Detection of Abnormalities in Cardiac Rhythm Using Spiking Neural Networks

DORSA MOHAMMAD
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Detection of Abnormalities in Cardiac Rhythm Using Spiking Neural Networks

Dorsa Mohammad
dorsa.mohammad.3180@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Amir Aminifar, Saeed Bastani
Host company: Ericsson

Examiner: Christian Nyberg

March 31, 2023

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Artificial Intelligence (AI) and Machine Learning (ML) have been increasingly attracting attentions in many different fields. Healthcare is one of the areas that has greatly benefited from the advances in AI/ML. This includes a wide range of applications such as medical data interpretation, disease or abnormality detection or prediction, monitoring specific health condition and medical data management. On the other hand, patients can also take advantage of available healthcare devices to be more conscious of their health status and increase their quality of life. However, implementing AI/ML algorithms on resource-constrained wearable devices is challenging. One way to tackle this problem is to exploit the neuromorphic computing solutions such as Spiking Neural Networks (SNNs), which are more energy efficient than conventional neural networks because of their more similar function to how the brain works. In this thesis project, we investigate the working-mechanism of these networks, how we can design, train and use them for the detection of abnormalities in cardiac function.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors, Dr. Saeed Bastani from Ericsson Research and Dr. Amir Aminifar, for all their guidance, support and patience during all the ups and downs I experienced during this thesis journey.

Also, I was so blessed to have amazing international coordinators in my program at LTH to help me with all the practicalities to finish my Master's studies.

Last but not least, I am extremely grateful to have my parents as my best friends, supporters and companions in all peaks and valleys of my life. They have always been the ones giving me the strength to reach my goals and the ones strongly believing in me and reassuring me throughout my life. Thank you for being the most selfless, caring, understanding and encouraging parents I could wish for.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction and Background

Artificial Intelligence (AI) and Machine Learning (ML) are attracting more and more attention in the realm of science, industry, economy and etc. This can be because of their promising results in a wide range of applications: from self-driving cars, robotics and optimization for different problem settings, to protein design, neuroscience, medical image analysis and sensor-based applications in health sector. Furthermore, ML will be an essential element in the next-generation Internet of Things (IoT) systems, including mobile sensors and wearables. One example can be using data obtained from healthcare devices to detect a disease or a dangerous condition. However, healthcare wearable devices have limited computational and power resources. This makes the implementation of computationally heavy neural networks, in particular deep neural networks (DNNs), for these devices challenging [1], [2]. A workaround for this issue could be found by exploring neuromorphic computing. Neuromorphic computing is referred to as designing a system that is inspired by biological structures. The goal is to create a system that can learn, retain information and do certain tasks in a way that human brains can. Unlike conventional neural networks that have a trade-off between computation speed and power consumption, neuromorphic systems can achieve both fast computation and lower power consumption. This performance is achieved by building a Spiking Neural Network (SNN). SNNs can be used to reduce the power consumption but still provide promising results. Moreover, on the hardware side, several companies are working on designing chips that are optimized for computations being done in an SNN. These advances will contribute to the real-world implementation and evaluation of the neuromorphic models. As a result, this field seems to become more attractive in a few years [3]. We discuss all these concepts in more details in this chapter. In this thesis project, we aim to investigate the working-mechanism of SNNs and use them for heart-beat classification.

The remaining of this chapter provides the knowledge that is required to follow through the project.

1.1 Stakeholders

The host company of this thesis project is the Department of Device Software Research at Ericsson, Lund, Sweden. They proposed the initial research topic and provided the required resources for this project. Moreover, AI/ML and neuromor-

phic computing research community could benefit from the approaches and ideas proposed by this work.

1.2 Classical Machine Learning and Deep Learning

Learning is defined as the ability to improve the performance on future tasks after making observations about the world [4]. ML is a branch of AI that uses data and algorithms to learn a defined task and gradually improves the target performance metric such as accuracy. ML can mainly be classified into two types: supervised learning and unsupervised learning. However, some also add a third and fourth type: semi-supervised machine learning and reinforcement learning.

In supervised learning, there are a collection of labeled data being used to train algorithms to classify data or predict outcomes accurately. In other words, each data sample is a pair of input and output. In the training step, we provide these data samples to the model and the model should adjust its weights until it reaches an appropriate mapping from the inputs to the outputs. There exist certain criteria used in cross validation step, such as overfitting or underfitting, to decide what is appropriate. Then, if a new input is fed into the model, it can predict the output. Neural networks, linear regression, logistic regression, support vector machine (SVM) and random forest are examples of supervised learning methods.

Unsupervised learning uses some ML algorithms to find patterns hidden in unlabeled datasets and the human supervision is not required. For instance, it is a good choice for exploratory data analysis because of its ability to discover similarities and differences in data samples. Unsupervised learning is also used to reduce the number of features (which is the dimension of data samples) in a model. Principal component analysis (PCA), singular value decomposition (SVD) and k-means clustering are a few of well-known algorithms used in unsupervised learning [5].

Semi-supervised learning is an approach that lies between supervised and unsupervised learning as it can be inferred from its name. This algorithm is provided with a labeled data set and usually a larger unlabeled data set. An example can be a supervised learning problem setting that aims to predict a target value for a given input while having some additional information on the distribution of the examples (unlabeled data set can be used here)[6].

Reinforcement learning is learning how to map situations to actions in order to maximize a numerical reward signal. The agent (or learner) itself should find out, in each situation, which actions lead to the most reward by trying them. Besides affecting the amount of immediately earned reward (or possibly punishment), actions can also change the next situation and, through that, all subsequent rewards. These two characteristics, experimenting the states by trial-and-error search and delayed reward, are the two key distinguishing features of reinforcement learning [7].

Artificial Neural Networks (ANNs) are a subset of machine learning algorithms. These algorithms are inspired by the way biological neurons signal to each other in the human brain. An ANN consists of an input layer, one or more hidden

layers, and an output layer. Each layer has some artificial neurons. Each neuron can connect to another and has an associated weight and threshold. If the output of a neuron is greater than its threshold value, that neuron passes data to the next layer of the network, otherwise, the neuron would be inactive [8]. The function that is defined for each neuron to enforce this active-inactive behavior is called the activation function. More detailed mathematical model of an artificial neuron is shown in Figure 1.1. There exist different architectures of ANNs, the most famous of which are Multi-layer Perceptron (MLP), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). Moreover, the training procedure of the neural networks are mostly based on backpropagation algorithm. Neural Networks with more than one hidden layer are called deep neural networks (DNNs) and considered as DL algorithm. The typical structure of DNNs is depicted in Figure 1.2.

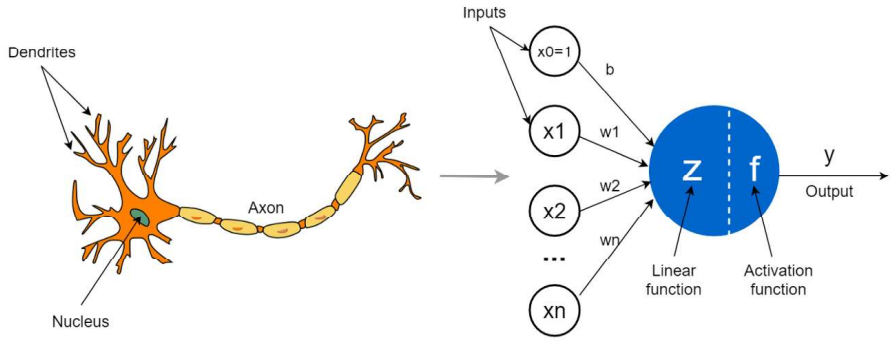


Figure 1.1: The biological and the artificial neuron [9].

1.3 The Role of AI/ML in Healthcare and its Challenges

Personalized healthcare is a concept that can considerably affect the patients' lives by customizing required care to the individual and providing an effective care that leads to the faster outcomes and/or preventative measures. This concept can play an active role through the whole journey of the patient, including the prevention, diagnosis, treatment and monitoring of disease. The digital revolution in healthcare has helped to collect, connect and analyze data from large population of patients. This is one of the key factors to be able to use methods that could provide healthier individuals [10].

The proliferation of mobile and wearable devices has resulted in the possibility of collecting large amount of physiological, biological and behavioral data about people's daily life [11]. AI and ML methods can take advantage of these data to predict or detect a specific health status. This has the potential to improve people's quality of life. Some examples of physiological data that various wearable devices are capable of collecting are heart rate, electrocardiogram (ECG),

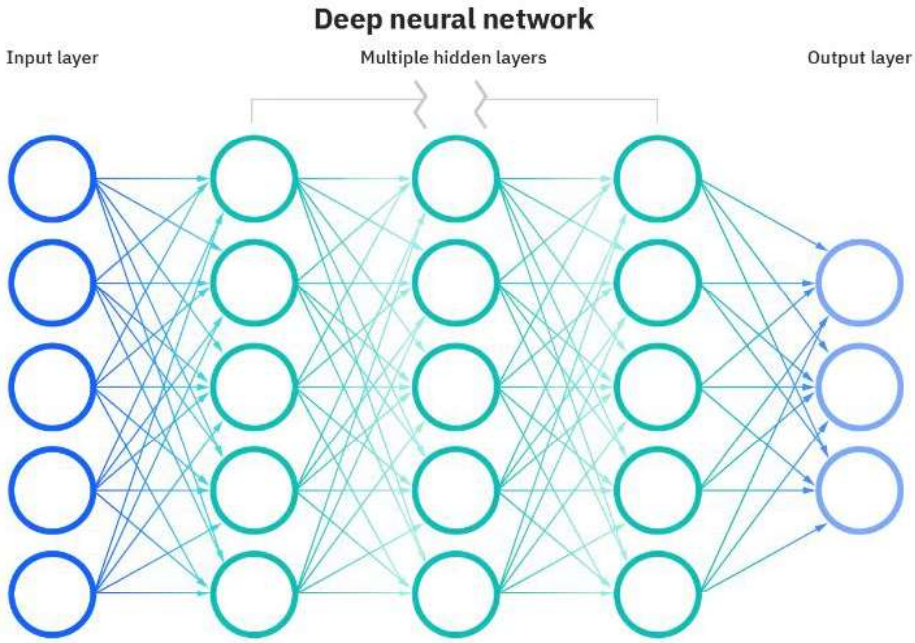


Figure 1.2: The typical structure of DNNs.

electroencephalogram (EEG), blood glucose, blood pressure, blood oxygen saturation, respiration rate, body temperature and photoplethysmography (PPG). These data can be used in several kinds of healthcare applications [10]:

a) Preventive health: Here, the goal is to detect a health related abnormality. An insightful example can be smartwatches that provide the obtained heart rate, ECG and blood pressure to an ML-based platform and notify the users about the potential conditions such as blocked arteries. It can act as an alert for patients to follow up and take care of the cardiac problems [1] [12] [13] [14] [15]. Therefore, a dangerous situation such as heart stroke could be prevented.

b) Medical consultation: This acts as an AI doctor that collects a large amount of medical knowledge from available medical databases. It can save the time for the diagnosis and help the real doctors to focus on finding treatment for the disease [16], [17].

c) Medication management: This includes collecting and managing physiological data, producing health reports and exploiting helpful information enabling wearable devices to recommend certain kinds of prescriptions or to remind the patients about the precise time they should take the drugs (based on the device continuous monitoring) [18].

d) Monitoring chronic disease conditions: Wearable devices can be used to keep track of a specific health metric whose changes and fluctuations have an important role in a chronic disease and should be taken care of constantly. For instance, using these devices for a diabetic patient can monitor oscillations in the blood sugar level and helps to provide more precise insulin treatment customized

for the patient and optimize the diabetic control. Other examples include sleep related issues (such as Apnea) and epilepsy [19], [20], [21], [2], [22], [23], [24], [25].

e) Monitoring stress: The heart rate variability information collected by wearable device can be used to cast light on people's mental well-being and body stress [26], [27]. Also, another physiological data that can be applied for stress management is respiration rate.

The work we have done in this project has the nature of the first and third group.

Unlike the data collected in clinic settings, the amount of data collected by wearable devices is much larger and also noisier because of irregular body movements. Signal processing techniques and ML-based techniques are two methods used to work with these data. These two techniques are usually applied together to improve the model performance. ML based approaches commonly should go through three main steps: **1) preparation:** splitting raw streaming data into discrete pieces for further processing, **2) feature extraction:** using methods to obtain effective feature representations from data, **3) decision:** training the ML model using the derived features to output a decision for testing data. Note that we can integrate steps 2 and 3 and make the learning process more end-to-end by using DL methods. Some popular AI/ML techniques and their use-cases in healthcare are outlined in Figure 1.3 [10].

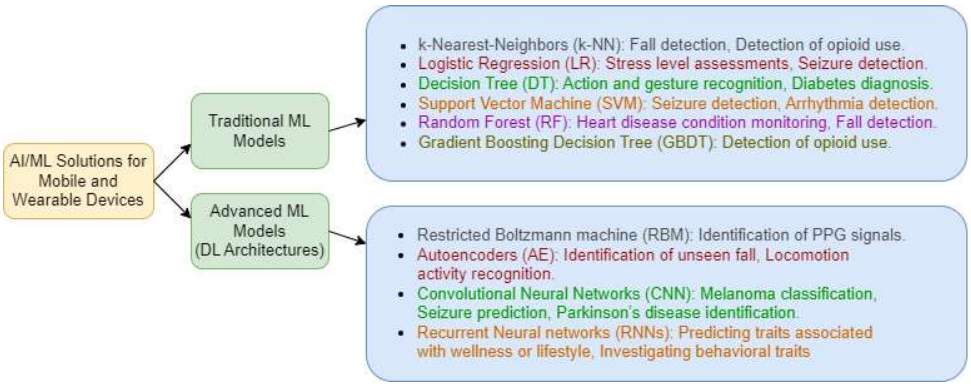


Figure 1.3: Some popular AI/ML methods used for mobile and wearable devices and their healthcare use-cases.

When it comes to the challenges of applying AI/ML methods for wearable devices, one of them is that most wearable devices have a miniaturized design and thus they have very limited energy supply and storage. However, the performance of most AI/ML approaches rely on huge energy and computational resources. Thus, taking energy efficiency into account and developing novel AI/ML techniques that requires less energy resources can be very beneficial.

1.4 Event-Driven Machine Learning

In event-driven computing, the happening of a specific action depends on the occurrence of predefined trigger events. This can help to lower the requirement of powerful computational resources, by performing the most computationally expensive processing only when a particular trigger event occurs. This is also more energy efficient and can improve the battery lifetime of wearable devices [28]. An example can be [28] that present an event-driven classification technique for Myocardial Infarction on wearable devices. One of the goals of this paper is to introduce a classification method being less computationally complex and more energy efficient, while obtaining a high accuracy result. The idea is to reduce the number of features that is required to compute for making confident decisions (accurate enough), and as a result reduce the energy consumption. This is done by using a hierarchical classifiers that can decide when it needs to go deeper in classification level. A kind of neural networks that has an approach similar to event-driven methods is called Spiking Neural Networks (SNNs) which is introduced in the next section.

1.5 The Biology of Neurons [29]

The neurons are responsible to generate electrical signals in response to chemical and other inputs, transmit them to other cells. The typical structure of a neuron is demonstrated in Figure 1.1. The neuron is made of dendrites that receive inputs from other neurons and the axon that conveys the neuronal output to other cells. Because of the branching structure of the dendritic tree, the neuron is able to receive inputs from many other neurons through synaptic connections. Also, the end of an axon where it is connected to another neuron is called synapse.

It is useful to explain briefly about different types of biological synapses, since it can give us insight to mathematically model a brain inspired neural network later on. The types of most synapses in the brain are excitatory and inhibitory. The process of learning creates new synapses between neurons in the brain and cut off old connections. Signals transmitted over excitatory synapses increase the activity of the receiving neuron, while signals transmitted over inhibitory synapses reduce neuron activity. The balance between excitation and inhibition is a key factor for the brain to work properly [30].

The mechanism of generating signals is based on the presence of various membrane-spanning ion channels that allow ions, mainly sodium (Na^+), potassium (K^+), calcium (Ca^{2+}) and chloride (Cl^-), to move into and out of the cell. Ion channels control the flow of ions across the cell membrane by opening and closing in response to voltage changes and to both internal and external signals. In the nervous system, the electrical potential difference between the interior of a neuron and the surrounding extracellular medium is defined as the electrical signal. In resting conditions, the potential inside the neuron membrane is about -70 mV relative to that of the surrounding medium. This is called the polarized condition. The process of flowing positively charged ions out of the cell is called hyperpolarization. This makes the membrane potential more negative. The reverse process, which means (positive) current flows into the cell changing the membrane

potential to less negative or even positive values, is called depolarization.

An action potential, which is an approximately 100 mV fluctuation in the electrical potential across the cell membrane lasting for about 1 ms, is generated when a neuron is depolarized adequately so that the membrane potential reach above a threshold level, initiating a positive feedback process. For a few milliseconds just after an action potential, another spike cannot be fired. This is called the refractory period. Figure 1.4 shows the membrane potential before, during and after an action potential. Also, the ion movements between the extracellular space and the neuron cytoplasm in different conditions are demonstrated in Figure 1.5.

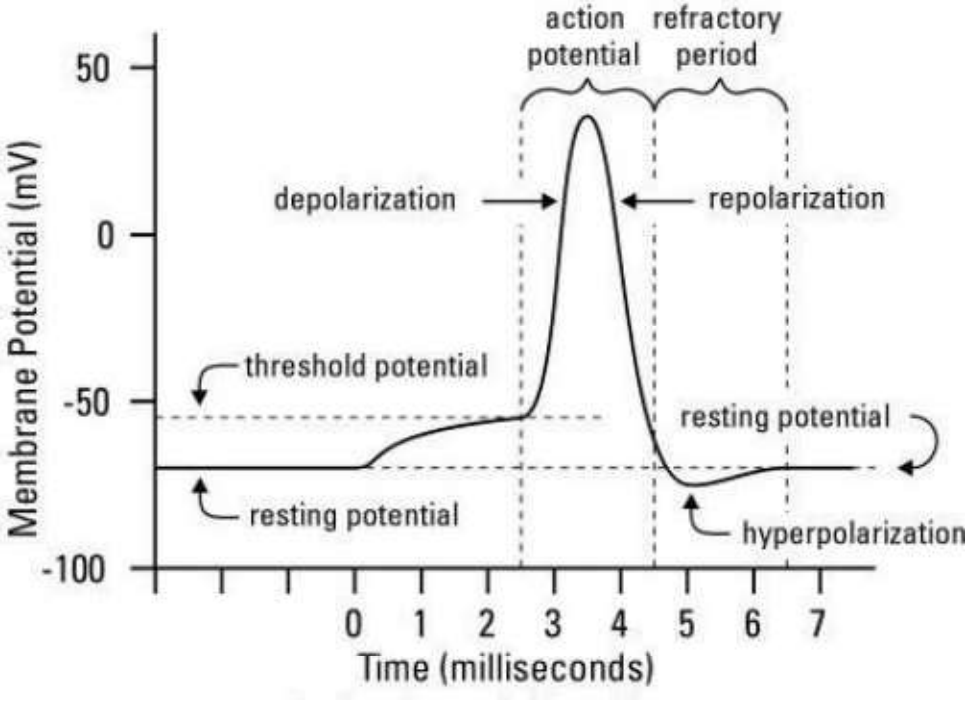


Figure 1.4: Diagram for action potential [31].

Action potentials transmit information through their timing. By neglecting the small duration of an action potential, we can represent an action potential sequence by a list of the times when spikes occurred (such as $t_1, t_2, ..., t_n$ for n spikes, where $0 \leq t_i \leq T$ for all i and 0 and T are the start and end time of recording the spikes). Moreover, the spike sequence can be shown as the sum of Dirac δ functions as the following:

$$\rho(t) = \sum_{i=1}^{n} \delta(t - t_i) \tag{1.1}$$

$\rho(t)$ is called the neural response function. A neuron can respond differently from trial to trial even when the same stimulus is enforced. Because of this behavior, we cannot describe and predict the timing of each spike in the action potential

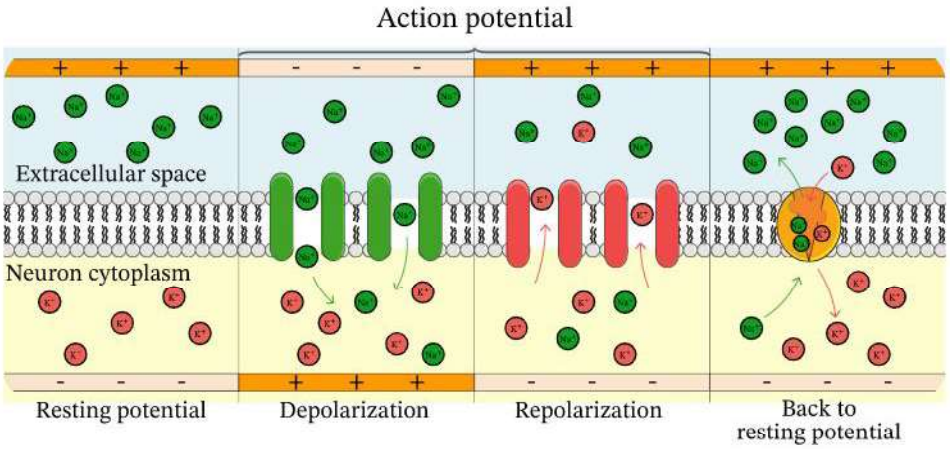


Figure 1.5: The ion movements between the extracellular space and the neuron cytoplasm in each stage of an action potential [32].

sequences deterministically (because there exist some randomness). Therefore, the neuronal responses are usually studied statistically or probabilistically. An example is the firing rates that can be used to characterize the neuronal responses instead of specific spike sequences. Now, firing rate would be defined step by step. First, we should introduce spike-count rate (r) which is the number of action potentials that appear during a trial (n) and divided by the trial duration (T):

$$r = \frac{n}{T} = \frac{1}{T}\int_0^T \rho(\tau)d_\tau \tag{1.2}$$

This can be interpreted as the time average of the neural response function over the trial duration. The spike-count rate r is not a function of time and can be computed from a single trial. A time-dependent firing rate can be defined as the number of spikes over short time intervals. However, in this case we cannot determine the value from a single trial, and we should average over different trials. Then, we can compute the time-dependent firing rate by taking the average number of spikes (averaged over trials) existing during a short interval (between times t and $t + \Delta t$, and dividing by the interval duration (Δt). From Equation 1.2 we can state that the number of spikes between times t and $t + \Delta t$ in a single trial is equal to $\int_t^{t+\Delta t} \rho(\tau)d_\tau$. Now, the average number (through N different trials) of spikes during this interval is equal to $\frac{1}{N}\sum_{n=1}^{N}\int_t^{t+\Delta t} \rho_n(\tau)d_\tau$, where $\rho_n(\tau)$ represents the neural response function of trial n. This can be equivalently written as $\int_t^{t+\Delta t}\langle\rho(\tau)\rangle d_\tau$, where $\langle\rho(\tau)\rangle$ is the trial-averaged neural response function. Finally, the firing rate (which means time-dependent firing rate $r(t)$) can be written as the following:

$$r(t) = \frac{1}{\Delta t}\int_t^{t+\Delta t}\langle\rho(\tau)\rangle d_\tau \tag{1.3}$$

For computing the value of $r(t)$ from data, Δt should be sufficiently large.

1.6 Artificial vs. Biological Neural Networks

The nature of the transmitted information: As we described in the last section, the working mechanism of biological neurons is based on spike generation. This implies that the information is encoded and transmitted in the form of discrete events called spikes and the important thing is the timing of their occurrence. On the other hand, artificial neurons in conventional neural networks can send continuous values in form of synaptic weights, that makes the training procedures such as backpropagation possible. The limit on the range of these values depends on the used activation function in the model. For instance if the Sigmoid function is used, the neuron can fire all the time, but with the different signal amplitudes.

Speed: It is clear that the artificial neurons, unlike their biological counterparts, never get fatigued meaning that they are functions being computed as many times and as fast as the computer hardware is capable of and that is the only limiting factor. Also, another contributing factor for speed is that there is no refractory period.[33]. Furthermore, the speed of the computations required for training an ANN, which can usually be reduced to multiple matrix operations and finding derivatives and calculating them for a large amount of data points over and over again, increases by using specific hardware for AI or graphics processing units (GPUs) [33].

Power consumption: As mentioned before, ANNs are much less energy efficient than the human brains. To have a reference, the thermal design power (TDP), that is defined as the power consumption under the maximum theoretical load, of NVIDIA GeForce RTX 3090, which is one of the most powerful available GPUs for DL applications at the time of writing this thesis, is 350 watts [34], while an adult human brain operates on about 20 watts [48]. Another issue is that computers generate a lot of heat when used, but humans operate in 36.5–37.5 °C [33].

1.7 Spiking Neural Networks

As a quick recap on feed-forward ANNs, we can briefly recall that they are multiple layers of activation function nodes connected successively by weight parameters. It is a mathematical model that is trained by numerous examples to learn the weight matrix to be able to map the input x to the output and form a function $f(x; w)$ and the widely used algorithm to find the weight parameters is backpropagation. Here, we should note that the model does not depend on time since no time-dependent element is incorporated to the function f [35].

Now, it is time to introduce SNNs. They could be considered as directed graphs consist of nodes (as spiking neurons) and directed edges (as synapses). The main difference compared to ANNs is that the spiking neuron does not act like the activation function node in ANNs. They incorporate a time-dependent state variable, such as membrane potential that is a function of presynaptic spikes. Also, unlike the real-value nature of inputs in the ANNs, the input information to the spiking neurons is in the form of a stream of binary numbers through time (existence or absence of a spike). In fact, the membrane potential is affected by

arrival of any spikes. The mechanism of a leaky integrate-and-fire (LIF) model is demonstrated in Figure 1.6. The membrane potential u changes following of input spikes and the neuron fires a spike when the membrane potential reaches a threshold u_{th}. Just after the firing, the membrane potential resets to the ground value and it is followed by the refractory period. The fact that the neuron's state progress through time, represents a memory-like effect [35].

It is worth mentioning that the existing neural networks for sequence learning such as recurrent neural networks (RNNs) and long short-term memory (LSTM) do not consider the time interval between neighboring events and just the concept of sequence is important to them. In other words, they do not incorporate physical time into their model. Therefore, different learning algorithms should be used for SNNs compared to ANNs [35]. In this section we introduce a few of them. We also discuss neuron models and coding schemes in SNNs.

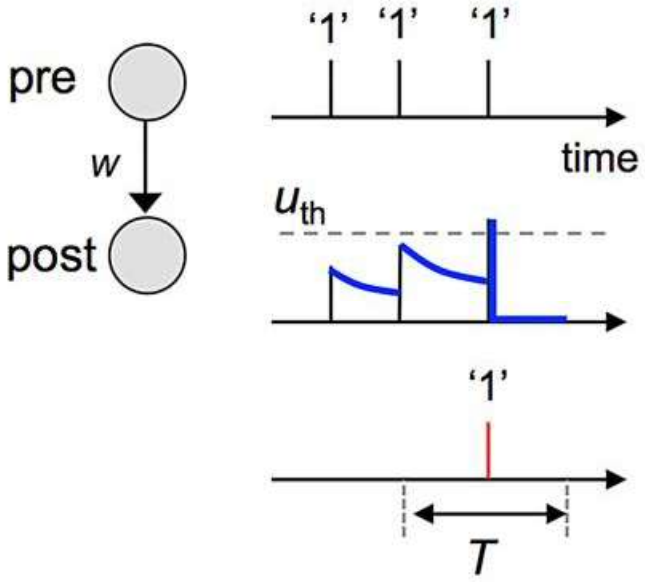


Figure 1.6: A model of spike generation. [35].

1.7.1 Neuron Models

There exist various spiking neuron models. One of the state variables used to model spiking neurons is membrane potential that we introduced earlier for LIF neuron (or Stein's model). However, to model the neuron in more details and make it more similar to the actual biological counterpart, the model would be more complex and more than one state variable would be required. Some of the proposed neuron models are Zhikevich's model [36], FitzHugh-Nagumo (FHN) model [37], [38]. Here, we look into the neuron models involving one state variable in more details.

One of the simple models is the one that involves a single state variable (which is membrane potential). An example of such a model is demonstrated in Figure 1.6. As mentioned earlier, the membrane potential upsurges when an input spike arrives and then decreases during the inter-spike interval (ISI). The membrane potential $u(t)$ can be written as the following [35]:

$$u(t) = u_0 + a \int_0^t D(s) \cdot w \cdot \sigma(t-s) d_s \tag{1.4}$$

Here, u_0, a, $D(s)$ and w are the base membrane potential, positive constant, a linear filter (kernel) and the synaptic weight, respectively. Also, σ is a sequence of N input spikes (i^{th} spike at time t_i) and it can be written as $\sigma(t) = \sum_{i=1}^{N} \delta(t-t_i)$. Two examples with two different kernels are shown in Figure 1.7. The first one, using a constant window kernel, is called integrate-and-fire (IF) model and the second one, using an exponentially decaying kernel, is called LIF or Stein's model as mentioned before.

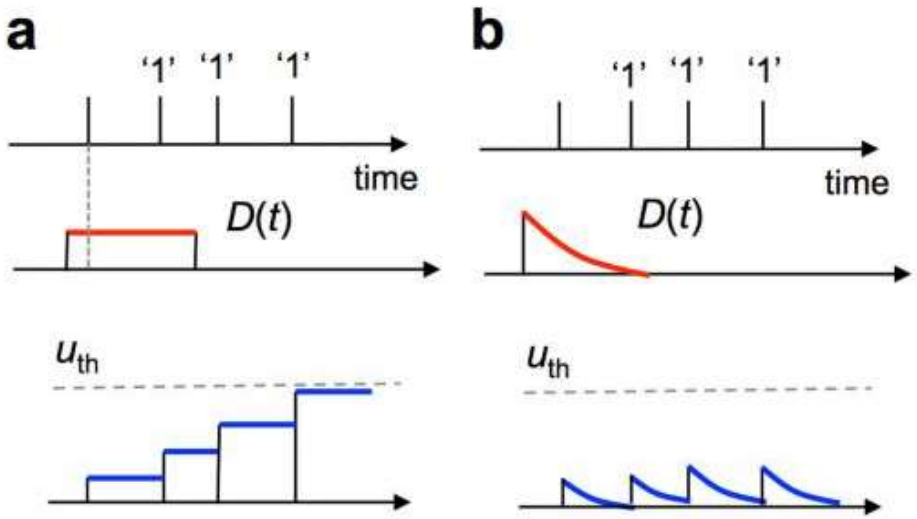


Figure 1.7: Membrane potential changes because of the arrival of spikes using a constant window kernel and an exponentially decaying kernel. [35]

1.7.2 Coding Schemes

In ANNs, an activation function encodes the real-valued sum of weighted inputs to a real-valued output. However, neural coding is not that simple in SNNs. Three coding schemes used in SNNs are introduced in this section [35]. Figure 1.8 illustrates the mechanism of these encoding methods. Figure 1.8(a) and (b) show the spike generation of input neuron (sensory neuron) in response to 3 different values of input stimuli. The other ones depict the postsynaptic spikes in response

to a presynaptic spike train for 3 introduced methods and two different weight values ($w_1 < w_2$).

Spike-count code: In the model of Figure 1.7(a), the postsynaptic neuron potential increases in proportion to the number of input spikes until firing a spike. Because of the fact that the leakage is not modeled here, how fast the input spikes arrive at the postsynaptic neuron (input firing rate) and when they arrive (input spike-timing) do not have any impact on the neuronal response. In other words, for a given synaptic weight, the number of input spikes is the determining factor for generating the postsynaptic spikes. A schematic of this encoding scheme is shown in Figure 1.8(c). It can be noticed that the postsynaptic neuron with weight w_1 fires a spike when it receives 3 presynaptic spikes, while the postsynaptic neuron with weight $w_2 > w_1$ fires a spike when it receives 2 presynaptic spikes. Furthermore, Figure 1.8(a) illustrates spike-count code for a sensory neuron.

Rate code: Here, the potential leakage is considered. Therefore, the presynaptic spiking rate determines the firing of postsynaptic spikes. The input spiking rate should be high enough (or in other words, the presynaptic spikes should be close together enough) to make the postsynaptic neuron reach the threshold and fire a spike. Note that the firing rate is defined as the average number of spikes per unit time. Figures 1.8(a) and (d) show examples of postsynaptic spike train in response to sensory inputs and a presynaptic spike train, respectively. In this method, the neuron should integrate input spikes long enough to determine the average of spike number per unit time. This makes rate coding time consuming, which is one of the drawbacks of this scheme.

Temporal code: In this method, the neural information is incorporated into individual spikes rather than their temporal average. This can perform faster than rate coding. An example of temporal coding is latency coding, which is depicted in Figure 1.8(b) for a sensory neuron. We can observe that the strength of the input value intensity controls the timing of the generated postsynaptic spike. The stronger the input the sooner a spike is elicited (the lower the spiking latency). Also, Figure 1.8(e) shows an example of temporal code in a pair of presynaptic and postsynaptic neuron. We can notice that a single spike (or more precisely, its spike timing) is enough to carry the input information. A disadvantage of temporal code is being vulnerable to error in the presence of the variability in ISI.

1.7.3 Training Techniques for SNNs

The mostly used methods for training typical ANNs are gradiant-based. Backpropagation is an algorithm applied in these methods that uses the chain rule from the last layer back to each weight to derive the gradient of the loss function with respect to each learnable parameter (weight) [39]. Because of the non-differentiability of spikes, there exist a problem to train SNNs by using backpropagation. This is also known as **dead neuron** problem [40]. In fact, the gradient of the loss function with respect to wights is either 0 or ∞, and thus, the weight update cannot be performed by using backpropagation.

There exist two main approaches to handle the dead neuron problem and be able to train SNNs. Here, we discuss them:

Shadow Training or Conversion Method: In this method, a DNN is

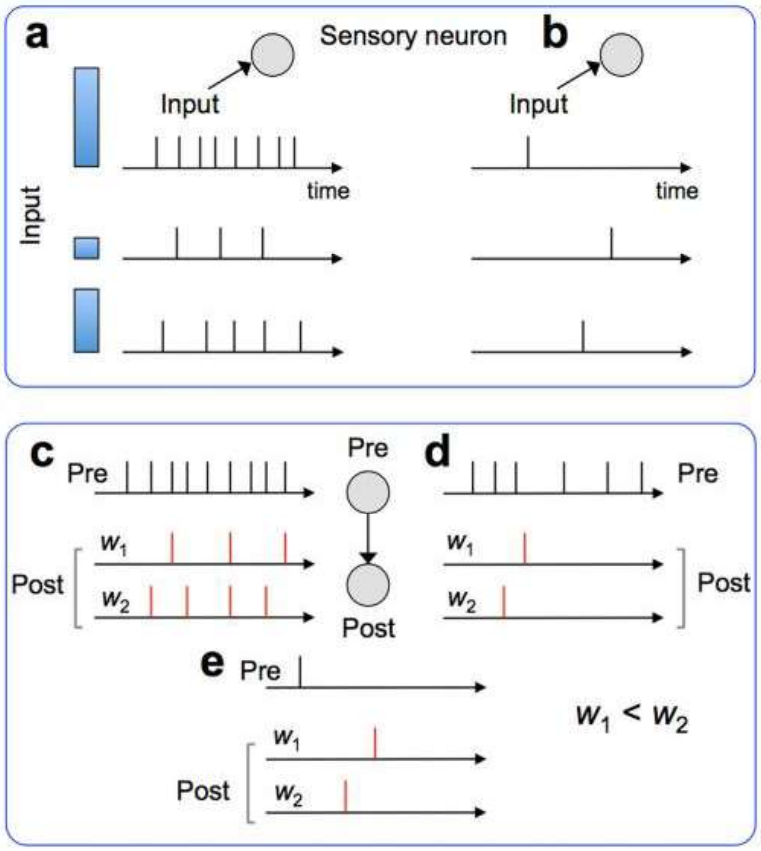


Figure 1.8: Illustration of the working mechanism of different coding scheme. (a) spike-count and rate coding for sensory (input) neurons. (b) latency coding for sensory neurons. (c) spike-count code, (d) rate code and (d) latency code in presynaptic-postsynaptic neuron setting at two different weight values [35].

trained using backpropagation, and then, all neurons should be converted into spiking neurons [41]. One of the reasons that can be convincing to use this technique in some applications is that the state-of-the-art algorithms developed for conventional DNNs can also be applied here. Also, this is a reasonable choice for applications where inference performance is more important than training efficiency [40].

In this project, we employ this method by replacing the ReLU neurons in our CNN with integrate and fire (IF) spiking neuron to have our SNN. We do this using NengoDL Python package. Figure 1.9 illustrates how IF spiking neuron functions compared to a ReLU neuron.

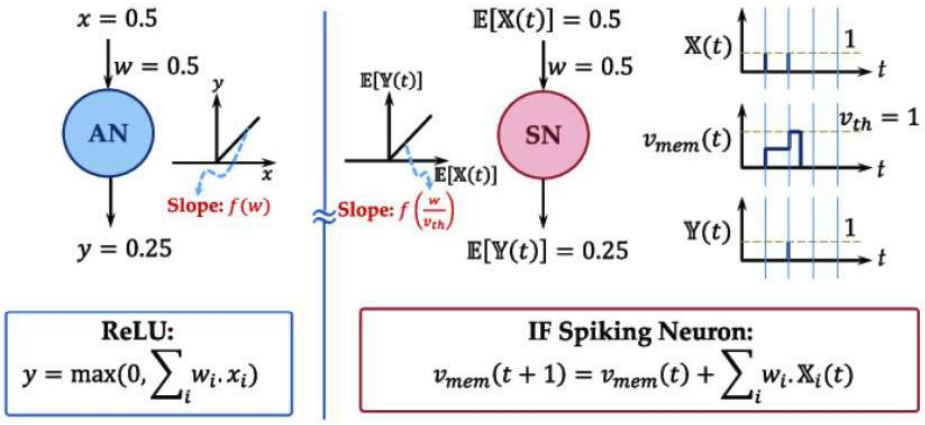


Figure 1.9: Comparison of artificial neuron with ReLU function and IF spiking neuron [42].

Direct training of SNNs: Another approach is to constitute other learning rules that does not require the use of original version of backbropagation for updating the weight parameters. Therefore, in this way, the SNN would be directly trained and built without involving any other ANN. Bohte *et al.* [43] proposed one of the first methods, called SpikeProp, to train multi-layer SNNs using spike times for backpropagation. In this method, the gradient of loss with respect to the spike **timing** (as opposed to spike itself, which is not continuous) is calculated [40]. The intuition behind how this works is as following: if the weight w changes by Δw, then the membrane potential U changes by ΔU. This leads to a change in spike timing f by Δf. This is shown in Figure 1.10 [40]. Another solution is to use variants of backpropagation through time (BPTT) algorithm, which is discussed in several papers such as [44] and [45]. Moreover, local learning rules at synapses, such as spike-timing-dependent plasticity STDP [46] similar to Hebbian learning [47], is another choice that is more biologically inspired [41]. We refer to a research using this method in the next chapter.

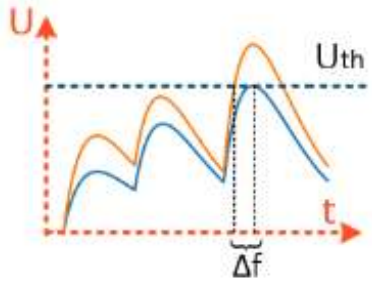


Figure 1.10: The way the spike-time gradient can work [40].

1.8 Neuro-Inspired Computing Processors

The real world applications of AI, requiring the processing of large amount of data, has been limited by the computing speed and power efficiency of the hardware. Therefore, the typical computing hardware based on a von Neumann architecture is not an efficient choice for dealing with AI tasks. Especially, the power-constrained applications of AI, for example in the fields of edge computing and the Internet of Things (IoT), require the development of new chip architectures. Neuro-inspired computing chips, imitating the structure and principles of the biological brain, are more energy efficient in performing AI tasks [48]. It is insightful to mention that the human brain has more than 10^{11} neurons and 10^{15} synapses and consumes only 20 watts (as said before) that is much more efficient than conventional von Neumann computers for recognition and decision-making tasks [48].

Dedicated chips designed for ANNs and SNNs are two types of neuro-inspired computing chips that share some features similar to those of the brain, such as a neuron-synapse structure, in-memory computation and learning capabilities, while working with either ANN or SNN algorithms. The neuron states in ANN chips, are encoded as digital bits, clock cycles or voltage levels. However, in SNN chips, information is encoded into spike timing. Also, some ANN chips try to enhance energy efficiency by computing weighted-sum computations in memory, and SNN chips are designed to reach ultra-low power consumption and run at relatively low frequencies. However, most available SNN chips today, are just able to perform basic AI tasks [48].

During the last decade, several neuromorphic processors have been released. Some prototypical examples being able to realize SNNs on the chips are Neurogrid [50], TrueNorth [51], DYNAPs [52], SpiNNaker [53] and Loihi [54] [35]. Neuromorphic hardware accompanied by a user-friendly compiler, for example by designing a graphical user interface (GUI), can be beneficial for easier implementation. An example of a GUI-based tools is Nengo that can be used to build SNN models, compile and map them on neuromorphic hardware [35].

1.9 Useful Simulation Tools for SNN Implementation

There exist multiple python libraries for building and testing SNNs. Nengo [55] [56], Brian2 [57], snnTorch [58], BindsNET [59], SpykeTorch [60] and ANNarchy [61] are example tools of this kind. None of these libraries are written in Python but they still provide us with an interface in Python. Some of them are also capable of acting as an interface to realize the SNN on a hardware. For example, Nengo has successfully been applied to Loihi and Braindrop chip [35]. For the main part of this project, we decided to use Nengo, because it provides comprehensive documentation, useful examples, more online resources, continued developmental support and updates. we have also explored Brian2 to understand the limits of this tool, as it will be discussed in the next chapter.

The Nengo Brain Maker (simply called Nengo) is a Python package for building, testing, and deploying neural networks. The summary of this ecosystem is depicted in Figure 1.11. The Python library Nengo (which is the core of the Nengo ecosystem), contains five Nengo objects (Ensemble, Node, Connection, Probe, Network) and a NumPy-based simulator. NengoDL simulates Nengo models (meaning that it gets a Nengo network as input) using the TensorFlow library. This makes the interaction with DL networks easier. It is also capable of using DL training methods to optimize Nengo model parameters.[55]

Besides the applications in AI/ML, Nengo could be also used for modeling the human brain which is a very challenging task. Nengo is based on a theoretical framework proposed by Eliasmith and Anderson [62] called the Neural Engineering Framework (NEF). The NEF had been used to build the world's then largest functional brain model called Spaun [63], which is a network consisting of 2.5 million spiking neurons that can perform eight cognitive tasks [56]. This model has resulted in a more advanced vesion Spaun 2.0 that has approximately 6.6 million neurons and can perform 12 cognitive tasks [64].

Brian2 is another simulator of spiking neural network models. Here, the models are mostly defined as dynamical equations. Users can write code with simple and brief high-level descriptions, and Brian converts them into efficient low-level code [65].

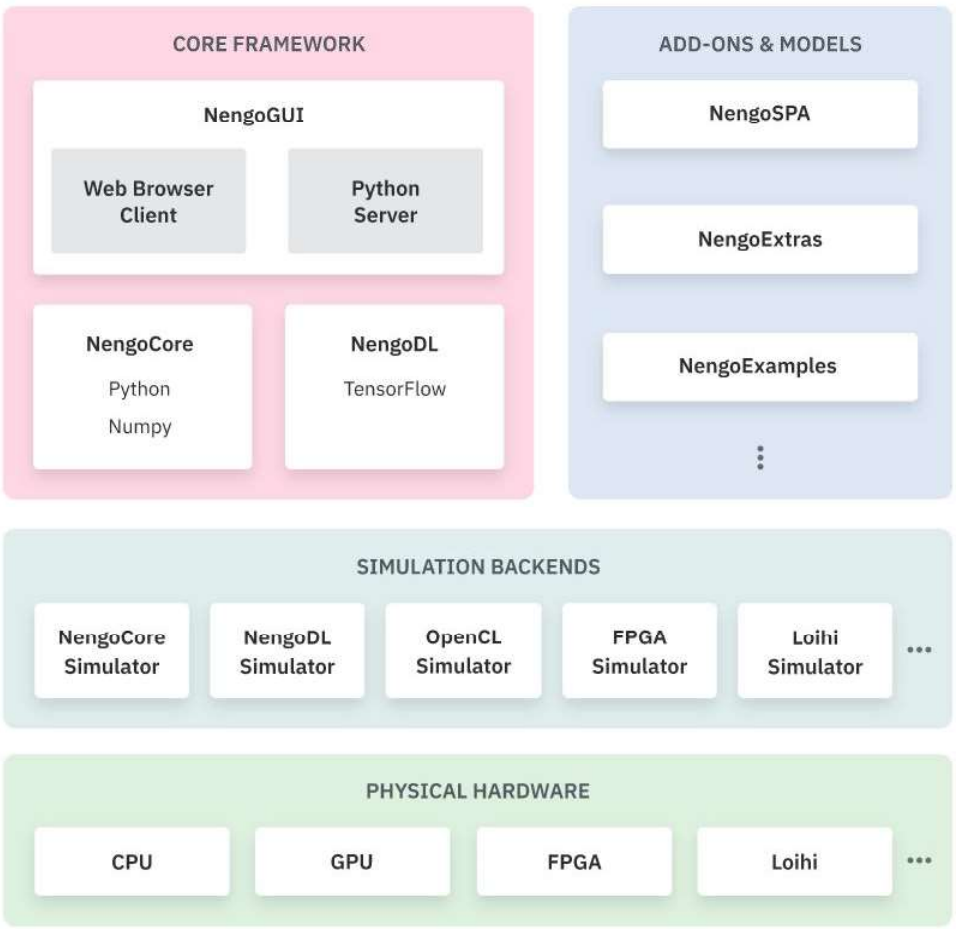


Figure 1.11: An overview of the Nengo ecosystem including several interacting projects. [55].

Chapter 2
Related Work

In this chapter, we review the state of the art work in relation to this thesis. This is organized into three sections. In the first section, we describe a publicly available database containing large amount of ECG signals. Then, several previous research work using ECG data are described and finally, we explain a specific research paper that we investigated in more details throughout this thesis project.

2.1 ECG Data Description

Electrocardiograms (ECGs) are widely used as an inexpensive, noninvasive and simple ways to evaluate the heart. Electrodes, being connected to an ECG machine, are placed at certain spots on the body. Then, the electrical activity of the heart is measured and interpreted. Continuous recording of the ECG in ambulatory subjects over many hours are called long-term ECG. This is a standard way for observing transient aspects of cardiac electrical activity [66].

MIT-BIH Arrhythmia Database [67] has been widely used for ECG classification which was the main work in this project. This database is publicly available on PhysioNet website [68], [69]. It contains 48 half-hour excerpts of two-channel ambulatory ECG recordings, which are related to 47 people (two records are from the same person). One channel is obtained from a modified limb lead II (MLII), that is when the electrodes are placed on the chest. The other channel is usually V1 and sometimes V2, V4 or V5. These spots can be seen in Figure 2.1 [70].

These excerpts were selected to include examples of uncommon but clinically important arrhythmias that would not be well represented in a small random samples. The digitization rate is 360 samples per second per channel. There are approximately 110,000 beats and each beat is assigned to a specific annotation by cardiologists [67]. Figure 2.2 shows 10 seconds from a recording of the MIT-BIH Arrhythmia Database [67].

The beat annotations that has been used in this database are shown in Table 2.1. In this project, all beats are grouped into 5 classes inspired by AAMI EC57 categories. Because, according to the ANSI/AAMI/ISO EC57:1998/(R)2008 (American National Standard on testing and reporting performance results of cardiac rhythm and ST-segment measurement algorithms) [71], only these five classes are sufficient for arrhythmia detection, which can be seen as accurate classification of heartbeat classes. These classes are also depicted in Table 2.1.

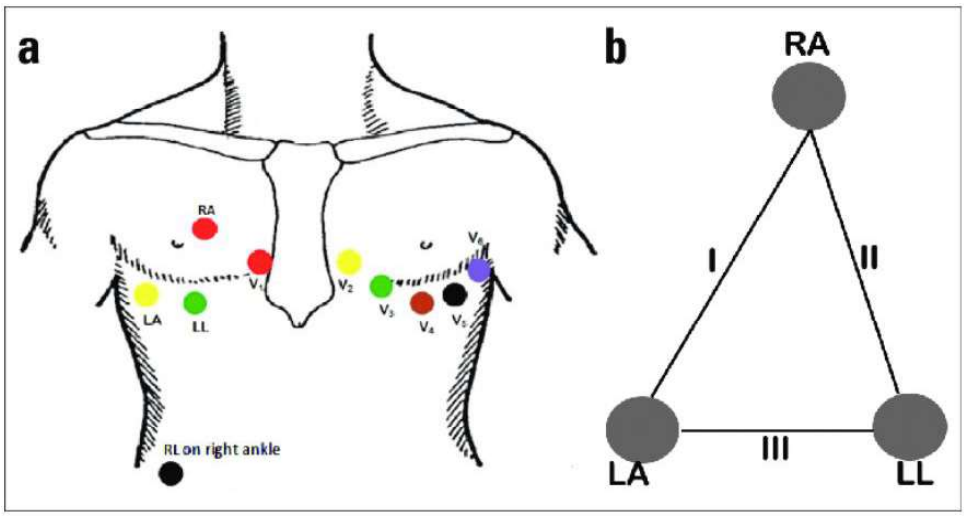


Figure 2.1: Different positions for electrodes [70].

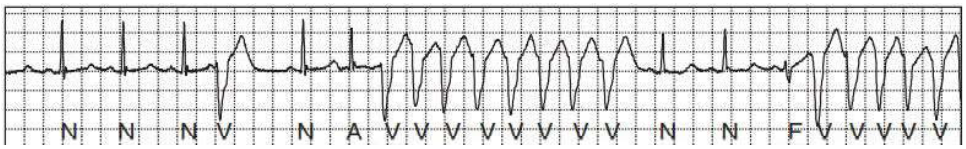


Figure 2.2: 10 seconds from a recording of the MIT-BIH Arrhythmia Database and the corresponding beat annotations [67].

Here, we briefly describe the structure of a heartbeat. The normal rhythm of the heart is called sinus rhythm. There exist 3 distinct waves (named with different labels) on ECG that are representatives of depolarization and repolarization of the atria and ventricles. The first wave is P wave representing atrial depolarisation. After the first wave there exists a short flat period allowing the atria enough time to pump all the blood into the ventricles. Then, it is followed by a complex of 3 waves known as the QRS complex. The largest one, R wave, represents ventricular depolarization. In other words, this wave denotes the electrical stimulus passing through the main portion of the ventricular walls. It is clearly the biggest wave existing in a normal heartbeat signal. The reason is that the wall of the ventricles is very thick and more voltage is required. The last wave in a whole cycle is T wave illustrating the ventricular repolarization. All these different waves in a heartbeat signal are shown in Figure 2.3 [76].

Beat annotation	Description	Class in this project
N	Normal	N (class 0)
L	Left bundle branch block	
R	Right bundle branch block	
e	Atrial escape	
j	Nodal escape	
A	Atrial premature	S (class 1)
a	Aberrant atrial premature	
J	Nodal premature	
S	Supra-ventricular premature	
V	Premature ventricular contraction	V (class 2)
E	Ventricular escape	
F	Fusion of ventricular and normal	F (class 3)
/	Paced	Q (class 4)
f	Fusion of paced and normal	
Q	Unclassifiable	

Table 2.1: Beat annotation in the database and beat classes in this project.

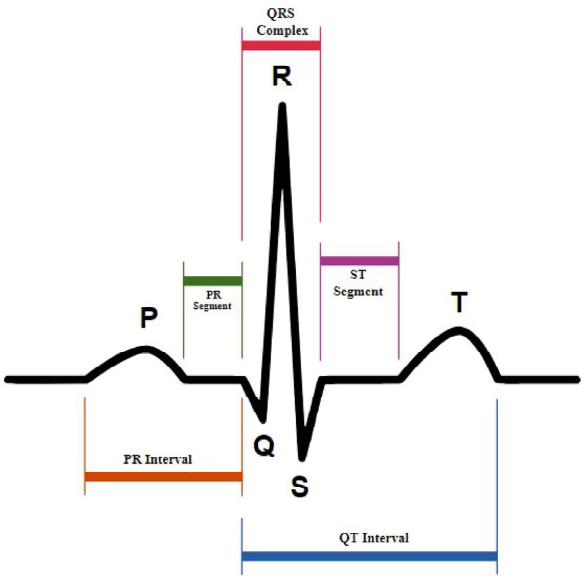


Figure 2.3: Different parts of a heartbeat signal [77].

2.2 Related Work

There exist several recent works applying SNNs for use cases that require ECG as input data. Y. Feng *et al.* in [78] have built an SNN, using the ANN to SNN conversion approach, to classify ECG data into four classes, mainly to detect atrial fibrillation (AF). They have used 2017 PhysioNet/CinC Challenge [79] data set. Also, they have tested multiple activation functions in ANN and their experiments show that the accuracy of SNN converted from the ANN with ReLU activation functions has the best overall accuracy result, which is 84.41%.

In [80], F. Corradi *et al.* have proposed a method for encoding and compressing the ECG signals into spike trains. Then, they have designed a recurrent SNN for heart beat classification to distinguish 17 types of cardiac patterns. Moreover, they have implemented the network on a mixed-signal analog/digital Very Large Scale Integration (VLSI) neuromorphic processor.

Another similar work has been done by K. Buettner *et al.* [81]. They have used *SNN-Toolbox* framework to convert their designed CNN for heartbeat classification into its SNN counterpart. Moreover, they have implemented this SNN on Loihi and provided the latency, power, and energy efficiency measurements and compared them with those of an architecturally identical ANN implemented on Intel Core i7 CPU, Intel Neural Compute Stick 2, and Google Coral Edge TPU devices.

2.3 ECG Classification Algorithm Based on STDP and R-STDP Neural Networks

Another project that we based our work on, was Amirshahi and Hashemi's [82]. Although we could not produce a final result because of an error occurred during training step using Brian2 package for the implementation, the whole effort gave us a valuable insight into the methods used for implementing SNN for ECG classification problem. In this section we discuss about this work and the important ideas behind it.

The goal of this paper is to continuously monitor the ECG signal in real-time, on the wearable device itself and to classify them. Wearable devices have limited computational and energy resources. On the other hand, neural network algorithms are able to extract the features from data and are more resilient to variations among different ECG waveforms [83]- in the classical features of ECG signals, there can be remarkable variations among different people and under different conditions -, thus, they result in high accuracy for arrhythmia detection. However, these neural networks usually require large power and high computational resources. To address this issue (the conflict between high accuracy and low power resources when using ANNs), the paper uses an SNN.

The overall view of the proposed solution can be seen in Figure 2.6. The input of the network is a heartbeat. Here, a heartbeat is considered as a segment of the ECG signal starting from 0.25 seconds before an R peak and ending at 0.45 seconds after that R peak. Then, each heartbeat is split into 7 overlapping windows. Each of these windows contains 64 samples. These points are actually the inputs of the first layer which will then be encoded into spike signals. As we explained in the introduction section, this means that the information is encoded in the timing of the spikes and not in their amplitude. Also, the Leaky Integrate-and-Fire (LIF) neurons has been used in this SNN. If we associate the weight w_{ij} to synapse ij that connects pre-synaptic neuron i and post-synaptic neuron j, and if we call the membrane potential of neuron j as u_j, the LIF neuron operates based on the following differential equation:

$$\tau \frac{d}{d_t} u_j(t) = -(u_j(t) - u_{rest}) + \alpha \sum_i s_i(t) w_{ij} \tag{2.1}$$

The sum iteration is over all neurons whose outputs are connected to neuron j. Also, $s_i(t)$ is one if neuron i has produced a spike at time t and zero otherwise. Here we can say that when w_{ij} is larger, it is more probable that the spike of neuron i causes a spike in neuron j. Moreover, the first term represents the role of leakage current that causes membrane potential u_j to go toward the rest potential u_{rest} with time constant τ. The constant parameter α regulates the strength of the second term (related to spikes) compared to the first term (the leakage current) in changing the membrane potential.

Now, we explain each layer in more detail and discuss the results.

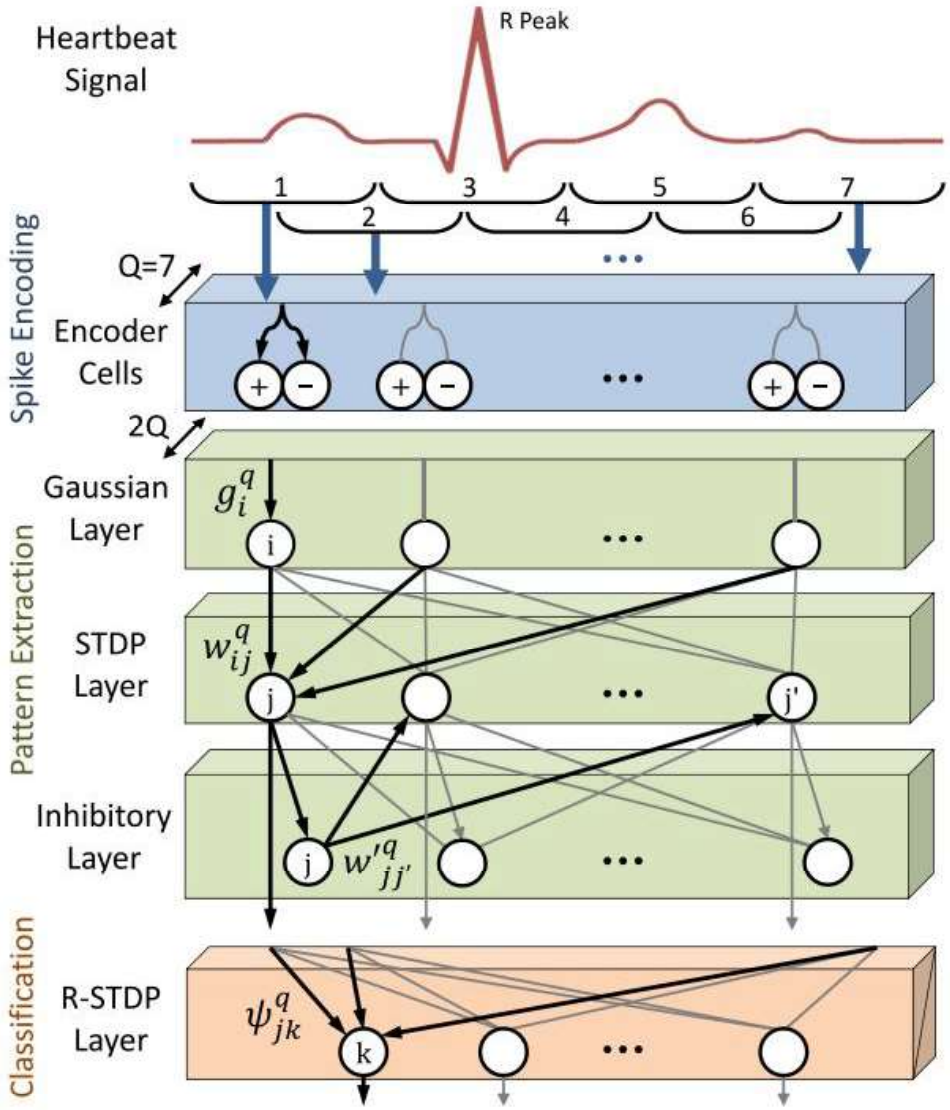


Figure 2.4: Overall view of the proposed solution [82].

Encoder

As we mentioned earlier, the input of the network is the samples of 7 windows. Let $X_{ecg}^{(q)}$ be the the portion of the heartbeat signal X_{ecg} which falls in window $q \in [1, 7]$. Therefore, $|X_{ecg}^{(q)}| = 64$ (because each window has 64 samples). For every sample $i \in [1, 64]$ from every window, there exist two encoder cells. If the sample is positive, the first encoder cell produces the spikes and if the sample is negative, the second one does it. Spikes are generated randomly based on the Poisson process and the spike firing rate of this random Poisson process is set proportional to $X_{ecg}^{(q)}[i]$. In total, there are $64 \times 7 \times 2 = 896$ encoder cells in the spike encoding layer. Also, for each window there are $64 \times 2 = 128$ synapses carrying and transferring the generated spikes to the next layer. As shown in Figure 2.3, there exist $2 \times 7 = 14$ windows in the next layer since the synapses are split in two groups: The positive encoder cells are connected to the synapses in the odd windows, and the negative encoder cells are connected to the synapses in the even windows.

Gaussian Layer

In this layer, every window $q \in [1, 2 \times 7]$ is processed separately. Input synapse i in window q is connected to one neuron and its synaptic weight $g_i^{(q)}$ changes the spike firing rate by a factor of that weight. Let $R_{in,i}^{(q)}$ and $R_{out,i}^{(q)}$ be the rate of spikes on the input and on the outputs of the neuron in window q, respectively. Then, we can write:

$$R_{out,i}^{(q)} = g_i^{(q)} \times R_{in,i}^{(q)} \tag{2.2}$$

To reduce the effect of the heartbeat peak that may appear on the side of a window (in addition to the middle of another neighboring window, due to the overlap between windows), $g_i^{(q)}$ is set to a Gaussian kernel:

$$g_i^{(q)} = \beta^{(q)} \times \frac{1}{\sigma\sqrt{2\pi}} \times e^{-\frac{1}{2}(\frac{i-\mu}{\sigma})^2} \tag{2.3}$$

To put this Gaussian kernel at the center of each window μ is set to $\mu = \frac{1}{2}|X_{ecg}^{(q)}|$. Also, in the paper, σ is chosen such that the effect of a side peak is reduced but not completely neglected. $\beta^{(q)}$ is a trainable parameter being different for each q. It should be trained in such a way that makes the average firing rates of different windows to be similar. Let $\overline{R_{out}^{(q)}}$ be the average number of spikes fired by the neurons in window q in this layer. $\beta^{(q)}$ is initialized to 1 and then the update rule for that in training step is as the following:

$$\Delta\beta^{(q)} = \alpha \times (1 - \frac{\overline{R_{out}^{(q)}}}{R_{target}}) \tag{2.4}$$

STDP Layer

STDP is a phenomenon found in live neurons by Bi and Poo [46] and has been a source of inspire for learning event-based networks. STDP learning is an unsupervised learning algorithm based on dependencies between times of pre-synaptic and post-synaptic spikes [84]. Here, the synaptic weight is also called plasticity. Assume that $w_{ij}^{(q)}$ is the synaptic weight in window q for synapse ij, where i and j denote the pre-synaptic and post-synaptic neurons, respectively. These weights are trained as the following. The synaptic weight increases if the spike time of the post-synaptic neuron (shown as t_{post}) is after (during a specific time window) the spike time of the pre-synaptic neuron (shown as t_{pre}). This is called long-term potentiation (LTP). Similarly, a decrease in the synaptic weight happens if the post-synaptic spike occurs before the pre-synaptic spike. This is called long-term depression (LTD). This learning rule is shown in Figure 2.4. We can see that, the smaller the time difference $\Delta t = t_{post} - t_{pre}$ is, the larger the amount of change in the synaptic weight would be ($\Delta w = w_{new} - w_{old}$). The equation for this STDP learning rule is:

$$\Delta w = \begin{cases} a^+ \times e^{(-\frac{|\Delta t|}{\tau})} & \Delta t > 0 \\ a^- \times e^{(-\frac{|\Delta t|}{\tau})} & \Delta t < 0 \end{cases} \tag{2.5}$$

Where a^+ and a^- are learning rates which are positive and negative constant values, respectively, and τ is the time constant. However, in Amirshahi and Hashemi paper, they have modified this learning rule with a reasonable intuition to alleviate some problems of the above-mentioned rule.

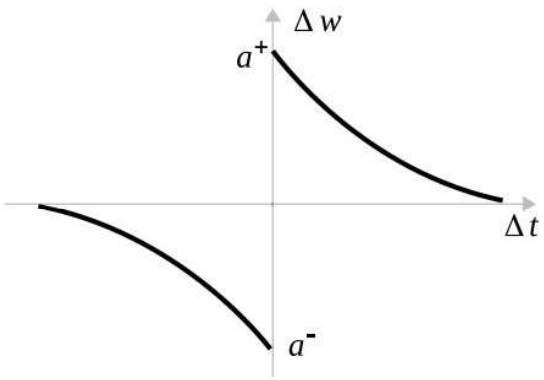


Figure 2.5: Learning rule for the synaptic weights update in STDP layer

Inhibitory Layer

In the previous layer, all neurons might extract the similar patterns of spikes. To avoid this and to capture different spike patterns, inhibitory neurons are added. In the inhibitory Layer there exists exactly one inhibitory neuron for every neuron

in the STDP layer. When a neuron j in the STDP layer fires, its corresponding inhibitory neuron j fires as well and prevents all the other neurons $j' \neq j$ in the same window from firing in the STDP layer. As can be seen in Figure 2.6 there are negative backward synaptic weights $w'^{(q)}_{jj'}$ that decrease the membrane potentials of neurons $j' \neq j$ in STDP layer when carrying spikes coming from neuron j in STDP layer. The learning rule that works in this scenario is as the following:

$$\Delta w' = \begin{cases} b^- & |\Delta t| \leq \lambda \\ b^+ & |\Delta t| > \lambda \end{cases} \tag{2.6}$$

Note that b^- and b^+ are negative and positive constant values, respectively. This means that if any neuron j' (which is in the same window q as neuron j), spikes around the same time as neuron j, the weight $w'^{(q)}_{jj'}$ would decrease (would become more negative). Adding this inhibitory layer might cause a problem: when there is not much difference between all generated patterns, this inhibition plays a significant role and decreases the number of spikes and therefore weaken the power of STDP training. In Amirshahi and Hashemi's paper [82], a modified version of the previous update rule has been introduced. We do not intend to discuss it here, but it can be checked for more details. It is also worth mentioning that inhibitory neurons only exist during the training step, and they would be eliminated during the test step.

Reward-modulated STDP (R-STDP) Layer

This layer acts as the classifier. Each neuron in this layer corresponds to one class and the one firing more frequently (the winner neuron), determines the predicted heart-beat class. Unlike STDP layer that would be trained according to a learning rule without providing any true labels (unsupervised learning approach), R-STDP layer would be trained according to supervised learning procedure. During training, if the winner neuron k predicts correctly, a reward is applied to all its synaptic weights $\psi^{(q)}_{jk}$. It can be seen in Figure 2.6 that j represents all the pre-synaptic neurons from all windows q that are connected to the winner neuron k. Similarly, if the prediction is incorrect, a punishment signal is applied. The learning equation used here, is as follows:

$$\Delta\psi = \begin{cases} a^+_r \times \psi(\psi_{max} - \psi) & t_{post} > t_{pre} \\ a^-_r \times \psi(\psi_{max} - \psi) & t_{post} \leq t_{pre} \end{cases} \tag{2.7}$$

$$\Delta\psi = \begin{cases} a^-_p \times \psi(\psi_{max} - \psi) & t_{post} > t_{pre} \\ a^+_p \times \psi(\psi_{max} - \psi) & t_{post} \leq t_{pre} \end{cases} \tag{2.8}$$

When the prediction is correct, the first equation is used where a^+_r and a^-_r are learning rates in this reward situation. Similarly, when the prediction is incorrect, the second equation is used where a^+_p and a^-_p are learning rates in this punishment situation. There are two cases in each situation: $t_{post} > t_{pre}$ and $t_{post} \leq t_{pre}$. This is because this layer also has the STDP characteristics. The term $\psi(\psi_{max} - \psi)$ is used to prevent the value of weight ψ go far beyond $\psi = 0$ or $\psi = \psi_{max}$.

Paper	Model	Accuracy
Amirshahi *et al.*	SNN	97.9%
Hu *et al.*	MLP	94.8%
Crippa *et al.*	KLT, GMM	98.9%
Kachuee *et al.*	CNN	93.4%
Kiranyaz *et al.*	FFT, CNN	98.6%
Saadatnejad *et al.*	Wavelet, RNN	99.2%
Ince *et al.*	Wavelet, MLP	97.6%
Kolagasioglu *et al.*	Wavelet, SNN	95.5%
Lee *et al.*	Wavelet	97.2%

Table 2.2: Results of some other similar works done for heart-beat classification [82].

Training and Results

As we mentioned before, STDP has the unsupervised learning nature but R-STDP is of supervised learning nature. Thus, the layers should be trained consecutively: First, the Gaussian layer would be trained. Then, the STDP and inhibitory layers are trained at the same time. At the end R-STDP layer should be trained. After training the whole network, the trained weights should be stored to be used for the test stage.

Finally, a result summary of some similar works has been provided in the paper that we have added it here in Table 2.2. It would be a solid base for providing a comparison with our own results, later.

Chapter 3

ECG Classification Using Spiking Neural Networks

In this section, we explain the main work we have done in this thesis. As we mentioned in the first section, there are mostly two points of view when it comes to the SNN implementation: 1) in direct method, the neural network is originally designed based on dynamicism paradigm, to capture temporal dimension of information (e.g. spikes and spikes timing). An example of using this method is ECG Classification by Amirshahi and Hashemi [82] that was described in the previous section, 2) in indirect method, a (non-spiking) neural network is trained, and then we store the learned parameters, and convert it to an SNN, test it on our data and tune the required parameters. In the conversion step, we need to take care of some adjustments to have better classification results. In this thesis, we used the indirect method.

The main goals of this project are to investigate if it is possible to build an SNN for ECG classification while not sacrificing the accuracy, and to explore several actions we can take to improve the final results.

3.1 Overview

Most existing works on signal classification that are based on machine learning methods, use time samples or frequency components of the signal. In this project we have used time-frequency joint distribution to capture time-varying frequency components. This has been inspired by A. Das *et al.* paper [89]. That paper is based on a two-phase approach to real-time heartbeat classification: 1. Derive the time-frequency joint distribution of each heart-beat, convert it into sparse distributed signatures, use them as inputs of a multi-layer perceptron (MLP), and train that network for classification. 2. Use the trained network to classify live ECG heart-beats. In this work, we have used a similar approach. However, for exploiting features of time-frequency joint distribution of a heart-beat, we design and use a CNN instead of generating sparse distributed signatures. Also, after training our whole network, we convert the CNN to SNN, which is then used for testing of heart-beat classification. Figure 3.1 outlines our proposed approach in this thesis.

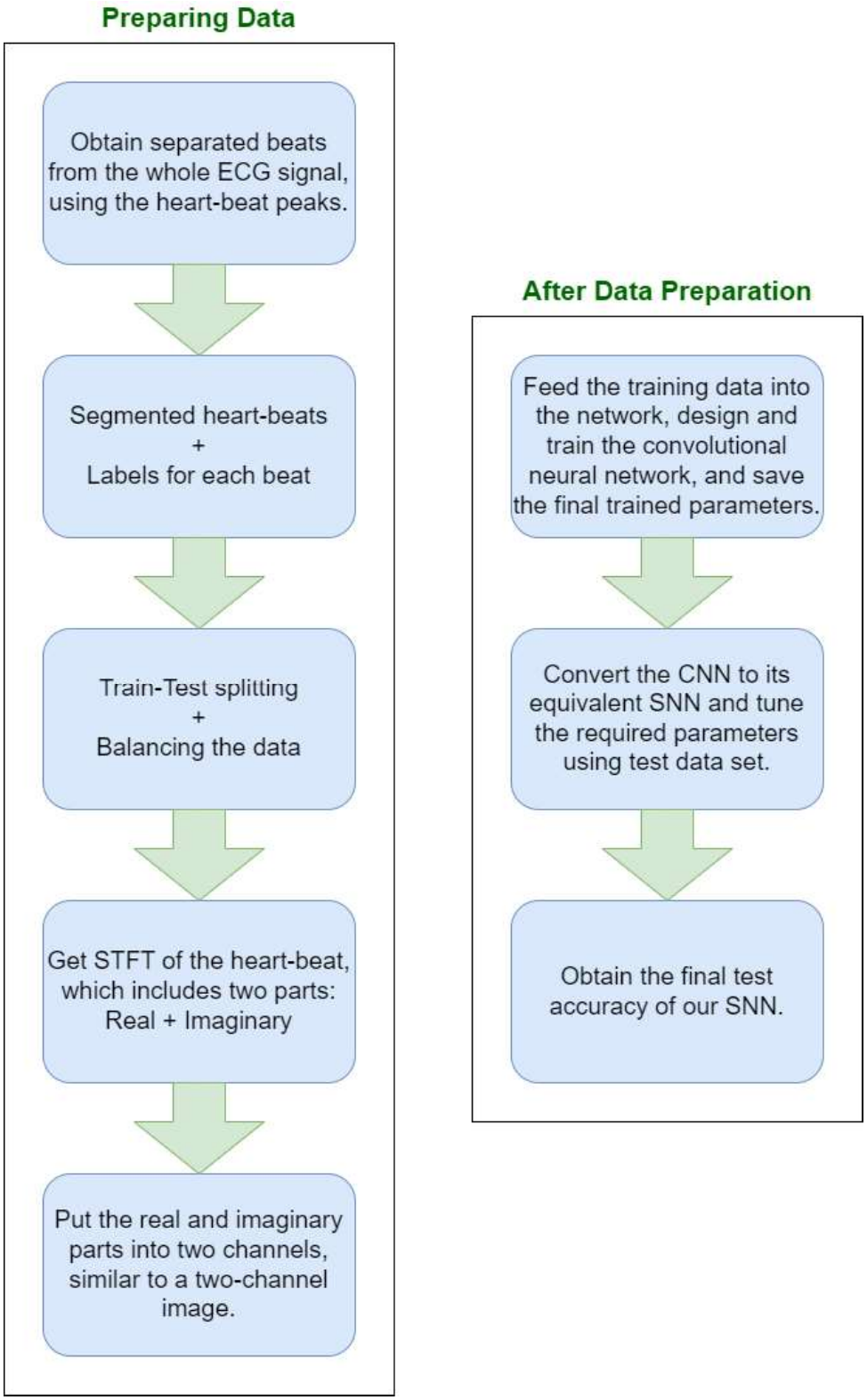


Figure 3.1: An overview of the approach of this project.

3.2 Preparing the Data

3.2.1 Heart-beat Segmentation

As the first step, heart-beats are segmented. We have done this by using the location of peaks in the ECG signal. A heartbeat is considered as a segment of the ECG signal starting from 0.25 seconds before an R peak and ending at 0.45 seconds after that peak. This is equivalent to acquiring 90 samples before and 162 samples after the peak sample, since the sampling frequency of the signal is 360 Hz ($360 \times 0.45 = 162$, $360 \times 0.25 = 90$). Thus, after this step we have all segmented heart-beats and their labels that were introduced in Table 2.1.

3.2.2 Balancing Data

In this step, we first check if our data is balanced, and, if not, how to balance it. By data balancing, we mean all class labels are fairly represented in the dataset. This is important for the assessment of our classification result. Consider we have an extremely imbalanced data set, for instance, 90% of its samples belong to class 1, and the remained samples (10% of the whole data set) belong to all other classes. In this scenario, if the classifier simply predicts class 1 for all the input samples, the overall classification accuracy will be 90% which might seem pretty good at the first glance. However, the per-class accuracy is 0% for all classes except the first one which is 100%. Therefore, making the data set relatively balanced is of great importance. Table 3.1 shows the number of samples and the contribution percentage of each class. There exist 109452 heart-beat samples in total. According to the table, the data set is highly imbalanced. We investigate two ways for making our data more balanced and describe these methods as follows.

1. Down-sampling

As can be seen in Table 3.1, Class 0 has far more data samples than four other classes. Thus, in the first method, we tried to decrease this huge gap by taking a simple action. The average number of samples in Class 1 to 4, is 4714, thus, we randomly select 4714 out of 90594 samples for Class 0. Therefore, data imbalance decreases. The problem of this method is that the number of data samples is not enough. We would see that our neural network could not be trained well and the accuracy result is not satisfying at all.

2. Combination of down-sampling and up-sampling

In the second method, we tried to alleviate the problem of the previous one (insufficiency of the data samples for training). Therefore, we used up-sampling for Class 1 to 4, in addition to using down-sampling for Class 0. We aimed for around 23000 samples for each class. Therefore, we randomly selected 23000 out of 90594 samples in Class 0. Then, we replicated the samples in other classes as much as there would be around 23000 samples in each class. It is worth mentioning that this step should be done separately in training and test set. It means that, test-train splitting should be done before the replication step, so that we would not

Class 0	Class 1	Class 2	Class 3	Class 4	Average number of samples in Class 1, 2, 3, 4
90594 (82.77%)	2781 (2.54%)	7235 (6.61%)	802 (0.73%)	8040 (7.35%)	4714

Table 3.1: The number of samples in each class.

have any exact same sample in both training and test sets. Thus, we could have more realistic assessment of our model when we test it. More details are depicted in Figure 3.2. We can see that we used 70% of our data for training the network. Also, we have 8 copies of each sample in Class 1, 3 copies of each sample in Class 2, 25 copies of each sample in Class 3 and 3 copies of each sample in Class 4.

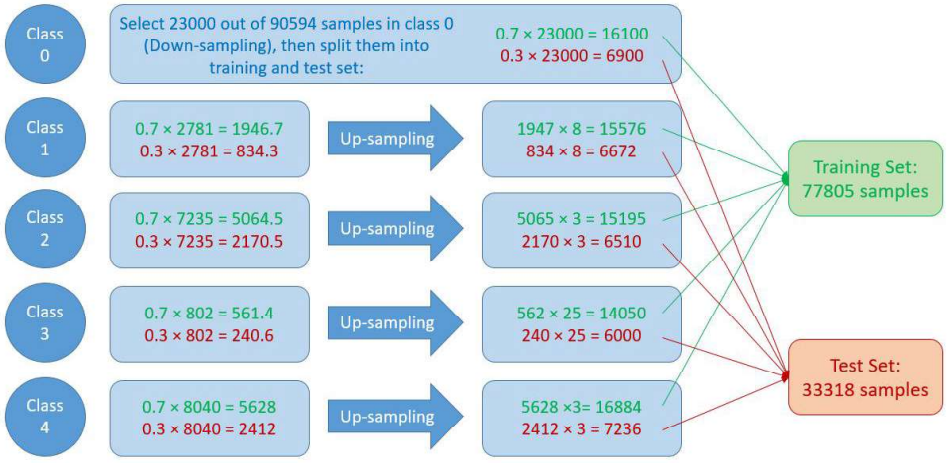


Figure 3.2: More details of balancing data. The upper group (in green) includes the training samples and the lower one (in red) contains the test samples.

3.2.3 Short-Time Fourier Transform

ECG signals have transient nature and are considered as non-stationary signals. Time-frequency joint distributions, such as short-time Fourier transform (STFT), can be used to capture time varying frequency components of these transient signals. In other words, by using STFT, we can obtain frequency content of local sections of a time signal. The short-time Fourier transform of a discrete-time signal

$x(t)$ is derived as the following [89]:

$$X(t,\omega) = \sum_{\tau=-\infty}^{\infty} h(t-\tau)x(\tau)e^{-j\omega\tau} \tag{3.1}$$

Here, $h(t)$ is a time window dividing the signal into shorter segments of equal length. Then, the Fourier transform can be calculated for each segment. Signal $X(t,\omega)$ is a two-variable function (time and frequency) containing two components, real and imaginary. Therefore, each component can be shown in a 3-D diagram.

We know that the narrower a function is in one domain (time or frequency) the wider it will be in the other domain. Therefore, the length of the window function $h(t)$ is related to the time–frequency resolution. This means that a wider window results in better frequency resolution but worse time resolution, and similarly, a narrower window leads to better time resolution but worse frequency resolution. This could help us experimenting with some parameters required for STFT and deciding about their values. Another parameter, besides the window length, is the length of the overlap between successive segments of input signal, generated by $h(t)$. This overlap helps to minimize the data loss in the window boundary. In this project, we used the Gaussian window with length 30 and overlap length 20. By choosing these parameters the STFT of the heart-beats would be quite similar to A. Das *et al.* work [89] with which we wanted to compare our final results. Two heart-beats from two different classes and the real component of their STFT are shown in Figure 3.3. The dimension of our obtained STFT, for both real and imaginary components, is 120×23 (along frequency and time axis, respectively).

3.2.4 Two-channel input to neural network

The last step of preparing our data is to put two 2-D data (real and imaginary part) together to have a structure of a two-channel input. This is a pretty typical data structure to feed into CNN networks.

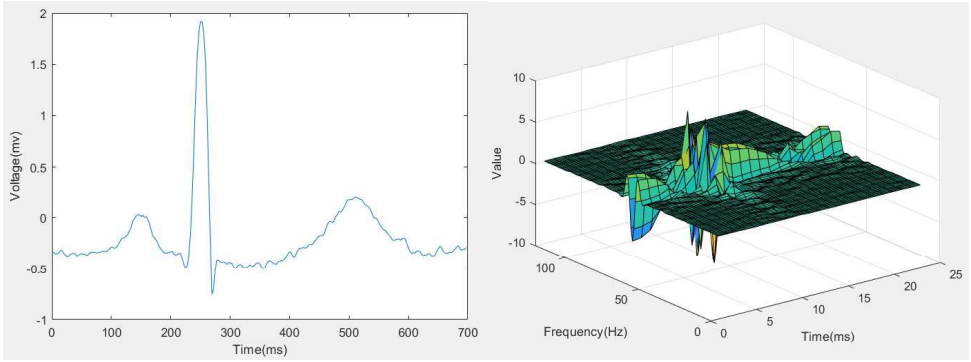
3.3 Building our CNN Model

Before designing and testing our CNN, we consider using a random forest for our heart-beat classification problem using our final prepared data (STFT version of the heart-beats) to have an idea about the expected accuracy. A random forest with 100 trees resulted in an overall accuracy of 88.71. As we mentioned before, our final goal is to build an SNN for heart-beat classification and we want to use the conversion approach, thus, we cannot work with random forest and need a neural network instead. Now, we should start building our neural network. We mentioned in the previous section that we explored two methods for balancing data. Here we discuss the results of both methods.

3.3.1 Using down-sampling method for balancing data

In this case, we had 23572 samples in total. We experimented some different CNNs, but none of their results was satisfying. In fact, the overall test accuracy

a) A heart-beat belonging to Class 0



b) A heart-beat belonging to Class 2

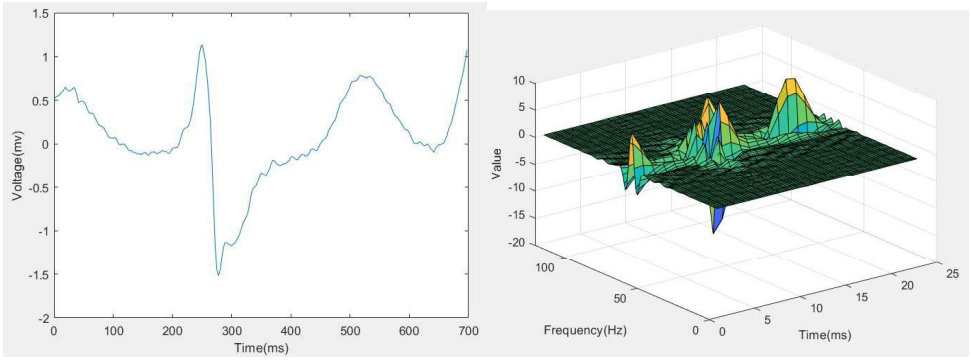


Figure 3.3: Two heart-beats belonging to two different classes and the real part of their STFT. The first one is a normal beat. The second one, which is a ventricular escape beat, has some distinguishing characteristics such as wide QRS complex and abnormal morphology of QRS complex.

was around 60-65% using a network with around 115000 trainable parameters. Therefore, we realized that the number of samples might not be sufficient and the network was not able to learn enough. Therefore, we decided to increase the sample size and use the second method to make data balanced while having more samples.

3.3.2 Using joint down-sampling and up-sampling method for balancing data

Here, we had 111123 samples in total as we showed in Figure 3.2. After several experiments, our final CNN architecture, that we used before converting to the SNN version, is depicted in Figure 3.4. We observe that the input is of size $120 \times 23 \times 2$ that corresponds to what we described before. Then, we have two convolutional layers using 64 kernels for each. They are followed by the average pooling, another convolutional and a few fully connected layers ending with a 5-neuron output layer to solve our 5-class classification problem. This network has 105,279 trainable parameters. It is worth mentioning that we used Keras API of TensorFlow to build our network. Moreover, we used NengoDL to train our CNN, convert it to SNN and finally test it. We could obtain around 95% accuracy after 10 epochs using the CNN showed in Figure 3.4. This CNN could even reach 98% accuracy using more epochs. At this point, we had an acceptable result, thus, we saved the trained parameters using $save\_params$ function of $nengo\_dl.Simulator$ and then moved on to the next step which is CNN to SNN conversion.

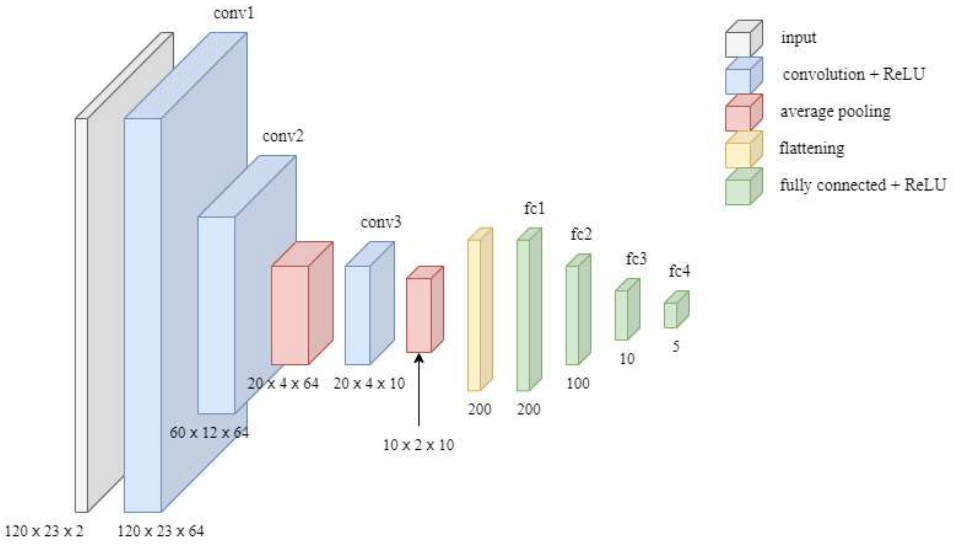


Figure 3.4: Structure of our NN with 105,279 trainable parameters.

3.4 CNN to SNN Conversion

We converted our CNN to its spiking version by using $nengo\_dl.Converter$. This would perform the conversion by replacing the neurons activation functions from $tf.nn.relu$ in CNN version to $nengo.SpikingRectifiedLinear()$ to have the SNN version. Here, we had to carefully configure some tuning parameters including: **scale firing rate**, **number of steps** and **synapse value**. We would discuss the effect of each of them, but before that note that the initial accuracy result of our SNN without tuning the parameters (scale firing rate = 1, synapse = None, number of steps = 10) was around 20%, which is a very low accuracy. Therefore, we had to experiment with different values to realize their effect and think about the explanation and intuition behind that behaviour. Finally, we could find an acceptable set of parameters that improves the performance of the spiking model. Here, we explain these parameters and their roles. We leave the explanation of the parameters' optimization in the next section.

3.4.1 Scale Firing Rate

We can think of a typical ANN as an SNN that spikes continuously and non-stop [90]. In other words, the firing rate is infinite in a typical ANN. We could expect that the SNN performance improves by increasing the neurons spike firing rates, because then, the neurons can update their output more frequently. We could use the $scale\_firing\_rates$ parameter in NengoDL converter when we intend to convert our CNN to SNN for testing the performance of the spiking model, without retraining our network [90]. The larger scale parameters means more spikes to process, which is a negative factor for energy efficiency. The infinite scale firing rate leads to the behavior of ANNs, which we criticize for their inefficiency. The effect of the scale firing rate on our spiking model accuracy (while other parameters are fixed) is shown in Table 3.2. We observe that the overall trend when raising the firing rate is an increase in accuracy.

Figure 3.6 and 3.7 show the effect of firing rate on the neural activity and the output prediction (performance). Two samples of each class has been selected and feed into our SNN, once with $scale\_firing\_rates = 50$ depicted in Figure 3.6, and once with $scale\_firing\_rates = 30$ depicted in Figure 3.7 while other parameters are the same. We observe that the first network performs better and with more confidence, which means that it reaches a more stable prediction sooner.

3.4.2 Synapse time constant

Now, that we have a spiking model, which means the model functions based on discrete events called spikes, we can move on to the next step. In fact, the neurons' outputs have spike nature and each spike is present for a very short time and disappears quickly. Therefore, we could expect that the output predictions would be very noisy without applying any low-pass filter. Also, even if correct output neuron spikes pretty quickly, there is no guarantee that it will spike on exactly the last time-step, which is the time we check the result [90]. To mitigate the problem of these rapid changes and to help smoothing the spikes, we should use

scale_firing_rate	1	10	50	75	100
overall accuracy	20.51%	19.64%	25.09%	22.7%	30.11%

Table 3.2: The effect of the scale firing rate on overall accuracy, with parameters: n_steps=10, synapse=None.

synaptic filters provided in Nengo. The default synapse used in Nengo is a low-pass filter, and the value we specify for the synapse parameter, is the low-pass filter time constant enforced on every neuron output. We can imagine that this would act as computing a running average of each neuron's activity over a short window of time, instead of just looking at the spikes on the last timestep [90]. As a result, it is of high importance to mention that using large time constant (the synapse parameter value) leads to the longer time required for the output neurons to settle. The intuition behind this fact is that the filters with long duration result in inferior output responsiveness to rapid changes in the input. Thus, we need to average over longer time window of neuronal activities. This means an increase in the latency and less responsiveness. In fact, there exists latency versus accuracy trade-off. After trying several values for synapse time constant, we decided to use the value synapse=0.01.

3.4.3 Number of Steps

It is worth reminding that spiking neural networks and Nengo models always run over time. Therefore, we should enter time into our data. We do that by running our spiking model (used for testing step) for several time steps (instead of a single time) to be able to collect the spike data over time [90]. In more details, we add a fourth dimension to our data used for putting the replications of the actual data. This means that a single input sample was of size $120 \times 23 \times 2$ and now it would be $120 \times 23 \times 2 \times n\_steps$. Note that, our data set size is relatively huge and it is very important to consider that the replication action, or in other words extending our data over time, can be very time-consuming and also can cause memory shortage (depending on the number of steps and the way we implement it). Thus, we are not able to set the number of steps as large as we desire. Particularly, it would affect the capacity required and delay of the model, and this would question the underlying goal of the project. Therefore, again there exists a trade-off between latency and memory capacity versus accuracy [90].

Finally, we can see the overall accuracy and per-class accuracy results of SNN of Figure 3.2, for different number of steps in Table 3.3.

3.5 Optimizing the Parameters Using Regularization

Here, we have our last chance to experiment a little more with the architecture of our previously fixed neural network to see if we could achieve better results.

	n_steps=40	n_steps=60	n_steps=120
overall accuracy	53.90	86.7	90.78
accuracy for Class 0	57.48	87.14	93.84
accuracy for Class 1	56.58	81.15	80.79
accuracy for Class 2	77.53	93.36	96.51
accuracy for Class 3	20.17	73.30	81.87
accuracy for Class 4	54.73	96.53	99.31

Table 3.3: Results after balancing data and using SNN of Figure 3.4 for different number of steps and scale_firing_rate=100, synapse=0.01.

overall accuracy	92.39
accuracy for Class 0	95.97
accuracy for Class 1	88.07
accuracy for Class 2	96.45
accuracy for Class 3	80.53
accuracy for Class 4	99.14

Table 3.4: Results after balancing data and using SNN of Figure 3.4 with parameters: scale_firing_rate=50, synapse=0.01, n_steps=60.

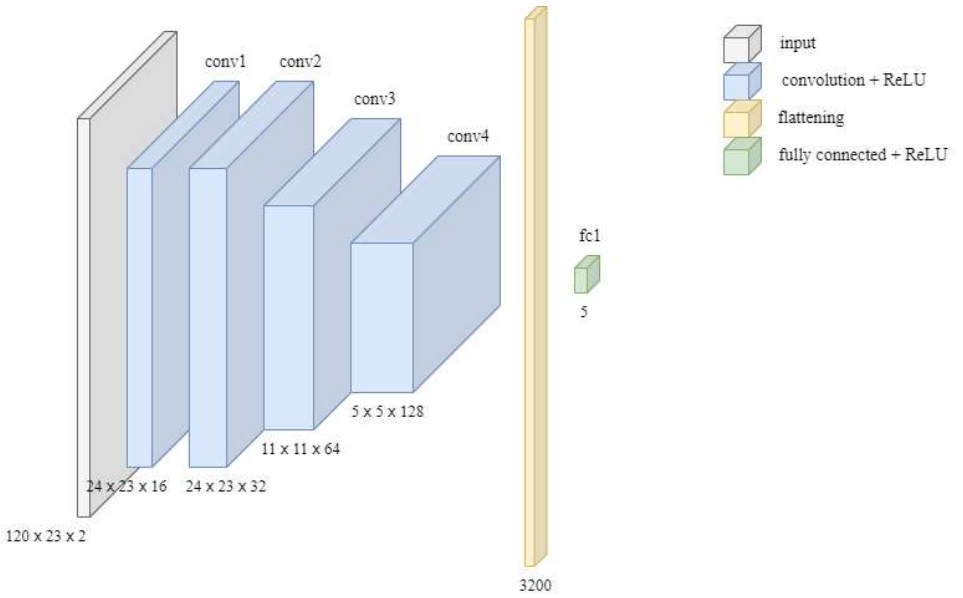
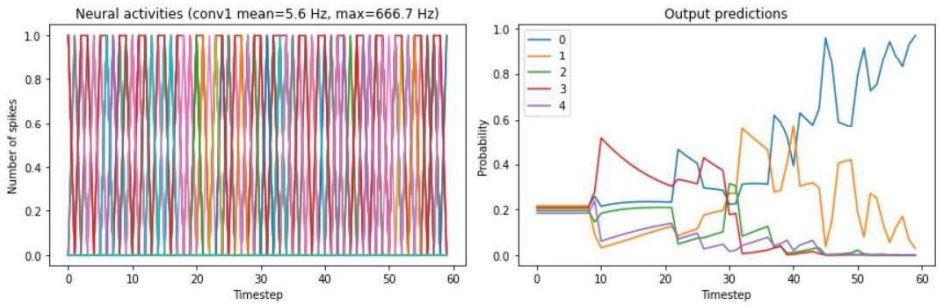


Figure 3.5: Our final NN structure with 114,453 trainable parameters.

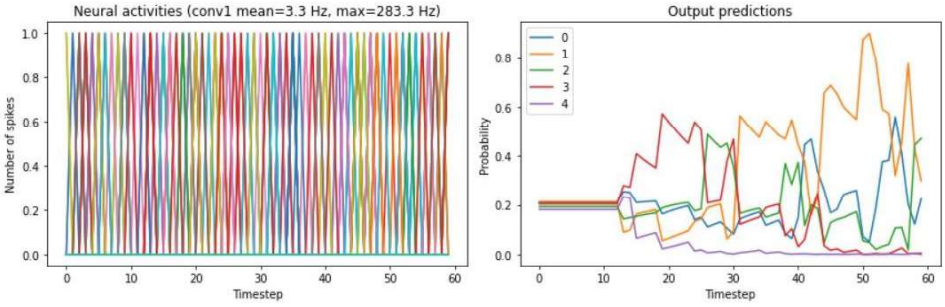
This leads us to a slightly different architecture showed in Figure 3.5 (this was tested with and without pooling layers). Then, we explore the space of other hyper-parameters, such as number of epoches, training batch size and the choice of optimizer. Also, we aim to optimize the firing rate for each convolutional layer instead of fixing their values from the beginning. We could accomplish this by adding loss functions that compute the mean squared error (MSE) between the neurons output of each of the convolutional layers and a specific firing rate we define for each layer. This approach is inspired by L2 regularization and we similarly should define a value for regularization hyperparameters (also called as regularization rate or λ parameter in L2 regularization) [90]. To implement this method, we should again train our CNN, but this time with using regularization term for different layers. Table 3.5 shows the results for our exploration in the hyper-parameters space. The setting of the colored row leads to the best SNN overall accuracy result.

architecture summary	#epochs	optimizer	training batch size	regularization hyperparameters	CNN test accuracy	SNN accuracy
no pooling	120	RMSprop(0.001)	105	0.001 all conv layers	86.68%	
pooling no regularizing	300	RMSprop(0.001)	105	0.001 all conv layers	86.82%	
pooling no regularizing	300	Adam(0.001)	195	0.001	74.82%	
pooling no regularizing	300	RMSprop(0.001)	195	1e-4, 1e-5	21%	
pooling no regularizing	300	RMSprop(0.001)	195	0.001 all conv layers	21%	
no pooling	300	RMSprop(0.001)	105	0.001	58%	
no pooling	400	RMSprop(0.001)	105	0.001 only last conv	96%	94.23%
no pooling	400	RMSprop(0.001)	105	0.001 only first conv	94%	86.63%
no pooling	400	RMSprop(0.001)	105	0.1 only last conv	95.39%	49%

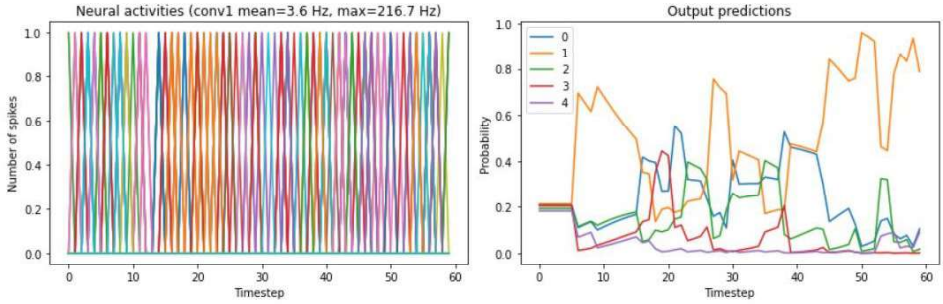
Table 3.5: Parameters space exploration using neural network of Figure 3.5, synapse=0.01, n_steps=30.



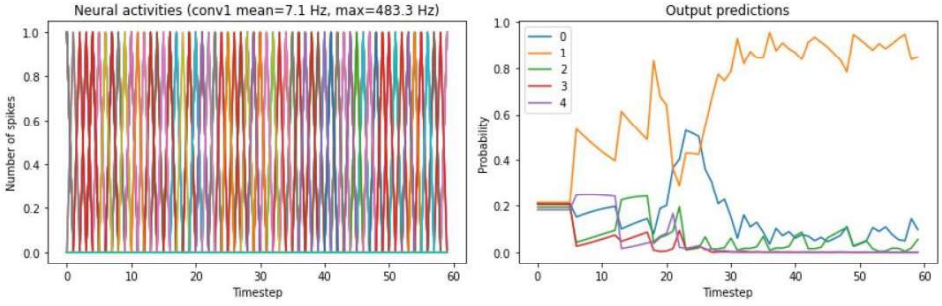
True Class: 0, Predicted Class: 0, Test sample number: 11



True Class: 0, Predicted Class: 2, Test sample number: 7

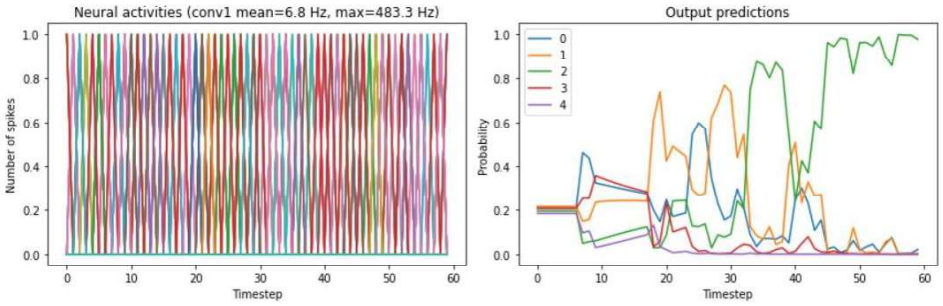


True Class: 1, Predicted Class: 1, Test sample number: 3

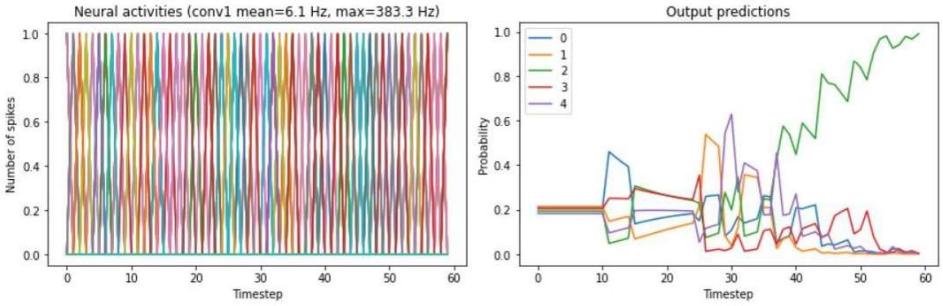


True Class: 1, Predicted Class: 1, Test sample number: 6

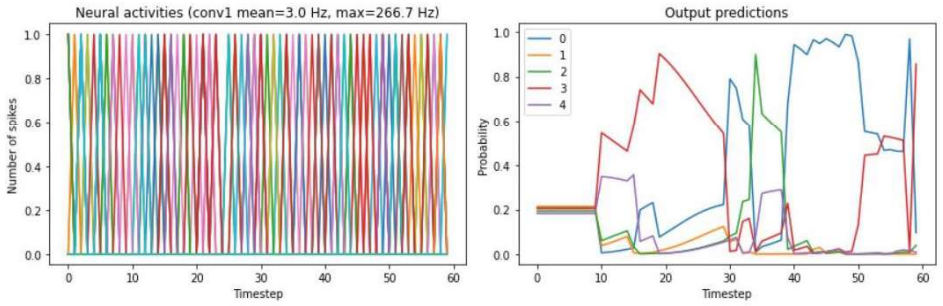
Figure 3.6: Neural activities of the first convolutional layer and the output predictions. scale firing rate = 50, number of steps = 60, synapse = 0.01.



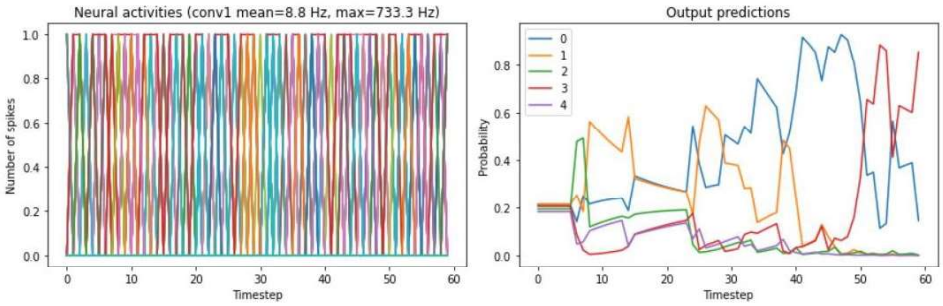
True Class: 2, Predicted Class: 2, Test sample number: 1



True Class: 2, Predicted Class: 2, Test sample number: 2

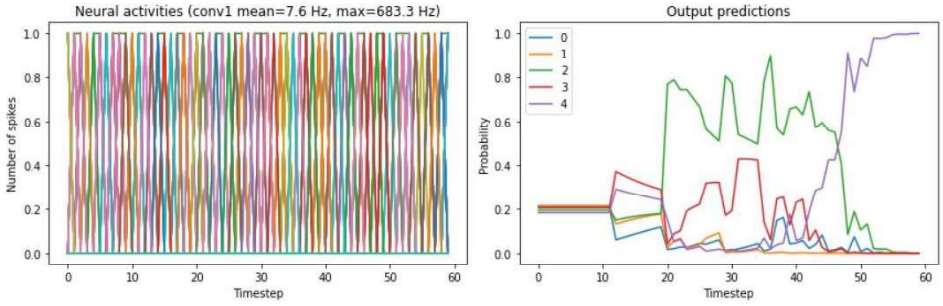


True Class: 3, Predicted Class: 3, Test sample number: 46

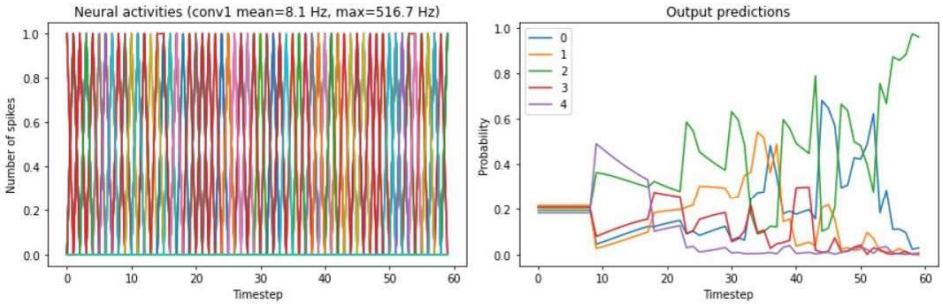


True Class: 3, Predicted Class: 3, Test sample number: 28

Figure 3.6: Neural activities of the first convolutional layer and the output predictions. scale firing rate = 50, number of steps = 60, synapse = 0.01.

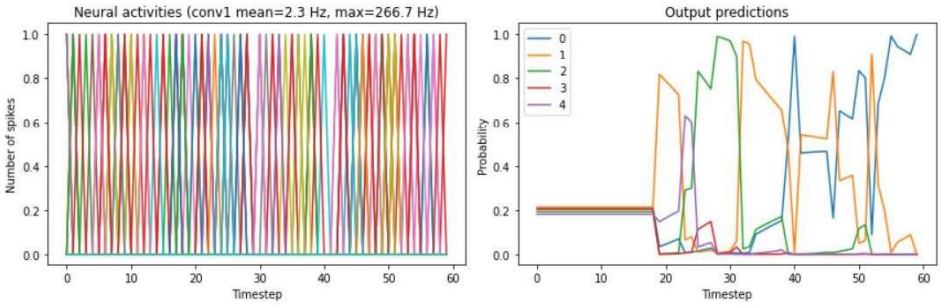


True Class: 4, Predicted Class: 4, Test sample number: 4

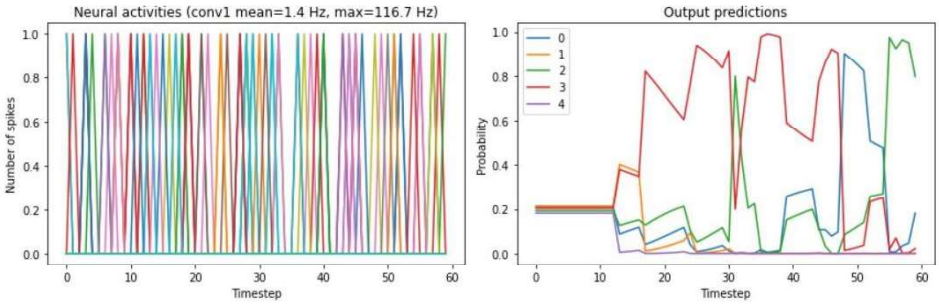


True Class: 4, Predicted Class: 2, Test sample number: 5

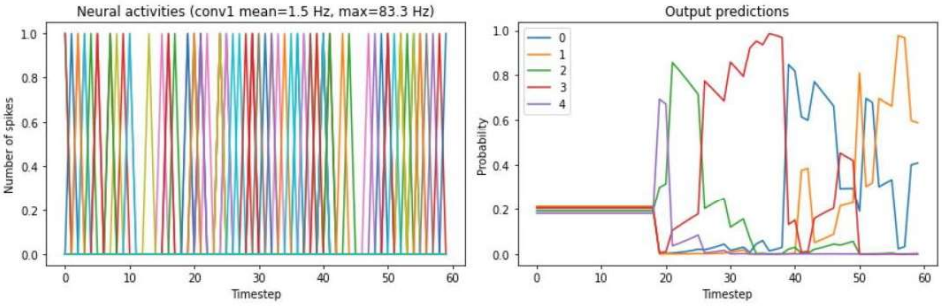
Figure 3.6: Neural activities of the first convolutional layer and the output predictions. scale firing rate = 50, number of steps = 60, synapse = 0.01.



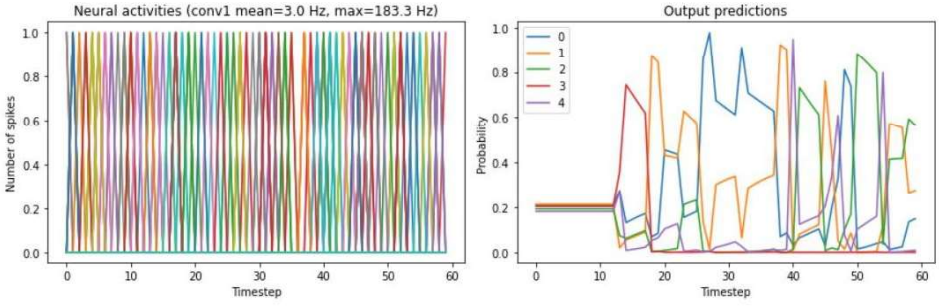
True Class: 0, Predicted Class: 0, Test sample number: 11



True Class: 0, Predicted Class: 2, Test sample number: 7

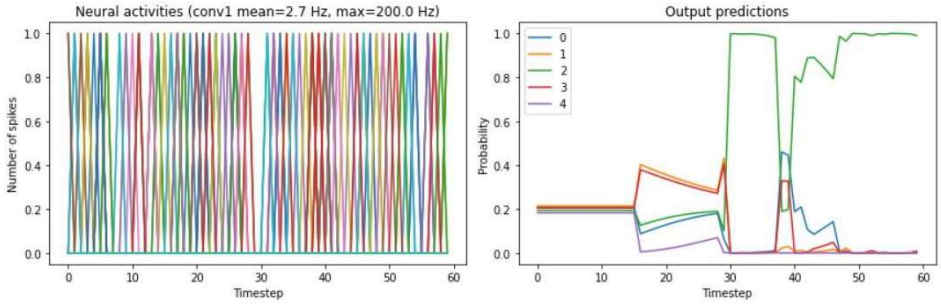


True Class: 1, Predicted Class: 1, Test sample number: 3

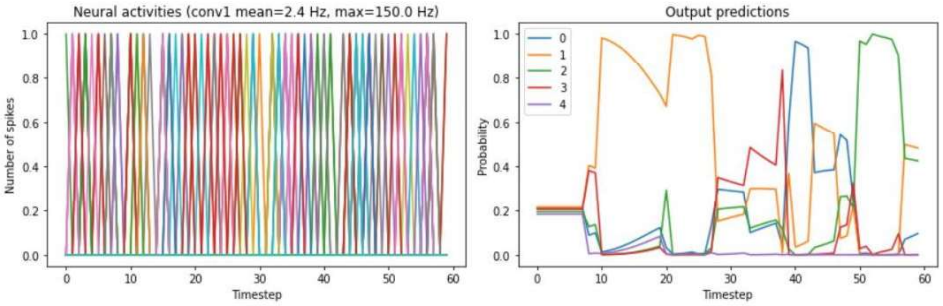


True Class: 1, Predicted Class: 2, Test sample number: 6

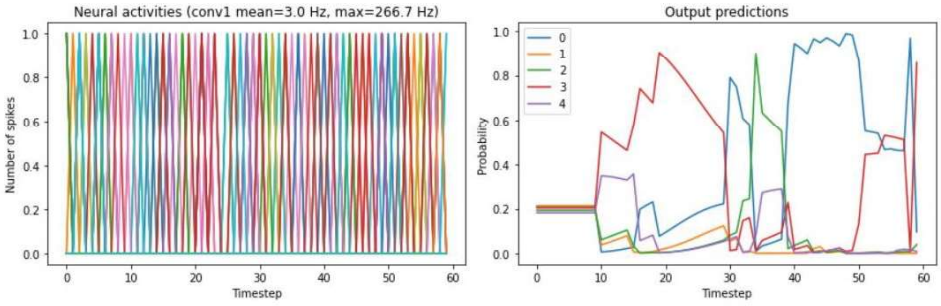
Figure 3.7: Neural activities of the first convolutional layer and the output predictions. scale firing rate = 20, number of steps = 60, synapse = 0.01.



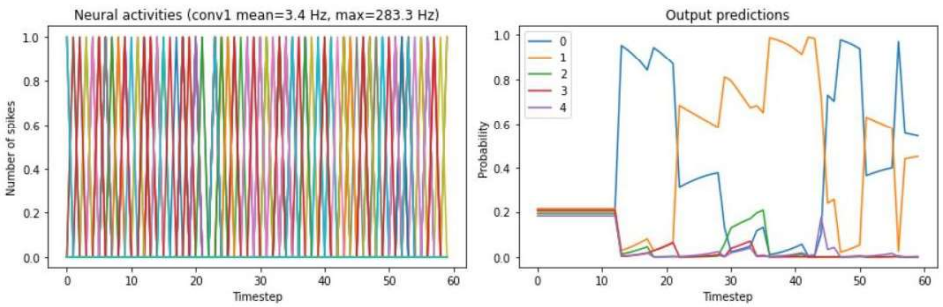
True Class: 2, Predicted Class: 2, Test sample number: 1



True Class: 2, Predicted Class: 1, Test sample number: 2

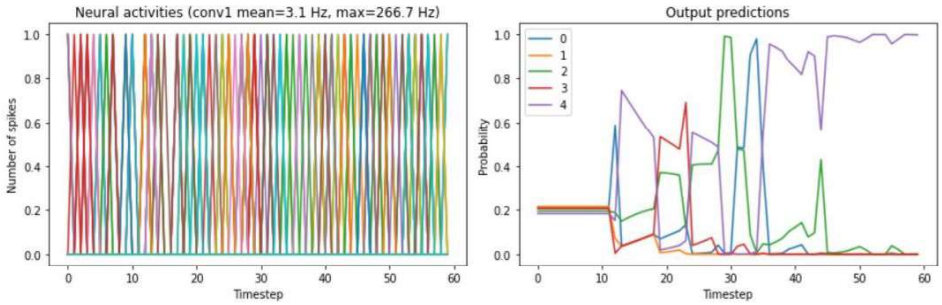


True Class: 3, Predicted Class: 3, Test sample number: 46

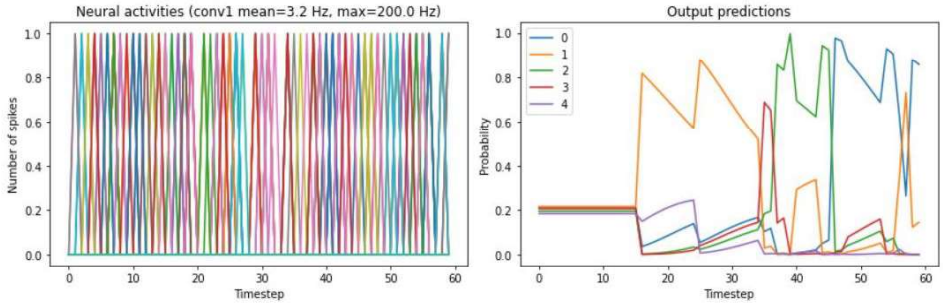


True Class: 3, Predicted Class: 0, Test sample number: 28

Figure 3.7: Neural activities of the first convolutional layer and the output predictions. scale firing rate = 20, number of steps = 60, synapse = 0.01.



True Class: 4, Predicted Class: 4, Test sample number: 4



True Class: 4, Predicted Class: 0, Test sample number: 5

Figure 3.7: Neural activities of the first convolutional layer and the output predictions. scale firing rate = 20, number of steps = 60, synapse = 0.01.

Chapter 4
Conclusions and Future Work

4.1 Conclusions

A. Das *et al.* in [89] state that their approach has resulted in an overall accuracy of 95.7% for heart-beat classification, which shows an improvement compared to the preceding studies. This result has been obtained using an ANN. However, we could reach an overall test accuracy of 94.23% while using an SNN. Moreover, we observe that our result is comparable to the ones in Table 2.2 (in Chapter 2) considering that we use SNN instead of conventional ANNs. Another point is that the previous proposed models usually do not perform well on heart-beats of Class 1 and Class 3. This can also be noticed in Table 3.4. But, these results still outperform several previous studies. For instance, one of the most recent papers using SNN for ECG classification [91] obtains around 68% and 66% accuracy for Class 1 and 3, respectively. Overall, our SNN model performs well in ECG classification compared to the state of the art work.

4.2 Ideas for Future Work

4.2.1 Using Direct Method for Building SNN

As we mentioned in the first chapter, another method for generating and using an SNN, besides ANN to SNN conversion, is building the SNN from scratch and training it directly without involving any ANN. This would have several advantages over the indirect method that includes training a non-spiking neural network. It is expected to alleviate the problem of high memory requirements and the slow speed of training if an efficient spiking learning algorithm could be developed. A novel example of such algorithm is proposed in Bojian *et al.* paper [92]. The authors of this paper have presented a recently developed alternative to BPTT, Forward-Propagation Through Time (FPTT) and its application for training the SNNs.

4.2.2 Implementation on a Neuromorphic Hardware

It is valuable to implement and evaluate our SNN on a neuromorphic hardware. Then, we would be able to measure and evaluate the energy consumption and

latency and compare the results with what is achievable with ANNs. This could be performed using *NengoLoihi* package and Intel's Loihi chip, or a simulated Loihi if the Loihi hardware is not available.

As we discussed earlier, it is beneficial to develop methods and algorithms that are capable of decreasing the energy consumption while not compromising the accuracy, so that by applying them, wearable devices performance and efficiency could be improved. Also, the existence of these methods and the possibility of their implementation on an actual hardware, can encourage the manufacturers to produce more health wearable devices. This makes such devices more accessible in the market and they might be used more often to raise the quality of life by notifying users about any abnormality in their heart signals and helping to decrease the probability of occurrence of dangerous and life-threatening situations.

References

[1] D. Sopic, Amin Aminifar, Amir Aminifar, D. Atienza, "Real-Time Event-Driven Classification Technique for Early Detection and Prevention of Myocardial Infarction on Wearable Systems." *IEEE Transactions on Biomedical Circuits and Systems*, July 2018.

[2] F. Forooghifar, A. Aminifar, L. Cammoun, I. Wisniewski, C. Ciumas, P. Ryvlin, D. Atienza, "A Self-Aware Epilepsy Monitoring System for Real-Time Epileptic Seizure Detection." *Mobile Networks and Applications*, 2019.

[3] K. Roy, A. Jaiswal, P. Panda, "Towards spike-based machine intelligence with neuromorphic computing." *Nature*, 575, 607–617, 2019.

[4] S. J. Russell, P. Norvig, 2010, *Artificial Intelligence: A Modern Approach.* 3rd ed. Prentice Hall.

[5] "What is Machine Learning?", IBM, Available online: https://www.ibm.com/topics/machine-learning

[6] O. Chapelle, B. Schölkopf, A. Zien, 2006, *Semi-Supervised Learning.* The MIT Press.

[7] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction.* 2nd ed. The MIT Press.

[8] "What are Neural Networks?", IBM, Available online: https://www.ibm.com/uk-en/cloud/learn/neural-networks

[9] "The Concept of Artificial Neurons (Perceptrons) in Neural Networks", Available online: https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc

[10] S. Syed-Abdul, X. Zhu, L. Fernandez-Luque, *Digital Health: Mobile and Wearable Devices for Participatory Health Applications.* Elsevier.

[11] E. De Giovanni, F. Forooghifar, G. Surrel, T. Teijeiro, M. Peon, A. Aminifar, D. A. Alonso, "Intelligent Edge Biomedical Sensors in the Internet of Things (IoT) Era." *Emerging Computing: From Devices to Systems*, pp 407–433, 2022.

[12] D. Sopic, E. De Giovanni, A. Aminifar, D. Atienza, "Hierarchical Cardiac-Rhythm Classification Based on Electrocardiogram Morphology." *2017 Computing in Cardiology (CinC)*.

[13] E. De Giovanni, A. Aminifar, A. Luca, S. Yazdani, Jean-Marc Vesin, D. Atienza, "A Patient-Specific Methodology for Prediction of Paroxysmal Atrial Fibrillation Onset." *2017 Computing in Cardiology (CinC)*.

[14] E. De Giovanni, A. A. ValdÉs, M. PeÓn-QuirÓs, A. Aminifar, D. Atienza, "Real-Time Personalized Atrial Fibrillation Prediction on Multi-Core Wearable Sensors." *IEEE Transactions on Emerging Topics in Computing*, Volume: 9, Issue: 4.

[15] G. Surrel, T. Teijeiro, A. Aminifar, D. Atienza, M. Chevrier, "Event-Triggered Sensing for High-Quality and Low-Power Cardiovascular Monitoring Systems." *IEEE Design and Test*, Volume: 37, Issue: 5.

[16] Eduardo José da S. Luz, W. R. Schwartz, G. Cámara-Chávez, D. Menotti, "ECG-based heartbeat classification for arrhythmia detection: a survey." *Computer Methods and Programs in Biomedicine*, Volume 127, April 2016, Elsevier.

[17] A. Craik, Y. He, Jose L Contreras-Vidal, "Deep learning for electroencephalogram (EEG) classification tasks: a review." *Journal of Neural Engineering*, 16, April 2019.

[18] P. R. Chai, R. K. Rosen, E. W. Boyer, "Ingestible Biosensors for Real-Time Medical Adherence Monitoring: MyTMed." *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2016.

[19] G Surrel, A Aminifar, F Rincon, S Murali, D Atienza, "Online Obstructive Sleep Apnea Detection on Medical Wearable Sensors." *IEEE Transactions on Biomedical Circuits and Systems (TBioCAS)*, 2018.

[20] D. Sopic, A. Aminifar, D. Atienza, "e-Glass: A Wearable System for Real-Time Detection of Epileptic Seizures." *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018.

[21] F. Forooghifar, A. Aminifar, D. Atienza, "Resource-Aware Distributed Epilepsy Monitoring Using Self-Awareness From Edge to Cloud." *IEEE Transactions on Biomedical Circuits and Systems*, Volume: 13, Issue: 6, 2019.

[22] F. Forooghifar, A. Aminifar, D. Atienza, "Self-Aware Wearable Systems in Epileptic Seizure Detection." *21st Euromicro Conference on Digital System Design (DSD)*, 2018.

[23] S. Baghersalimi, T. Teijeiro, D. Atienza, A. Aminifar, "Personalized Real-Time Federated Learning for Epileptic Seizure Detection." *IEEE Journal of Biomedical and Health Informatics*, Volume: 26, Issue: 2, 2022.

[24] D. Pascual, A. Amirshahi, A. Aminifar, D. Atienza, P. Ryvlin, R. Wattenhofer, "EpilepsyGAN: Synthetic Epileptic Brain Activities with Privacy Preservation." *IEEE Transactions on Biomedical Engineering (TBME)*, 2020.

[25] R. Zanetti, A. Aminifar, D. Atienza, "Robust Epileptic Seizure Detection on Wearable Systems with Reduced False-Alarm Rate." *42nd annual international conference of the IEEE engineering in medicine and biology society (EMBC)*, 2020.

[26] V. Montesinos, F. Dell'Agnola, A. Arza, A. Aminifar, D. Atienza, "Multi-Modal Acute Stress Recognition Using Off-the-Shelf Wearable Devices." *41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2019.

[27] , "Real-Time EEG-Based Cognitive Workload Monitoring on Wearable Devices." *IEEE Transactions on Biomedical Engineering*, 2021.

[28] D. Sopic, Amin Aminifar, Amir Aminifar, D. Atienza, "Real-Time Event-Driven Classification Technique for Early Detection and Prevention of Myocardial Infarction on Wearable Systems." *IEEE transactions on biomedical circuits and systems*, 2018, Volume 12, Issue 5.

[29] P. Dayan, L. F. Abbot, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems.* The MIT Press.

[30] Z. Huang, H. G Khaled, M. Kirschmann, S. M. H. Gobes, R. H. R. Hahnloser, 2018, "Excitatory and inhibitory synapse reorganization immediately after critical sensory experience in a vocal learner." *eLife 7:e37571.*

[31] S. Sharma, G. Kumar, D. K. Mishra, D. Mohapatra, "Design and Implementation of a Variable Gain Amplifier for Biomedical Signal Acquisition." *International Journal of Advanced Research in Computer Science and Software Engineering*, Volume 2, Issue 2, February 2012.

[32] Available online: https://www.nagwa.com/en/explainers/494102341945/

[33] "The differences between Artificial and Biological Neural Networks", written by R. Nagyfi, Available online: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

[34] "Best GPU for AI/ML, deep learning, data science." Available online: https://bizon-tech.com/blog/

[35] D. S. Jeong, "Tutorial: Neuromorphic spiking neural networks for temporal learning." *Journal of Applied Physics*, 124, 152002, 2018.

[36] E. M. Izhikevich, "Simple model of spiking neurons." *IEEE Transactions on Neural Networks*, Volume: 14, Issue: 6, November 2003.

[37] R. FitzHugh, "Impulses and Physiological States in Theoretical Models of Nerve Membrane." *Biophysical Journal*, Volume 1, Issue 6, July 1961.

[38] J. Nagumo, S. Arimoto, S. Yoshizawa, "An Active Pulse Transmission Line Simulating Nerve Axon." *Proceedings of the IRE*, Volume: 50, Issue: 10, October 1962.

[39] D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning representations by back-propagating errors." *Nature*, 1986.

[40] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, W. D. Lu, "Training Spiking Neural Networks Using Lessons From Deep Learning." *arXiv:2109.12894*.

[41] M. Pfeiffer, T. Pfeil, "Deep Learning With Spiking Neurons: Opportunities and Challenges." *Frontiers in Neuroscience*, Volume: 12, 2018.

[42] S. Lu, A. Sengupta, "Exploring the Connection Between Binary and Spiking Neural Networks." *Frontiers in Neuroscience*, June 2020.

[43] S. M. Bohte, J. N. Kok, H. La Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons." *Neurocomputing*, Volume: 48, Issues: 1–4, 2002.

[44] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, K. Roy, "Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures." *Frontiers in Neuroscience*, Volume: 14, 2020.

[45] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons." *arXiv:1803.09574*, 2018.

[46] G. Q. Bi, M. M. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type." *Journal of Neuroscience*, 1998.

[47] S. Song, K. D. Miller, L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity." *Nature Neuroscience*, 2000.

[48] W. Zhang, B. Gao, J. Tang, P. Yao, S. Yu, M. F. Chang, H. J. Yoo, H. Qian, H. Wu, "Neuro-inspired computing chips." *Nature Electronics* , 3, 371–382, 2020.

[49] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, B. Kay, "Opportunities for neuromorphic computing algorithms and applications." *Nature Computational Science*, Volume 2, 10–19, 2022.

[50] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, K. Boahen, "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations." *Proceedings of the IEEE*, Volume: 102, Issue: 5, May 2014.

[51] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface." *Science*, Volume: 345, Issue 6197, August 2014.

[52] S. Moradi, N. Qiao, F. Stefanini, G. Indiveri, "A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)." *IEEE Transactions on Biomedical Circuits and Systems*, Volume: 12, Issue: 1, February 2018.

[53] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, "SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation." *IEEE Journal of Solid-State Circuits*, Volume: 48, Issue: 8, August 2013.

[54] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, H. Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning." *IEEE Micro*, Volume: 38, Issue: 1, January-February 2018.

[55] "Nengo", Available online: https://www.nengo.ai/

[56] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, C. Eliasmith, "Nengo: a Python tool for building large-scale functional brain models." *Frontiers in Neuroinformatics*, Volume 7, January 2014.

[57] "Brian2", Available online: https://brian2.readthedocs.io/en/stable/

[58] "snnTorch", Available online: https://snntorch.readthedocs.io/en/latest/

[59] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann, R. Kozma, "BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python." *Frontiers in Neuroinformatics*, Volume 12, December 2018.

[60] M. Mozafari, M Ganjtabesh, A. Nowzari-Dalini, T. Masquelier, "SpykeTorch: Efficient Simulation of Convolutional Spiking Neural Networks With at Most One Spike per Neuron." *Frontiers in Neuroscience*, Section Neuromorphic Engineering, Volume 13, July 2019.

[61] J. Vitay, H. Ü. Dinkelbach, F. H. Hamker, "ANNarchy: a code generation approach to neural simulations on parallel hardware." *Frontiers in Neuroinformatics*, Volume 9, July 2015.

[62] C. Eliasmith, C. H. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems.* The MIT Press, 2004.

[63] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, D. Rasmussen, "A Large-Scale Model of the Functioning Brain." *Science*, Volume 338, Issue 6111, November 2012.

[64] Feng-Xuan Choo, "Spaun 2.0: Extending the World's Largest Functional Brain Model." UWSpace. Available online: http://hdl.handle.net/10012/13308

[65] M. Stimberg, R. Brette, D. FM Goodman, "Brian 2, an intuitive and efficient neural simulator." *eLife*, 2019.

[66] "Electrocardiogram", Johns Hopkins Medicine, Available online: https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/electrocardiogram

[67] G. B. Moody, R. G. Mark, "The impact of the MIT-BIH Arrhythmia Database", *IEEE Engineering in Medicine and Biology Magazine.* Volume 20, Issue 3, May-June 2001.

[68] A. Goldberger, L. Amaral, L. Glass, J. Hausdorff, P. C. Ivanov, R. Mark, J. E. Mietus, G. B. Moody, C. K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals." Circulation [Online]. 101 (23), pp. e215–e220. (2000).

[69] Available online: https://physionet.org/content/mitdb/1.0.0/

[70] S. Jayaraman, V. Sangareddi, R. Periyasamy, J. Joseph, "Modified limb lead ECG system effects on electrocardiographic wave amplitudes and frontal plane axis in sinus rhythm subjects." *The Anatolian Journal of Cardiology*, January 2017.

[71] A. for the Advancement of Medical Instrumentation et al., "Testing and reporting performance results of cardiac rhythm and st segment measurement algorithms. american national standards institute, inc.(ansi)," Inc.(ANSI), ANSI/AAMI/ISO EC57, 2008.

[72] "Electroencephalogram (EEG)", Johns Hopkins Medicine, Available online: https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/electroencephalogram-eeg

[73] A. Shoeb, "Application of Machine Learning to Epileptic Seizure Onset Detection and Treatment." PhD Thesis, Massachusetts Institute of Technology, September 2009.

[74] Available online: https://physionet.org/content/chbmit/1.0.0/

[75] X. Hou, Y. Liu, O. Sourina, W. Mueller-Wittig, "CogniMeter: EEG-based Emotion, Mental Workload and Stress Visual Monitoring." *2015 International Conference on Cyberworlds (CW).*

[76] "Cardiology Teaching Package: A Beginners Guide to Normal Heart Function, Sinus Rhythm and Common Cardiac Arrhythmias." *The University of Nottingham.* Available online: https://www.nottingham.ac.uk/nursing/practice/resources/cardiology/function/sinus_rythm.php

[77] P. Madona, R. I. Basti, M. M. Zain, "PQRST wave detection on ECG signals." *Gaceta Sanitaria*, Volume 35, 2021.

[78] Y. Feng, S. Geng, J. Chu, Z. Fu, S. Hong, "Building and training a deep spiking neural network for ECG classification." *Elsevier: Biomedical Signal Processing and Control*, Volume 77, August 2022.

[79] G. D. Clifford, C. Liu, B. Moody, H. L. Li-wei, I. Silva, Q. Li, A. E. Johnson, R. G. Mark, "AF Classification from a Short Single Lead ECG Recording: the PhysioNet/Computing in Cardiology Challenge 2017." Computing in Cardiology (CinC), IEEE, 2017.

[80] F. Corradi, S. Pande, J. Stuijt, N. Qiao, S. Schaafsma, G. Indiveri, F. Catthoor, "ECG-based Heartbeat Classification in Neuromorphic Hardware." *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE.

[81] K. Buettner, A. D. George, "Heartbeat Classification with Spiking Neural Networks on the Loihi Neuromorphic Processor." *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI).*

[82] A. Amirshahi, M. Hashemi, "ECG Classification Algorithm Based on STDP and R-STDP Neural Networks for Real-Time Monitoring on Ultra Low-Power Personal Wearable Devices." *IEEE Transactions on Biomedical Circuits and Systems*, Volume 13, Issue 6, December 2019.

[83] S. Kiranyaz, T. Ince, M. Gabbouj, "Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks." *IEEE Transactions on Biomedical Engineering*, Volume: 63, Issue: 3, March 2016.

[84] T. Iakymchuk, A. Rosado-Muñoz, J. F. Guerrero-Martínez, M. Bataller-Mompeán, J. V. Francés-Víllora , "Simplified spiking neural network architecture and STDP learning algorithm applied to image classification." *EURASIP Journal on Image and Video Processing*, 2015.

[85] F. Tian, J. Yang, S. Zhao, M. Sawan, "A New Neuromorphic Computing Approach for Epileptic Seizure Prediction." *2021 IEEE International Symposium on Circuits and Systems (ISCAS).*

[86] Y. Luo, Q. Fu, J. Xie, Y. Qin, G. Wu, J. Liu, F. Jiang, Y. Cao, X. Ding, "EEG-Based Emotion Classification Using Spiking Neural Networks." *IEEE Access*, Volume: 8, March 2020.

[87] S. Koelstra, C. Muehl, M. Soleymani, J.-S. Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, I. Patras, "DEAP: A Database for Emotion Analysis using Physiological Signals." *EEE Transactions on Affective Computing*, Volume: 3, 2012.

[88] https://bcmi.sjtu.edu.cn/home/seed/

[89] A. Das, F. Catthoor, S. Schaafsma, "Heartbeat Classification in Wearables Using Multi-layer Perceptron and Time-Frequency Joint Distribution of ECG." *2018 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE).*

[90] "Converting a Keras model to a spiking neural network." Available online: https://www.nengo.ai/nengo-dl/examples/keras-to-snn.html

[91] Y. Xing, L. Zhang, Z. Hou, X. Li, Y. Shi, Y. Yuan, F. Zhang, S. Liang, Z. Li, L. Yan, "Accurate ECG Classification Based on Spiking Neural Network and Attentional Mechanism for Real-Time Implementation on Personal Portable Devices." *Electronics*, 2022.

[92] B. Yin, F. Corradi, S. M. Bohte, "Accurate online training of dynamical spiking neural networks through Forward Propagation Through Time."

Printed by Tryckeriet i E-huset, Lund 2023

LUND
UNIVERSITY

Series of Master´s theses
Department of Electrical and Information Technology
LU/LTH-EIT 2023-915
http://www.eit.lth.se