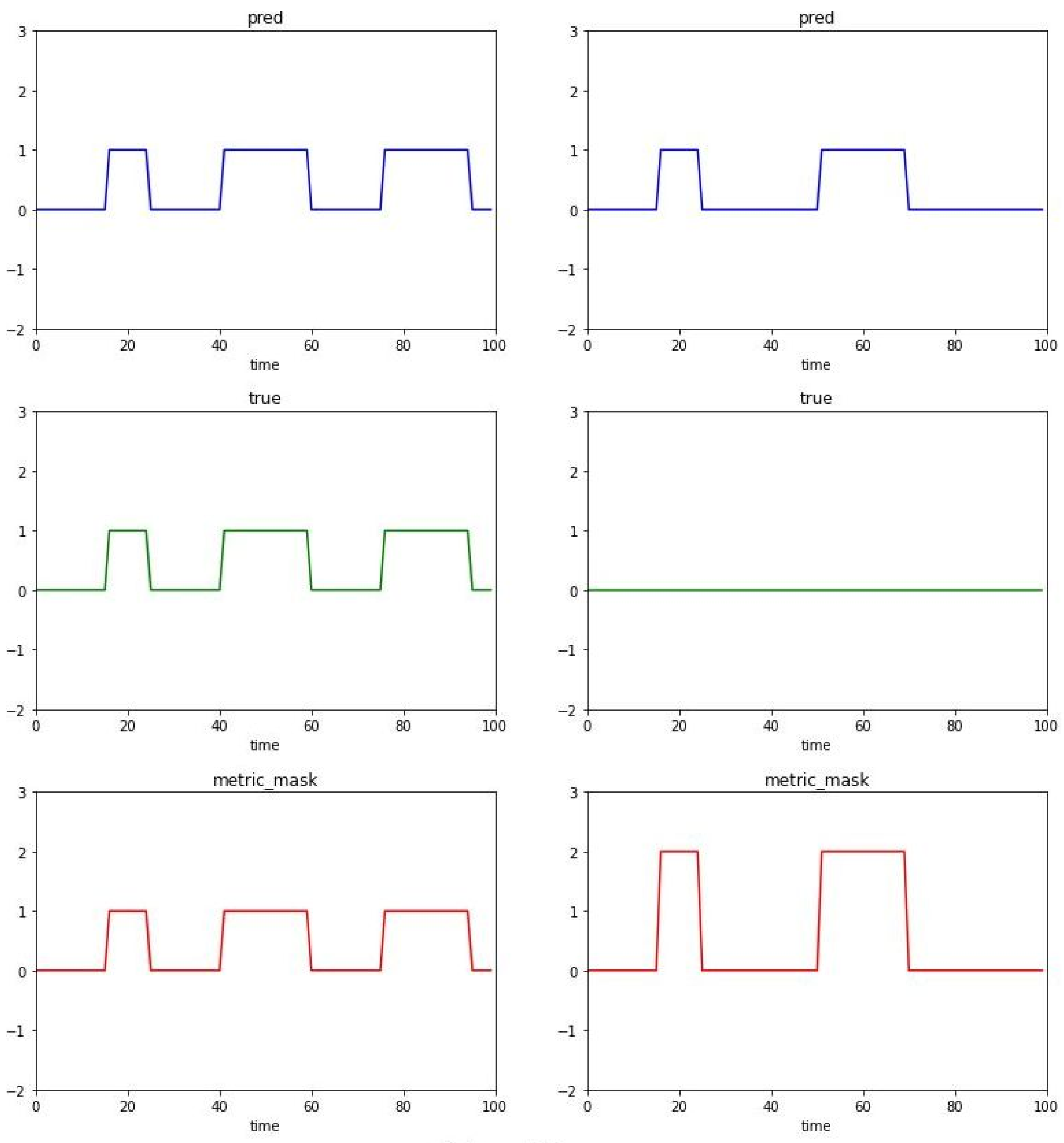
```
precision:  0.32142857142857145
recall:  0.3829787234042553
f1:  0.3495145631067961
```

Figure 3.6: The Evaluation metric applied over two samples, one that contains events (left column), one that contains only baseline (right column) and their combined precision recall and f1 at the bottom, when both predictions were wrong.



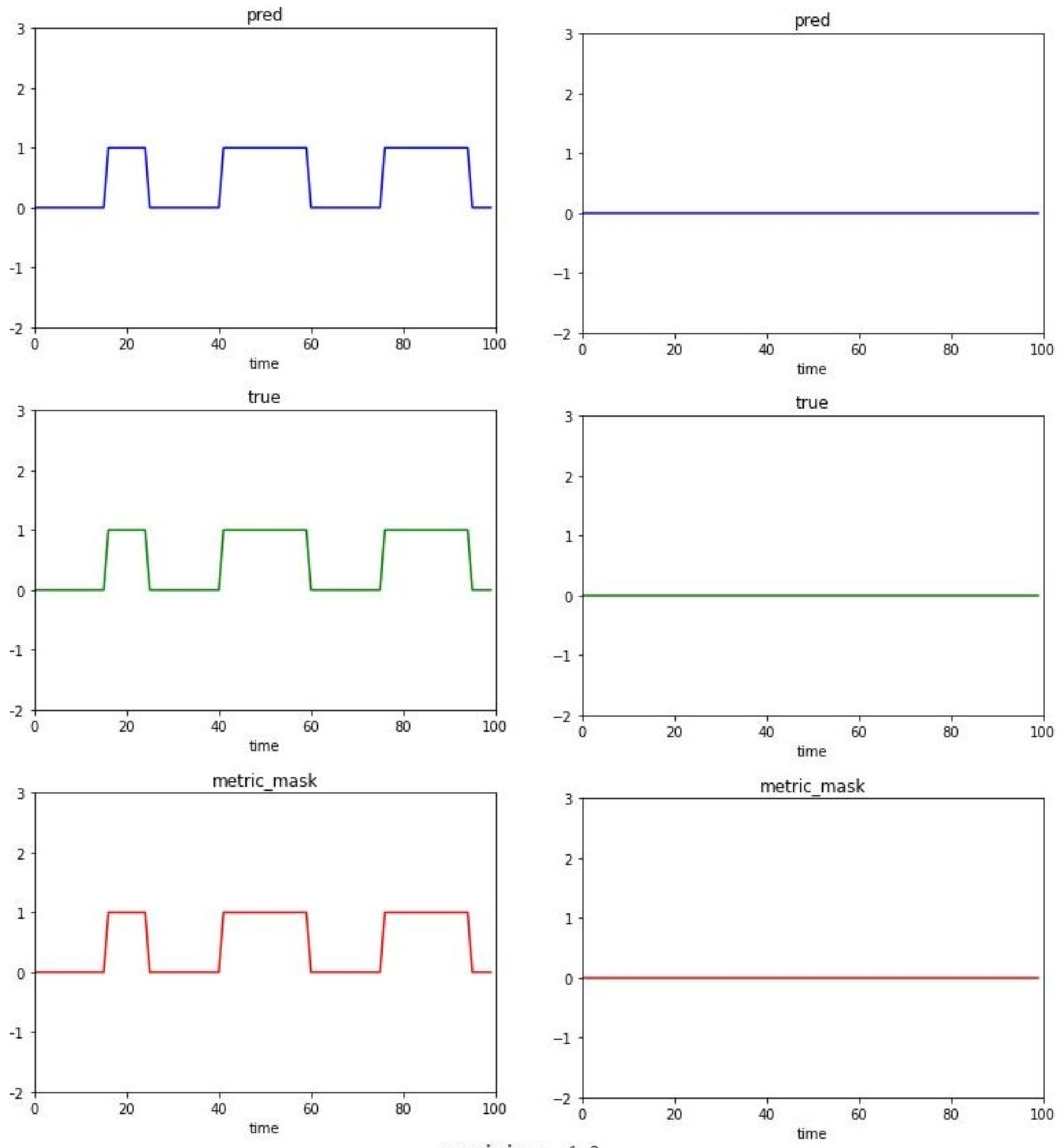
```
precision:  0.6428571428571429
recall:  0.3829787234042553
f1:  0.48
```

Figure 3.7: The Evaluation metric applied over two samples, one that contains events (left column), one that contains only baseline (right column) and their combined precision recall and f1 at the bottom, when the 1st prediction is wrong and the 2nd is correct.



```
precision:  0.5
recall:  1.0
f1:  0.6666666666666666
```

Figure 3.8: The Evaluation metric applied over two samples, one that contains events (left column), one that contains only baseline (right column) and their combined precision recall and f1 at the bottom, the 1st prediction is correct and the 2nd is wrong.



```
precision:  1.0
recall:  1.0
f1:  1.0
```

Figure 3.9: The Evaluation metric applied over two samples, one that contains events (left column), one that contains only baseline (right column) and their combined precision recall and f1 at the bottom, when both predictions were correct.

3.6. Results

After training the network for 30 epochs, we achieved the following maximum efficiency with batch size = 64, for the validation data :

- $f1$: 0.679659
- precision: 0.688305
- recall: 0.671227
- loss: 0.0977

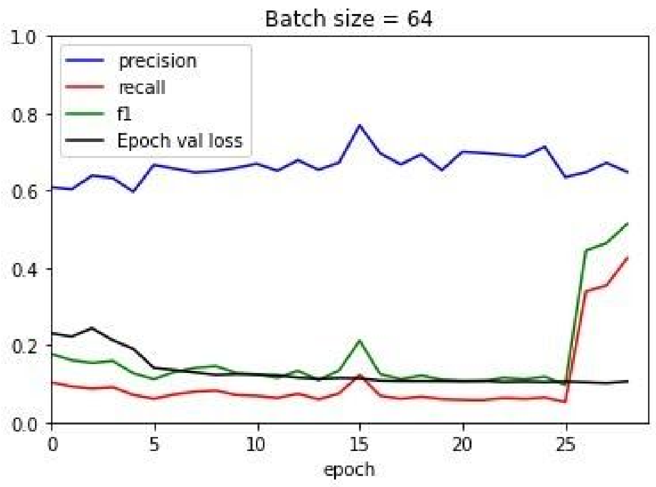


Figure 3.10: Plot of precision, recall, f1 and validation loss by epochs.

The sudden increase of both recall and $f1$, shown at figure 3.10, happened because we fine-tuned the hyperparameters η_{NMS} and $\theta_{confidence}$ after the training had already reached the 25th epoch. Thus, the evaluation metrics showed that our system had already achieved great efficiency and should we had the time to train for more epochs, it would improve even more.

On the next page, in Figure 3.11, we can see the actual results of detection and classification on ten samples:

a) This sample contained a single event, which was perfectly detected.
b) This sample contained only baseline, and no events were detected.
c) This sample contained one event near its start. Its offset was perfectly detected, but not its onset.
d) This sample contained one event near its end. Its duration was overestimated by our system.
e) This sample contained two events. The first one's duration was overestimated, but its center was perfectly predicted. The second one was perfectly detected.
f) This sample contained two events. The first one's duration was perfectly estimated, but not its center. The second one wasn't detected at all.
g) This sample contained two events. The first one was located right next to the start of the sample and it was perfectly detected. The second one had both the onset and offset wrong.
h) This sample contained two events. The first one had correct offset, but wrong onset, while the second one had correct onset, but wrong offset.
i) This sample contained one really short event whose duration was overestimated.
j) This sample contained only baseline, but our system wrongly predicted an event.

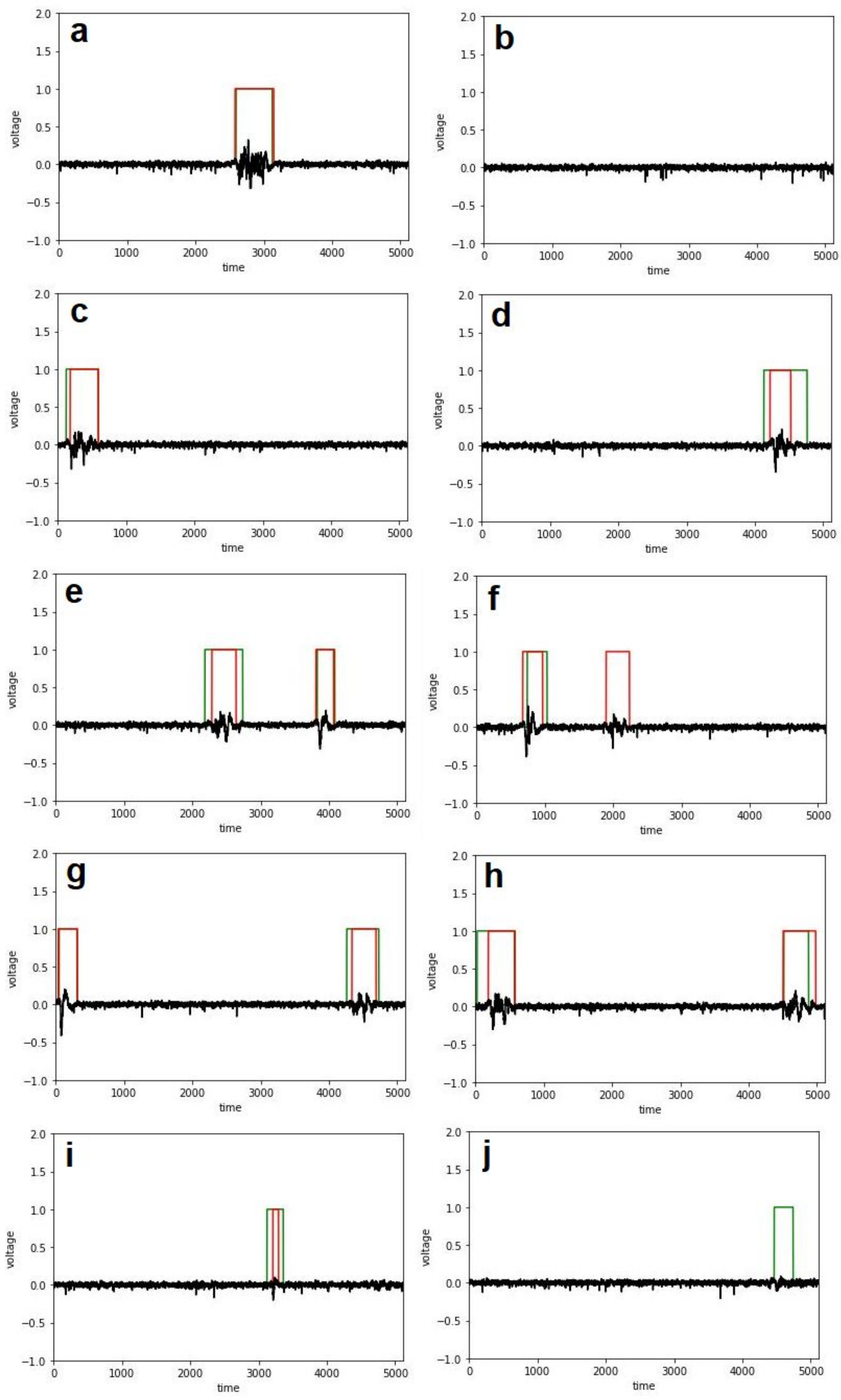


Figure 3.11: Our system's output on 10 samples. True mask in red, predicted mask in green.

3.6.1. Batch size comparison

We have trained our system with three different batch sizes: 16, 32 and 64. Here we present the results, comparing the three approaches.

Table 3.4: Best observed loss and evaluation metrics by batch size.

	Batch size = 16	Batch size = 32	Batch size = 64
Validation loss	0.1109	0.1040	0.0977
Precision	0.697034	0.635029	0.688305
Recall	0.624974	0.626305	0.671227
$f1$	0.659040	0.6738911	0.679659

On Figure 3.12 we have plotted various losses by batch size:

On the left column of Figure 3.12:

- Panel 1, we have plotted the batch loss' value by batch for the three different batch sizes. The batch loss is the training loss that is computed at the end of every batch.
- Panel 2, we have plotted the epoch loss' value by epoch for the three different batch sizes. The epoch loss is the training loss that is computed at the end of every epoch.
- Panel 3, we have plotted the epoch val loss' value by epoch for the three different batch sizes. The epoch val loss is the validation loss that is computed at the end of every epoch.

On the right column - panels 1-3, we have plotted the epoch training and validation losses for each of the three batch sizes.

While on the batch and epoch loss plots of the left column, we observe that the losses decrease faster for batch sizes 16 and 32 than 64, on the epoch val loss plot we observe the exact opposite. Moreover, we can see on the panels of the right column, that the epoch loss and epoch val loss converge to one another the fastest when the batch size is 64. These facts show as that the selection of the batch size as 64, proves to be the best choice to avoid overfitting, i.e. to avoid the network of making great predictions over the training data, data that it has already learned from and processed a lot of times, while making poor predictions over the validation data, that it hasn't learned from, and didn't specialize on. The overfitting problem is one of the major problems in DL, as when it is observed, the DNN fails to generalize well to new "unseen" data.

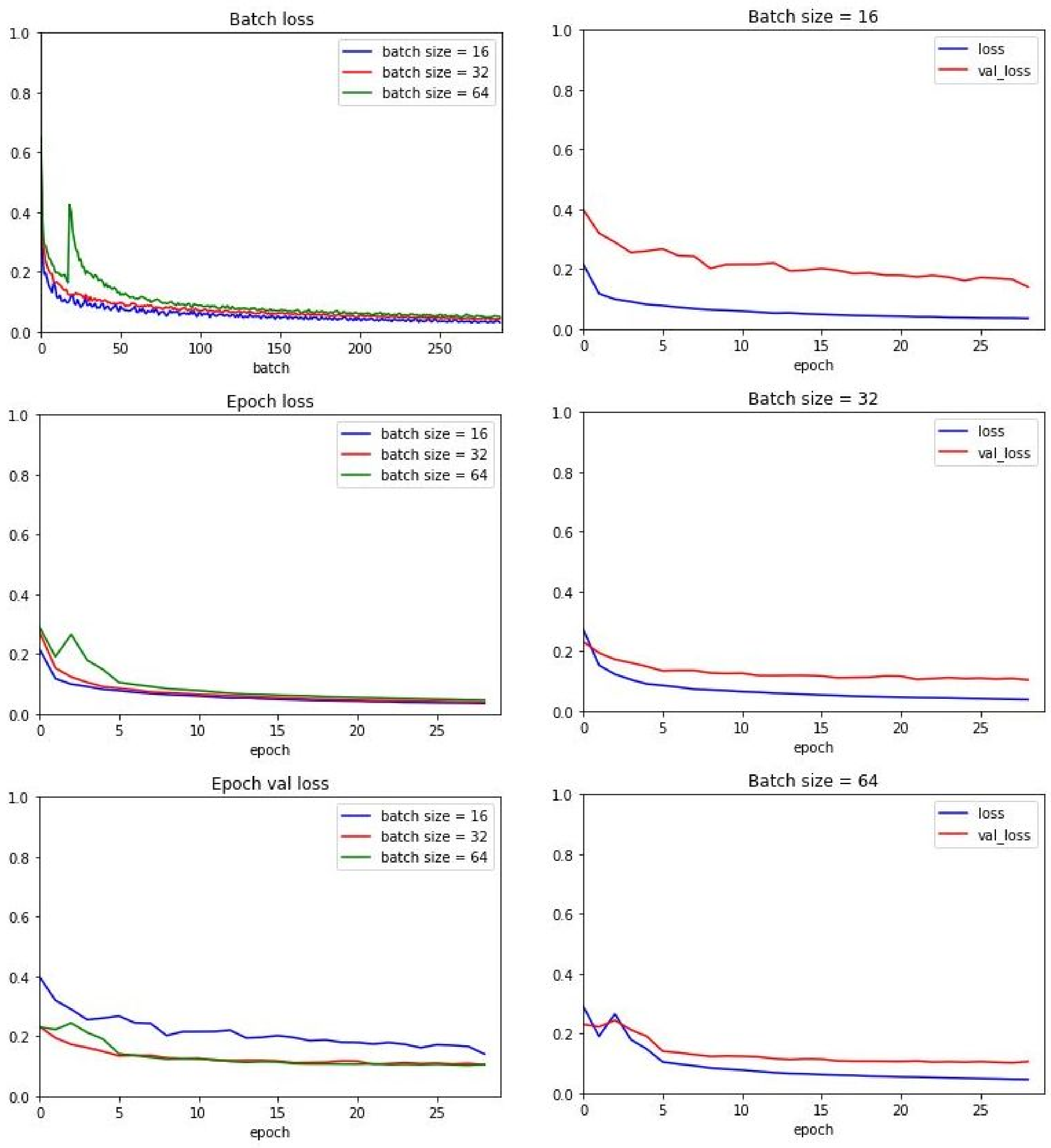


Figure 3.12: Comparison of various loss functions' values for the various batch sizes.

On Figure 3.13 we compare the evaluation metrics of our approach by batch size.

On the left column we have plotted the three metrics, precision, recall and $f1$ by epoch, for the three batch sizes, while on the right column we have plotted together the three metrics by epoch, for each of the batch sizes.

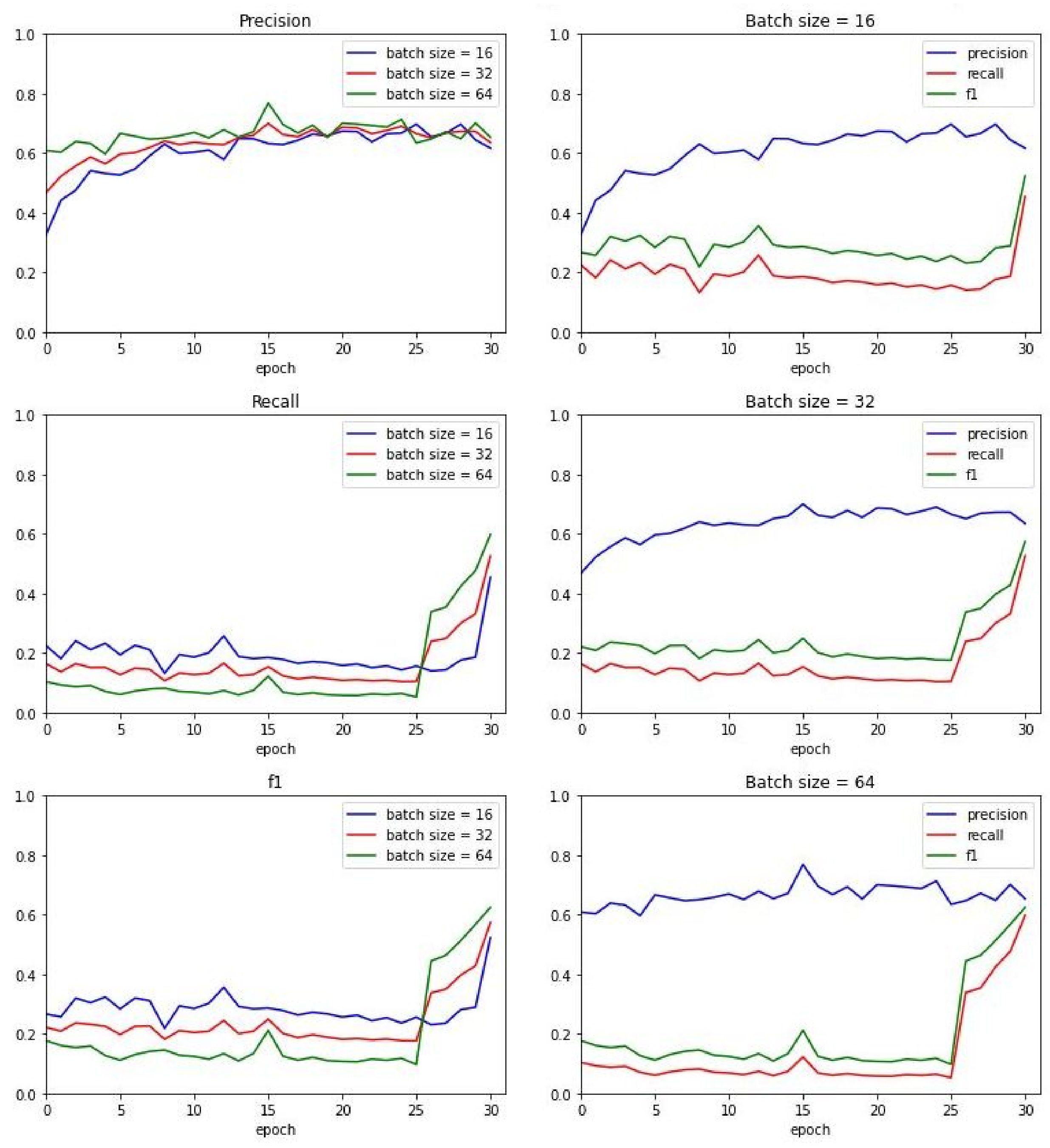


Figure 3.13: Comparison of the evaluation metrics for the various batch sizes.

Here we can see clearly that the batch size of 64 outperforms the rest on all three evaluation metrics. Again, the sudden increase in recall and $f1$ is observed due to the fine-tuning of the aforementioned thresholds after epoch 25. The reason that before epoch 25, the others seemed to outperform the network with batch size = 64, is that they provided better classification scores, in the sense that they gave higher probabilities to potential events that actually contained true events, than the probabilities provided by the other system. As we explained in section 3.3.5 Batch size and number of Epochs, we observed that as the batch size increases, the “learning” during each batch of data seems to decrease. The most plausible explanation is that the greater the batch size, the greater the number of samples with only baseline in it, so it balances out the learning of detecting the true events, as the samples with baseline have little to no difference among each other and as such are reinforced, while the samples that contain events differ greatly among each other and as such are weakened out. In the end, the greater the number of batch size, the better the network learns to detect baseline over true events. But as shown above, in the long run that is the preferable choice. By epoch 25, the network has learned to detect baseline so well, that it gives really high probabilities to segments not containing an event, i.e. it is very confident in detecting baseline. Thus, we can lower the $\theta_{confidence}$ threshold for detecting events, mentioned in section 3.3.4 Thresholds, to 0.2, and everything with higher probability is actually a true event.

Moreover, selecting a greater batch size than 64 would result in slower convergence time for our system, as explained in section 3.3.5 Batch size and number of Epochs and would favor the baseline detection too much over the event detection.

To sum up, the choice of a batch size of 64 is the best one as it provides the best trade-off between the following factors:

- Overfitting avoidance
- Performance of evaluation metrics
- Training time
- Balance in baseline and event detection

3.6.2. Efficiency vs LFPAnalyzer

On Figure 3.14 we compare the results of our approach with the ones from LFPAnalyzer:
We can see on the bar chart the difference in number of detected events between the two approaches. Then, on the other three panels we have created three scatter-plots of the differences on durations, onsets and offsets of the detected events. For all three we plotted the numbers of our approach minus the ones from LFPAnalyzer.

Even though we only managed to train our system for only 30 epochs, we have achieved a satisfactory efficiency, as shown by the distributions on the scatter-plots.

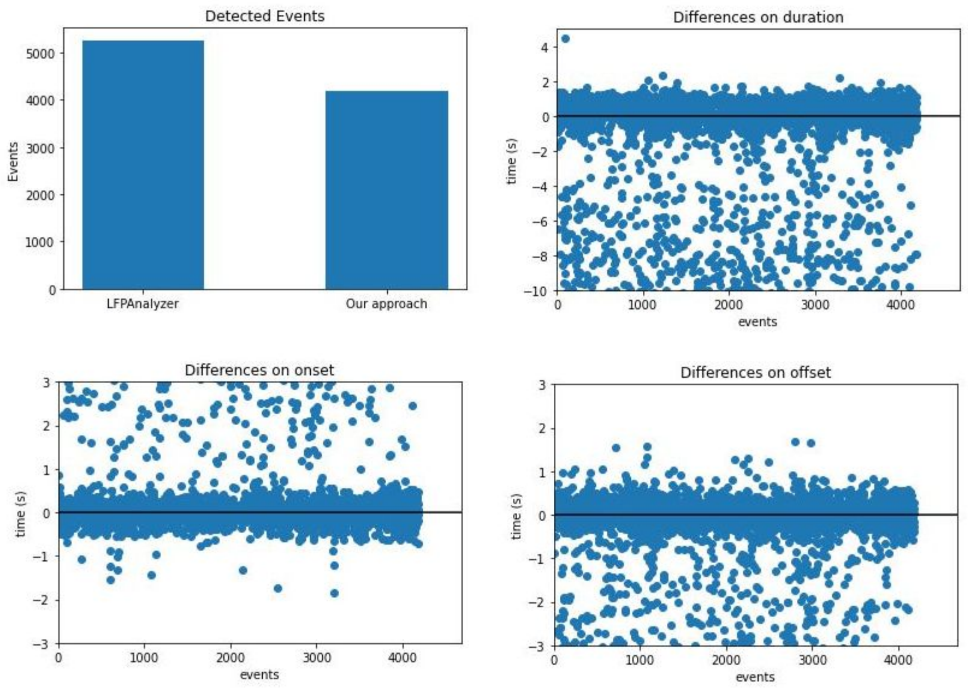


Figure 3.14: Comparison of our approach with LFPAnalyzer.

3.6.3. Computational Time vs LFPAnalyzer

Our system may need a great amount of time to train the CNN, but it only needs less than 500 ms to process a sample, in order to make a prediction. When presented with a full LFP recording of approximate duration of 20 mins, the system extracts 80 samples of 20 s duration, with an overlap factor of 0.25, in order to avoid an event getting caught in between two consecutive samples and remaining undetected. So the average time to predict the onset and offset of the events in the LFP recording is well under a minute, as the processing time of the 80 samples is 40 s at most, and the extraction of the samples and their recombination into the original recording is almost instantaneous. Consequently, our approach needs half the time to predict the onsets and offsets of events in a 20 min LFP recording than LFPAnalyzer, that performs the same task in under 2 mins.[16]

4. CONCLUSIONS AND FUTURE WORK

In this thesis, we explored how a signal processing problem, that of event detection and classification of in-vitro LFP electrophysiological signals, can be handled as a Computer Vision problem and solved with Deep Learning algorithms. We build a system, inspired by the DOSED[8] architecture, that relies on a CNN to learn the task at hand. We explained the mathematical foundations of the algorithm and how it can be implemented in Python using state-of-the-art software and hardware, like Tensorflow, Keras, Google Colab Pro and a powerful nvidia GPU. We created a dataset of LFP samples, that were extracted from LFP recordings, that contained events, whose onset and offset were detected using the LFPAnalyzer software and their labels were annotated by human experts. We discussed the preprocessing steps required to speed up the training period of the CNN, like DC offset removal and application of a low-pass Butterworth filter, as well as a magnification process, to help the CNN "see" the events better. Additionally, we presented the way to feed the preprocessed samples into the network for training. We explained how the network learns to create feature maps of the input data and utilizes the default events generated over each sample, with the aim of outputting the potential predicted events. Furthermore, we discussed the importance of NMS as a post-processing step, to finalize the output of the algorithm by extracting a unique solution from the CNN's output. Moreover, we fully examined the properties of our dataset and explained in detail the reasons behind the selection of the hyperparameters of our algorithm. We tackled the limitations of training a large CNN with a big dataset and discussed our tailor-made method for evaluating the efficiency of the algorithm.

We presented our results, when training with three different batch sizes and showed how fast our system learns to discriminate between events and baseline. We then compared our results in detection of events' onsets and offsets, as much as duration, with the ones of LFPAnalyzer and showed that our approach performs this task in half the time.

The results presented here are encouraging and leave room for additional improvements. In the future, we are going to further train our system in detection, to allow it to reach its full potential in efficiency, and use it to classify against all available classes. Should we have the necessary hardware, we are going to train the CNN with the sampling frequency of 1024 Hz, the one that has the best trade-off between accuracy and convergence time. Another direction worth exploring, would be whether a multivariate signal would improve the performance of the algorithm, by speeding up the training period.

TABLE OF TERMINOLOGY

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
Signal Processing	Επεξεργασία Σήματος
Electrophysiological Signals	Ηλεκτροφυσιολογικά Σήματα
Event Related Potential	Δυναμικό Σχετικό με Γεγονός
Electroencephalogram	Ηλεκτροεγκεφαλογράφημα
Local Field Potential	Δυναμικό Τοπικού Πεδίου
Machine Learning	Μηχανική Μάθηση
Artificial Intelligence	Τεχνητή Νοημοσύνη
Artificial Neural Networks	Τεχνητά Νευρωνικά Δίκτυα
Deep Learning	Βαθιά Μάθηση
Deep Neural Networks	Βαθιά Νευρωνικά Δίκτυα
Computer Vision	Μηχανική Όραση
Convolutional Neural Networks	Συνελικτικά Νευρωνικά Δίκτυα
Event Detection	Ανίχνευση Γεγονότων
Classification	Κατηγοριοποίηση
in-vitro	στο-γυαλί - “σε μη ζωντανό οργανισμό”
in-vivo	“σε ζωντανό οργανισμό”
in-silico	στον-υπολογιστη - “σε προσομοιωμένο οργανισμό”
Non-Maximum Suppression	Μη Μέγιστη Καταστολή
Intersection over Union	Τομή προς Ένωση
Loss function	Συνάρτηση κόστους
Back propagation	Οπίσθια Διάδοση
Stochastic Gradient Descent	Στοχαστική Κλίση Κατάβασης
Spatial Filtering	Χωρικό φιλτράρισμα
Temporal Feature Extraction	Εξαγωγή Χρονικών Χαρακτηριστικών
Predictor	Προβλεπτής
Evaluation Metric	Μετρική Αξιολόγησης

ABBREVIATIONS - ACRONYMS

ERP	Event Related Potential
EEG	Electroencephalogram
LFP	Local Field Potential
ML	Machine Learning
ANNs	Artificial Neural Networks
DL	Deep Learning
DNNs	Deep Neural Networks
CNNs	Convolutional Neural Networks
GMM	Gaussian Mixture Model
SD	Standard Deviation
DOSED	Dreem One Shot Event Detector
NMS	Non-Maximum Suppression
IoU	Intersection over Union
ReLU	Rectified Linear Unit
WT	Wild Type
b2KO	b2 Knockout
GPU	Graphical Processing Unit

REFERENCES

[1] Roland Priemer, Introductory Signal Processing, Volume 6 of *Advanced series in electrical and computer engineering*, World Scientific, 1991. p. 1.
[2] Gabriel Vasilescu, *Electronic Noise and Interfering Signals: Principles and Applications*, Springer Science & Business Media, 2005, p. 3.
[3] L. Martignon, *International Encyclopedia of the Social & Behavioral Sciences*, Pergamon Press, 2001, pp. 7476-7480.
[4] Suranai Poungponsri, Xiao-HuaYu. *An adaptive filtering approach for electrocardiogram (ECG) signal noise reduction using neural networks*, Neurocomputing, Volume 117, 2013, Pages 206-213, Fig. 11. Combined noise removal. (a) Signal with combined noise and (b) signal after filtering.
[5] S A Hillyard, M Kutas, *Electrophysiology of Cognitive Processing*, Annual Review of Psychology Vol. 34,1983, p 33-34
[6] Luck, S.J.; Kappenman, E.S., *The Oxford Handbook of Event-Related Potential Components.* Oxford University Press, 2012, p. 664.
[7] J. M. Dunn, et al, Independent component analysis for functional MRI. Appendix E in *Understanding the abnormal brain activity in epilepsy as a potential predictor of the onset of an epileptic seizure*, The ANZIAM Journal 52, 2011, Fig.1 Electroencephalography (eeg) data showing onset of an epileptic event.
[8] S. Chambon et al., *DOSED: A deep learning approach to detect multiple sleep micro-events in EEG signal,* Journal of Neuroscience Methods Volume 321, 2019, p 64-78.
[9] Buzsáki, G. *Large-scale recording of neuronal ensembles*. Nature Neuroscience 7, 2004, p. 446–451.
[10] Mitzdorf, U. *Current source-density method and application in cat cerebral cortex: investigation of evoked potentials and EEG phenomena*. Physiological Reviews 65, 1985, p. 37–100.
[11] Logothetis, N. K. *What we can do and what we cannot do with fMRI*. Nature 453, 2008, p. 869–78.
[12] Senkowski, D. et al. *Multisensory processing and oscillatory activity: analyzing non-linear electrophysiological measures in humans and simians.* Exp. Brain Res. 177, 2007, p. 184–195.
[13] Belitski, A. et al. *Low-frequency local field potentials and spikes in primary visual cortex convey independent visual information.* J. Neuroscience 28, 2008, p. 5696–5709.
[14] Denker, M. et al. *The local field potential reflects surplus spike synchrony*. Cerebral Cortex. 21, 2011, p. 2681–2695.
[15] Mazzoni, A., et al. *Encoding of naturalistic stimuli by local field potential spectra in networks of excitatory and inhibitory neurons. PLoS Computational Biology 4, 2008.*
[16] *Skaliora I. et al. High-Throughput Analysis of in-vitro LFP Electrophysiological Signals: A validated workflow/software package.* Nature: Scientific Reports 7, 2017, Article number 3055.
[17] Russell, Stuart J.; Norvig, Peter. *Artificial Intelligence: A Modern Approach* (Third ed.). Prentice Hall. 2010. section V Learning p. 693-853
[18] https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7
[19] https://towardsdatascience.com/neural-networks-parameters-hyperparameters-and-optimization-strategies-3f0842fac0a5
[20] https://medium.com/@jamesdacombe/an-introduction-to-artificial-neural-networks-with-example-ad459bb6941b
[21] Athanasios Voulodimos, et al. *Deep Learning for Computer Vision: A Brief Review* , Computational Intelligence and Neuroscience, vol 2018, Article ID 7068349
[22] Zixing Zhang et al. *Deep Learning for Environmentally Robust Speech Recognition: An Overview of Recent Developments* , ACM Transactions on Intelligent Systems and Technology, 2018, article No: 49
[23] Dan Stowell, et al., *Automatic acoustic detection of birds through deep learning: The first Bird Audio Detection challenge*, Methods in Ecology and Evolution, vol 10 issue 3, 2018, p.368-380.
[24] Tom Young, et al. *Recent Trends in Deep Learning Based Natural Language Processing*, IEEE Computational Intelligence, vol 13 issue 3, 2018, p. 55-75.
[25] Ashish Vaswani, et al, *Tensor2Tensor for Neural Machine Translation*, Cornell University eprint 1803.07416, arXiv.org, 2018
[26] Yankang Jing, et al., *Deep Learning for Drug Design: an Artificial Intelligence Paradigm for Drug Discovery in the Big Data Era*, The AAPS Journal vol 20, article number: 58, 2018.
[27] Kristian M. Black, et al. *Deep learning computer vision algorithm for detecting kidney stone composition*, BJUI vol.125 issue 6, 2020, p. 920-924.

[28] Seonwoo Min, et al. *Deep learning in bioinformatics,* Briefings in Bioinformatics, vol 18, issue 5, 2017, p. 851–869.
[29] David Silver, et al., *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*, Science vol 362 issue 6419, 2018, p. 1140-1144
[30] OpenAI and Christopher Berner, et al. *Dota 2 with Large Scale Deep Reinforcement Learning,* Cornell University eprint 1912.06680, arXiv.org, 2019.
[31] Russell, Stuart J.; Norvig, Peter. *Artificial Intelligence: A Modern Approach* (Third ed.). Prentice Hall. 2010. chapter 24 Perception, p. 928-965
[32] Zhijun Fang, et al. *Abnormal event detection in crowded scenes based on deep learning*, Multimedia Tools and Applications vol 75, 2016, p. 14617–14639
[33] de Polavieja G. et al. *idtracker.ai: tracking all individuals in small or large collectives of unmarked animals*, Nature Methods volume 16, 2019, p. 179–182
[34] Zhenchao Ouyang, et al., *A cGANs-Based Scene Reconstruction Model Using Lidar Point Cloud*, 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)
[35] Kai Zhang, et al., *Learning Deep CNN Denoiser Prior for Image Restoration*, Conference: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
[36] Ayşegül Uçar, et al. *Object recognition and detection with deep learning for autonomous driving applications*, Sage Journals vol 93, issue 9, 2017, p. 759-769
[37] Shengze Cai, et al. *Dense motion estimation of particle images via a convolutional neural network*, Experiments in Fluids vol 60, article number 73, 2019.
[38] Purvil Bambharolia, *Overview of Convolutional Neural Networks*, International Conference on Academic Research in Engineering and Management, 2017.
[39] https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
[40] Delorme, A. & Makeig, S. *EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics*, Journal of Neuroscience Methods 134, 2004, p. 9–21.
[41] Mitra, P. & Bokil, H. *Observed Brain Dynamics.* Oxford University Press, New York. 2008.
[42] Rutishauser, U., et al. *Online detection and sorting of extracellularly recorded action potentials in human medial temporal lobe recordings*. Journal of Neuroscience Methods 154, 2006, p. 204–224.
[43] Spike2: *Life sciences data acquisition and analysis system*. Cambridge Electronic Design Limited (CED), Cambridge, England, http://ced.co.uk/downloads/latestsoftware , 2015.
[44] Volgushev, M.,et al. *Precise long-range synchronization of activity and silence in neocortical neurons during slow-wave sleep*. Journal of Neuroscience 26, 2006, p. 5665–5672.
[45] 20. Seamari, et al., *Robust Off- and Online Separation of Intracellularly Recorded Up and Down Cortical States*. PLoS ONE 2, e888, 2007.
[46] Ruiz-Mejias, et al. *Slow and fast rhythms generated in the cerebral cortex of the anesthetized mouse*. Journal of Neurophysiology 106, 2011, p. 2910–2921.
[47] Sanchez-Vives et al. *Inhibitory Modulation of Cortical Up States*. Journal of Neurophysiology 104, 2010, p. 1314–1324.
[48] Sanchez-Vives, M. V. et al. *Rhythmic spontaneous activity in the piriform cortex*. Cerebral Cortex 18, 2008, p. 1179–1192.
[49] Compte, A. et al. *Spontaneous high-frequency (10–80Hz) oscillations during up states in the cerebral cortex in vitro*. J. Neuroscience 28, 2008, p. 13828–13844.
[50] Mukovski, et al. *Detection of Active and Silent States in Neocortical Neurons from the Field Potential Signal during Slow-Wave Sleep*. Cereb. Cortex 17, 2007, p. 400–414.
[51] Ruiz-Mejias, et al. *Slow and fast rhythms generated in the cerebral cortex of the anesthetized mouse*. Journal of Neurophysiology 106, 2011, p. 2910–2921.
[52] Recio-Spinoso, et al. *Basilar membrane responses to broadband noise modeled using linear filters with rational transfer functions*. IEEE Trans. Biomed. Eng. 58, 2011, p. 1456–1464.
[53] van Drongelen, W., et al. *Propagation of seizure-like activity in a model of neocortex*. Journal Clinical Neurophysiology 24, 2007, p. 182–188.
[54] Jalil, M., et al. *Short-time energy, magnitude, zero crossing rate and autocorrelation measurement for discriminating voiced and unvoiced segments of speech signals.* Technological Advances In Electrical, Electronics And Computer Engineering, 2013, p. 208–212.
[55] McLachlan, et al. *Finite Mixture Models*. Wiley, New York, 2000.
[56] Figueiredo, M. & Jain, A. K. *Unsupervised learning of finite mixture models.* IEEE. Trans. Pattern Anal. Mach. Intelli. 24, 2002, p. 381–396.
[57] McLachlan, G. J. & Krishnan, T. *The EM Algorithm and Extensions.* Wiley, New York, 2008.
[58] Theodoridis, S. & Koutroumbas, K. *Pattern Recognition*. 3rd Edn, Elsevier, USA, 2006.

[59] Redmon, J. et al.. *You Only Look Once: Unified, Real-Time Object Detection*. IEEE Conference on Computer Vision and Pattern Recognition, 2016, p. 779–788.

[60] W. Liu, et al., *SSD: single shot multibox detector*, European Conference on Computer Vision. 2016, arXiv:1512.02325.

[61] Chambon, S., et al. *Domain adaptation with optimal transport improves EEG sleep stage classifiers*. International Workshop on Pattern Recognition in Neuroimaging. 2018.

[62] Chambon, S., et al. *A Deep Learning Architecture for Temporal Sleep Stage Classification Using Multivariate and Multimodal Time Series*. IEEE Trans. Neural Syst. Rehabil. Eng. 26 (4), 2018, p. 758–769.

[63] Chambon, S., et al. *A Deep Learning Architecture To Detect Events In EEG Signals During Sleep*. IEEE International Workshop on Machine Learning for Signal Processing. 2018.

[64] Ren, S., et al. *Towards real-time object detection with region proposal networks.* IEEE Transactions on Pattern Analysis and Machine Intelligence 39,, 2015, p. 91–99.

[65] Ioffe, S., et al. *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* in: Proceedings of the International Conference on Machine Learning, vol. 37, 2015, p. 448–456.

[66] Nair, V., Hinton, G.E.. *Rectified Linear Units Improve Restricted Boltzmann Machines*. in: Proceedings of the International Conference on Machine Learning ,2010, p. 807–814.

[67] https://towardsdatascience.com/intersection-over-union-iou-calculation-for-evaluating-an-image-segmentation-model-8b22e2e84686

[68] https://developer.ibm.com/recipes/tutorials/deep-learning/non-max-suppression/

[69] Mingcong Song, et al. *Bridging the Semantic Gaps of GPU Acceleration for Scale-out CNN-based Big Data Processing: Think Big, See Small*, Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, p. 315-326.