Automated detection of mouse scratching behaviour using convolutional spiking neural network

Master's thesis in Complex Adaptive Systems

Kelly Ma

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Automated detection of mouse scratching behaviour using convolutional spiking neural network

Kelly Ma

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2023

Automated detection of mouse scratching behaviour using convolutional spiking neural network
Kelly Ma

© Kelly Ma, 2023.

Supervisor: HuanYu Xiang, Guangdong Institute of Intelligence Science and Technology, China
Examiner: Giovanni Volpe, Department of physics, Chalmers university of Technology, Sweden

Master's Thesis 2023
Department of Physics

Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Automated detection of mouse scratching behaviour using convolutional spiking neural network

Kelly Ma
Department of physics
Chalmers University of Technology

Abstract

In recent years, significant progress has been achieved in the field of artificial neural networks (ANNs). However, their performance still falls short when compared to their biological counterparts. As a result, there is growing interest in Spiking neural networks (SNNs), which are known for their biological plausibility and efficiency, aiming to combine the advantages of both existing ANNs and biological networks. Therefore, this project focuses on the development of an SNN that can reliably identify mouse scratching behaviour. The test animals were recorded using an event camera, and after the data was preprocessed and converted into frames, a data set was created. Among the different training methods tested, the SNN trained with backpropagation combined with the gradient surrogate method yielded the best result, achieving a final accuracy of approximately 97% on the test dataset. However, it should also be noted that the preprocessing process actually has a significant impact on overall performance. Taking this into consideration, the actual accuracy would be around 83%. This result obtained indicate that a SNN is capable of detecting scratching behaviour despite the limitations imposed by the small size of the available data, thus showcasing the potential of SNNs in real-life applications.

Keywords: ANN, SNN, dynamic behaviour detection

Acknowledgements

I would like to express my gratitude to my supervisor HuanYu Xiang and examiner Giovanni Volpe, for their guidance and valuable insight in this project, and also for providing the resources required to carry out this project.

Kelly Ma, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
ANN	Artificial Neural Network
BP	Backpropagation
BPTT	Backpropagation Through Time
GPU	Graphics Processing Unit
LIF	Leaky Integrate-and-Fire
NN	Neural Network
RNN	Recurrent Neural Network
SNN	Spiking Neural Network
STDP	Spike timing dependent plasticity

Contents

1
Introduction

In recent years, significant progress has been achieved through the utilization of various types of artificial neural networks (ANNs). In domains such as image recognition, game playing, and style imitation, artificial intelligence (AI) has already surpassed human performance. However, while general artificial intelligence remains speculative, ANNs still lag in terms of flexibility and efficiency. As AI is expected to continue its growth and expansion, the total energy consumption of ANNs is expected to increase in the coming years. Consequently, it is highly relevant to investigate and develop alternatives with higher flexibility and efficiency, namely spiking neural networks (SNN). Besides their flexibility and efficiency benefits, SNNs are also known for their biological plausibility, which many consider essential for achieving brain-like performance. Therefore, SNNs are sometimes referred to as the third-generation ANNs, with the expectation of combining the advantages of both existing ANNs and biological neural networks.

As implied by its name, the SNN is spike orientated. Instead of updating activation values for each neuron at every time step, information is transferred asynchronously through discrete voltage spikes, similar to how real neurons operate. Resembling recurrent neural networks(RNNs), each neuron in the SNN is associated with a state variable known as the membrane potential, and the detailed update rule for the membrane potential is determined by a biologically plausible neuron model, which also determines the timing to spike. Since energy is consumed only when a spike occurs, SNNs are significantly more energy efficient than conventional ANNs. It has for example been demonstrated that a spiking recurrent neural network can achieve over 100 times improvement in energy efficiency compared to classical RNNs while maintaining state-of-the-art accuracy [1].

However, thus far, applications of SNNs have generally been limited to established datasets and have seldom been employed in real-life scenarios. Therefore, this project aimed to develop an SNN to address the real-life problem of detecting mouse scratching behaviour. The motivation behind this project arises from the practical implications of achieving dependable scratching behaviour detection, along with the aspiration to demonstrate the utility of SNNs.

2
Background

This section presents the relevant background that motivated this project, as well as provides a brief overview of related works conducted by others.

2.1 SNN Frameworks

To prevent reinventing the wheel, the project was built upon existing SNN frameworks. Thus here follows a brief review of some popular SNN frameworks, including both neuroscience-orientated frameworks and machine learning-orientated frameworks.

2.1.1 Neuroscience orientated frameworks

There has been a long-standing interest in simulating biological networks, leading to the development of NEST, which started as early as 1994 [2]. At that time, the only available SNN frameworks were NEURON and GENESIS, which focused on multi-compartment neural models instead of how connected neurons behave as a network. NEST was one of the first frameworks to focus on accurately simulating a large neural network while still allowing inspection and modification of the state of each neuron during a simulation. Since its launch, NEST has been updated continuously and is still considered one of the best SNN frameworks by a large developer community.

When Brian (later updated to Brian2) was created, established frameworks already offered a wide selection of standard neuron and synapse models. However, research involving original models unknown by other still require a framework with even higher flexibility. Brian distinguishes itself from other frameworks by requiring all neural models and connectivity patterns to be explicitly defined with a set of equations [3]. The developers of Brian argued that requiring explicit definitions would eliminate the risk that users misunderstand the implemented models, but also that fundamental concepts and models are still actively investigated and have not yet settled to the point where they can be standardized.

Toward the end of the 20th century, the problem was no longer that too few frameworks existed, but rather there were too many of them, all expressing and implementing the same concepts differently. It significantly impedes communication between researchers and makes it harder to reproduce and build on the work of

others. The design goal of PyNN was, therefore, to build a high-level framework capable of directly running the script on any backend framework supported by PyNN without modification, for example, Brian, NEST, and NEURON [4]. Depending on the circumstances, the user can choose the most appropriate framework or run the script on multiple frameworks to cross-check the result.

2.1.2 Machine learning orientated frameworks

The previously mentioned SNN frameworks support a wide range of neural functionality, software abstraction levels, and hardware devices. However, they were primarily designed for neuroscience purposes rather than targeting machine learning purposes. To bridge this gap and catch up with recent advancements in machine learning, several machine learning-oriented SNN frameworks have emerged in recent years, including SpikingJelly, snnTorch, and Bindsnet [5] [6] [7]. These frameworks typically build upon and extend popular ANN frameworks like PyTorch or TensorFlow to take advantage of GPU-accelerated tensor computation. Moreover, they often adopt a similar syntax to the frameworks they are built upon, making them more user-friendly for individuals familiar with those existing frameworks.

It is worth noting that, compared to neuroscience-oriented frameworks, these machine learning-oriented SNN frameworks tend to sacrifice some degree of biological realism. This trade-off arises from the computational complexity of simulating very detailed SNN and the desire to avoid introducing additional hyperparameters. While they may not fully capture the intricacies of biological neural networks, these machine learning-oriented frameworks offer increased computational efficiency and ease of use, making them well-suited for machine learning applications.

For this project, the chosen framework is SpikingJelly, an open-source framework that has been actively developed since 2020. Although it is still under development, it is still one of the most mature and established options among machine learning-oriented SNN frameworks. SpikingJelly offers several notable advantages. One of its key strengths is its extensive support for a wide range of standard neural models and learning rules. This includes the inclusion of both unsupervised spike-timing-dependent plasticity learning and supervised backpropagation, which are fundamental learning mechanisms in spiking neural networks.

2.2 Scratch detection

Since the primary goal of the project was to develop a system capable of automatically detecting mouse scratching behavior, a brief review of related works and the reasons for the significance of this topic is presented.

2.2.1 Scratching Behavior

Scratching is a reflexive behavior that humans and many animals exhibit when they experience itching sensations on their skin or fur. Evolutionary, It has served creatures as a protective mechanism that allows them to detect harmful substances invading the skin and remove them by scratching. The understanding of scratching behavior holds significant importance for various reasons. In terms of etiology, scratching behavior is frequently used as an indicator in itch studies, proving insights into animals' social and environmental interactions. In the field of medicine, studying scratching behavior as a symptom allows researchers to gain insight into the underlying mechanisms of disease and conditions, leading to the development of more effective treatments. Furthermore, studying scratching behavior also contributes to the understanding of the neural and physiological mechanisms that govern the sensation of itching and the reflexive responses that follow. This can help researchers better understand how the nervous system processes sensory information and how it controls movement.

The scratching behavior of mice is specifically studied due to their extensive use as an animal model in scientific research. This is mostly due to they share many genetic, anatomical, and physiological similarities with humans, but also because they are relatively easy to handle and breed in the laboratory. For mice, scratch is formally defined as a rapid, repetitive, and back-and-forth movement of the hind limb toward the skin.

2.2.2 Traditional methods

In various research fields, researchers very often quantify mouse scratching behavior to assess itch intensity. Thus, various researchers have attempted to develop a quantitative and reliable method for evaluating scratching behavior. For example, Elliott et al. suggested that the scratching behavior could be identified by the vibration and sound generated when a mouse scratches its skin [8]. Inagaki et al. suggested inserting a small magnet into the paw of the test animal, which is placed in an observation chamber surrounded by a round coil. The movement of magnets attached to the paw would therefore induce an electric current in the coil, which is then amplified and analyzed [9]. However, these traditional methods are often limited by their invasive nature and the potentially distressing handling of animals, as well as the costly specialized equipment.

2.2.3 Machine learning-based methods

More modern methods based on combining computer vision with machine learning have also been developed to evaluate scratching behavior. Commonly, the test animal's movement is recorded using a camera with a high frame rate, then preprocessed using the subtraction method to reduce the data size and cancel out background noise. After that, the obtained frame sequences can be classified using some ma-

chine learning algorithm, such as a decision tree or a neural network [10] [11] [12] [13]. Performance-wise, the convolutional recurrent network(CRNN) is considered the best option. Convolutional layers are used to extract features from the inputted frame sequences, which are then inputted into recurrent layers to capture the temporal dynamics of the behavior.

3
Theory

This section will provide the relevant concepts and theories for this project.

3.1 Artificial neural network

The concept of the Artificial Neural Network (ANN) was inspired by the structure and function of biological neural networks in the brain. Back in the 1920s, neuroscientist MuCulloch and logician Walter Pitts introduced the first mathematical model of a biological neuron, known as the McCulloch-Pitts neuron. However, the significant breakthrough of artificial neural networks occurred almost a hundred years later, thanks to the invention of the layer-wise backpropagation(BP) algorithm and advancements in computational power. The invention of Graphics Processing Units (GPUs) played a crucial role in this advancement. While initially designed to accelerate the rendering of 3D graphics, their ability to perform multiple computations simultaneously in parallel greatly accelerated machine learning operations.

Nowadays, most ANNs utilize artificial neurons as their elementary units. As shown in Figure 3.1, the dendrites in the biological neuron which receive incoming signals are represented as a weighted sum. This weighted sum is then passed through a non-linear activation function, mimicking what the soma(cell body) does in a biological neuron. The resulting output is known as the activation value, which represents the neuron's action potential transmitted along its axon.

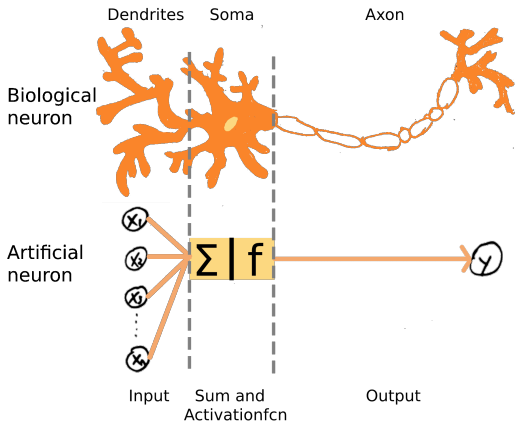

Figure 3.1: *Comparison of a simplified biological neuron and the artificial neuron model.*

3.2 Spiking neuron network

Although originally inspired by their biological counterpart, ANNs are far from being biologically plausible, as it is highly unlikely that information in the neural system is represented by continuous activation values. Consequently, spiking neural models have been introduced to more closely emulate the behaviour of biological neurons. Neural networks that are built upon these spiking neural models are therefore referred to as Spiking Neural Networks (SNNs).

The behaviour of a spiking neuron is typically governed by a set of differential equations, which describe the neuron's dynamics and determine when it generates a spike. The level of complexity and biological plausibility varies among different spiking neural models and may incorporate factors such as ion channels, synaptic connections, and refractory periods. The spikes produced by these neural models mimic the action potentials of biological neurons and are characterized by brief and rapid changes in the neuron's membrane potential. The spiking nature of SNNs allows them to encode information through the timing and frequency of spikes, enabling them to process data in a more event-driven manner, similar to biological neurons. Table 3.1 compares key characteristics of ANN and SNN, including the type of activation value, signal propagation/learning rule used, and the level of biological plausibility.

	ANN	**SNN**
Activation	Continuous value	Binary spike
Signal Propagation	Layer wise	Asynchronous
Learning Models	Backpropagation	STDP[†]
Biological Plausibility	Low	High

[†]Spiket Timing Dependent Plasticity

Table 3.1: The comparison of key characteristics of ANNs and SNNs

3.3 Leaky integrate and fire neuron model

The leaky integrate and fire (LIF) neuron model is the most commonly utilized model among neuron scientists. It is rather straightforward yet biologically plausible. In the LIF model, basic principles of electrical circuits are used to simulate a biological neuron, as shown in Figure 3.2. Specifically, the biophysical characterization of the neuron membrane is modeled as a resistor parallel coupled with a capacitor, with the voltage across the circuit representing the membrane potential of the neuron, and the input current representing the spikes received from connected neurons

The membrane capacitance represents the neuron membrane's ability to store charge and determines the speed at which the membrane potential changes in response to

incoming currents. The resistance represents the tendency of the charge stored to slowly leak through the cell membrane over time as through a finite leak resistor and determines the rate at which the membrane potential decays.

The external stimuli from either connected neurons or the environment are modeled as current flowing into the neuron, contributing to the accumulated potential until it exceeds a predefined threshold u_{th}. Then, an action potential is fired, and the capacitor discharges to its resting potential u_{rest} to start the process over again.

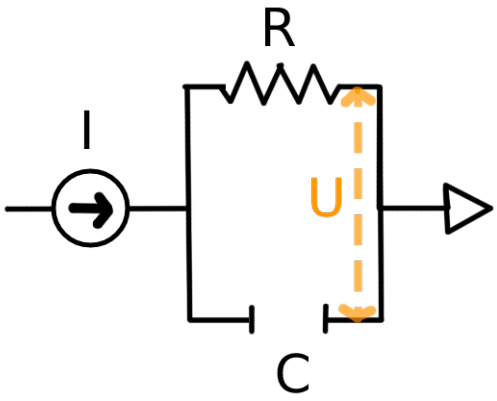

Figure 3.2: *The LIF neuron model simulates a biological neuron as a parallel RC circuit, where the voltage represents the membrane potential and the input current represents the spikes received from connected neurons.*

Following the current law, the electric circuit shown in Figure 3.2 can be mathematically defined by the following equations.

$$I(t) = I_R + I_C = \frac{U(t) - U_{rest}}{R} + C\frac{dU}{dt} \tag{3.1}$$

$$RC\frac{U(t)}{dt} = -(U(t) - U_{rest}) + RI(t) \tag{3.2}$$

Introduce membrane time constant $\tau_m = RC$

$$\tau_m\frac{dU}{dt} = -(U(t) - U_{rest}) + RI(t) \tag{3.3}$$

A common way of implementing Equation 3.3 is to solve it numerically using Euler's method, yielding Equation 3.4, which is also used in this project.

$$U(t + \Delta t) = U(t) - \frac{\Delta t}{\tau}(U(t) - U_{rest})) + \frac{\Delta t}{\tau}RI(t) \tag{3.4}$$

Consider the spiking mechanism, the whole process can be summarized into equations 3.5, 3.6, and 3.7. Firstly charge the potential with inputted spikes from connected neurons, and decay the potential. Secondly, determine whether to spike with

a Heaviside function taking the current potential and the threshold as inputs. Lastly, the potential is reset if any spike had been generated, otherwise, nothing happens.

$$H(t+\Delta t) = U(t) - \frac{\Delta t}{\tau}(U(t) - U_{rest})) + \frac{\Delta t}{\tau} RI(t) \tag{3.5}$$

$$S(t+\Delta t) = \Theta(H(t+\Delta t) - U_{thr}) \tag{3.6}$$

$$U(t+\Delta t) = (1 - S(t+\Delta t))H(t+\Delta t) + S(t)U_{rest} \tag{3.7}$$

However, as the field of spiking neural networks is still evolving, the implementation method of Equation 3.3 in SNNs has not yet established conventions. Although using numerical methods are the mainstream way, other implements based on the analytical homogeneous solution also exist.

3.4 Backpropagation through time

Backpropagation(BP) combined with gradient descent is the most widely used algorithm for training neural networks. Consequently, it is reasonable to investigate whether this methodology can be utilized in SNNs. Similar to conventional ANNs, the gradient is computed by iteratively applying the chain rule as shown in Equation 3.8.

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial S} \underbrace{\frac{\partial S}{\partial U}}_{\{0,\infty\}} \frac{\partial U}{\partial I} \frac{\partial I}{\partial W} \tag{3.8}$$

However, Equation 3.8 only calculates the gradient for one single time step, which is insufficient for SNNs to learn temporal dependency. The backpropagation through time(BPTT) is therefore presented, which summarizes the influence of the weight on present and historical losses together to define the global gradient.

The gradients of the loss function with respect to the intra-layer weights can be expressed using Equation 3.9, where L represents the loss function, S represents the spiking, and U represents the membrane potential. The constraint $s < t$ was added to ensure that only the contributions of immediate and prior influences of W on the loss are considered. The term $\frac{\partial \mathcal{L}[t]}{\partial W}$ vanishes because the weights are constrained to be shared across all steps, which means $W[0] = W[1] = \ldots = W$ and therefore $\frac{\partial W[s]}{\partial W} = 1$.

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_t \frac{\partial \mathcal{L}[t]}{\partial W} = \sum_t \sum_{s \leq t} \frac{\partial \mathcal{L}[t]}{\partial W[s]} \frac{\partial W[s]}{\partial W} = \sum_t \sum_{s \leq t} \frac{\partial \mathcal{L}[t]}{\partial W[s]} \tag{3.9}$$

Equation 3.10 showcases the gradient computation for the case $s = t - 1$, demonstrating how the backward pass can be computed recurrently by tracking back in time.

$$\frac{\partial \mathcal{L}[t]}{\partial W[t-1]} = \frac{\partial \mathcal{L}[t]}{\partial S[t]} \underbrace{\frac{\partial S[t]}{\partial U[t]}}_{\{0,\infty\}} \frac{\partial U[t]}{\partial U[t-1]} \frac{\partial U[t-1]}{\partial I[t-1]} \frac{\partial I[t-1]}{\partial W[t-1]} \tag{3.10}$$

3.5 Surrogate Gradient Method

So far, the BPTT is very similar to the backpropagation for other neural networks with recurrent structures. However, the use of spiking neurons introduces an additional challenge. While it is generally safe to assume that the task-specific loss function is differentiable, spiking neuron models may incorporate non-differentiable functions or functions whose differentials are unsuitable for further computation. For example, in the spike generation process of the LIF neuron model, the Heaviside function is involved. Since its differential the Delta function ranges from zero to infinity, some adjustments clearly must be made before computations can be carried out.

Therefore, the surrogate gradient method is introduced. While the neuron model remains unchanged in the forward pass, the derivative is replaced by the derivative of a differentiable surrogate function during the backward pass. In most cases, the surrogate function is a continuous and smooth function whose shape is similar to the original function.

However, although the surrogate gradient method enables the training of networks with non-differentiable activations, it can introduce certain challenges and potential performance drops. This may explain why SNNs typically exhibit lower accuracy compared to ANNs.

3.6 Spike Timing-Dependent Plasticity

Spike Timing-Dependent Plasticity (STDP) is a type of synaptic plasticity that governs how the strength of a synaptic connection between two neurons changes. Its effectiveness has been experimentally demonstrated in biological neural networks across various species, and it is believed to play a crucial role in the processes of learning and memory in the brain.

According to the Hebbian rule, the strength of a synaptic connection is related to the temporal correlation between the presynaptic and postsynaptic neurons. If the presynaptic neuron consistently fires just before the postsynaptic neuron, the connection is strengthened. Conversely, if the presynaptic neuron consistently fires after the postsynaptic neuron, the connection is weakened. This principle is commonly expressed as "those who fire together, wire together; those who fire out of sync, lose their link." The STDP learning rule operates on the same principle, but also takes into account the precise timing of the spikes generated by the neurons. Equation

3.11 illustrates how the strength of the synaptic between the presynaptic neuron i and postsynaptic neurons j is modified, where $A_+, A_-, \tau_+, \tau_-$are constants, t_i and t_j represent the exact firing timing.

$$\Delta w_{ij} = \begin{cases} A_+ exp(\frac{-|t_i - t_j|}{\tau_+}), & t_i \leq t_j \\ A_- exp(\frac{-|t_i - t_j|}{\tau_+}), & t_i \geq t_j \end{cases} \tag{3.11}$$

3.7 Trace method

A problem with directly applying Equation 3.11 is it would require storing the exact firing timing for each neuron, which is very computationally heavy. To address this issue, an additional state variable trace(tr) is commonly introduced when implementing STDPs, as shown in Equation 3.12. The computation for the trace variable strongly resembles the membrane potential from a LIF neuron, but only depends on its previous state and the spike generated by itself.

$$tr(t) = tr(t-1) - \frac{tr(t-1)}{\tau} + s(t) \tag{3.12}$$

The update of weight therefore become as shown in Equation 3.13, where additional functions F_{post} and F_{pre} can be introduced to further control how weight changes.

$$\Delta w_{ij} = F_{post}(w_{ij}(t)) \cdot tr_i(t) s_j(t) - F_{pre}(w_{ij}(t)) \cdot tr_j(t) s_i(t) \tag{3.13}$$

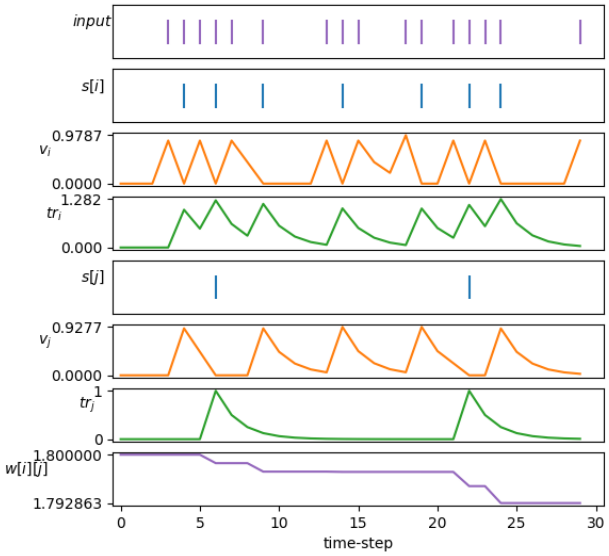

Figure 3.3: *Impact of incoming spikes on potential and trace for two connected neurons i, and j*

Figure 3.3 showcase how the potential and trace are affected by incoming spikes for two connected neurons i, and j. It can be seen that during the absence of an incoming spike, both trace and potential decrease exponentially. When an external spike comes in, both the potential and trace increase, but if this causes the neuron to generate a spike, the potential drop to reset level while the trace still increase.

4
Method

The method section outlines the specific approach and procedures employed in this project, with the purpose to allow readers to understand and replicate the obtained result. It describes the chosen methodology for data collection and preprocessing, how the dataset was created, and the training procedure of the model.

4.1 Data collection

AI approaches are known for their reliance on large amounts of data. The initial step involves capturing data to construct a dataset used for training the model.

To induce scratching behaviour in the mice, the mice were intradermally injected with 10mg/ml chloroquine at the nape, a location that can only be accessed if the mice lift their paw. Chloroquine is primarily a medication used for the prevention and treatment of malaria; however, in this study, its side effect of inducing skin itchiness was utilized. Immediately following the injection, the mice were placed inside an object chamber and recorded with an event camera for approximately 30 minutes, as illustrated in Figure 4.1. A total of three male mice from the C57BL/6 strain were used in this experiment, and the total recording duration amounted to approximately 120 minutes.

Figure 4.1: *Photo of an event camera recording a mouse placed in an objection chamber.*

4.1.1 Event camera

The data was captured using an event camera (sometimes also called a Dynamic Vision Sensor) of model DVXplorer, manufactured by iniVation. It has a spatial resolution of 640x480, and a temporal resolution of 65 - 200 µs (equivalent to 5 000-15 385 fps) [14].

An event camera is a bio-inspired sensor designed to emulate how the human eye perceives light, as shown in Figure 4.2. This design allows it to have better energy efficiency and overcomes some major drawbacks of conventional cameras. In conventional cameras, movement is captured frame by frame. However, with a high frame rate, many frames captured within a short time span are nearly identical, differing only in a small fraction of pixels. This results in the capture and storage of unnecessary data. Consequently, while temporal resolutions increase linearly with the frame rate, the storage requirements increase exponentially. The event camera effectively addresses this issue by letting each pixel operate independently and asynchronously, generating an output only when local changes in brightness exceed a predefined threshold. As a result, the event camera can capture contours and motion with exceptional precision and energy efficiency, without consuming extra bandwidth and processing resources. Another benefit of event cameras is being less sensitive to lighting conditions as long as it does not shift suddenly, and can operate even when lighting conditions are poor.

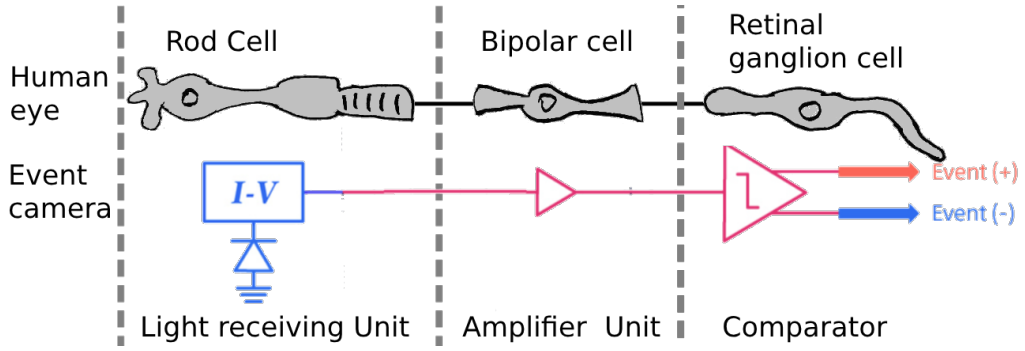

Figure 4.2: *How the human eye and event camera perceive light.*

The generated output is in the format of [timestamp, x coordinate, y coordinate, polarity, 0, 0], with a size of [8 bytes, 2 bytes, 2 bytes, 1 byte, 2 bytes, 1 byte]. Extra zero padding was added at the end to ensure better memory alignment. The timestamp is represented in Unix timestamps, which measures the time passed since 00:00:00 UTC on 1 January 1970 in microseconds. The polarity is represented in binary, with 1 indicating increasing brightness and 0 indicating decreasing brightness.

4.1.2 DV

The event camera is controlled by a computer using iniVation's official software DV, which utilizes four predefined modules, as depicted in Figure 4.3. The process begins

with the capture module, which directly receives raw data from the event camera. Subsequently, the data undergoes denoising via the Khodamoradi algorithm, providing noise reduction. Finally, the processed data is visualized and stored locally as an Aedat file [15].

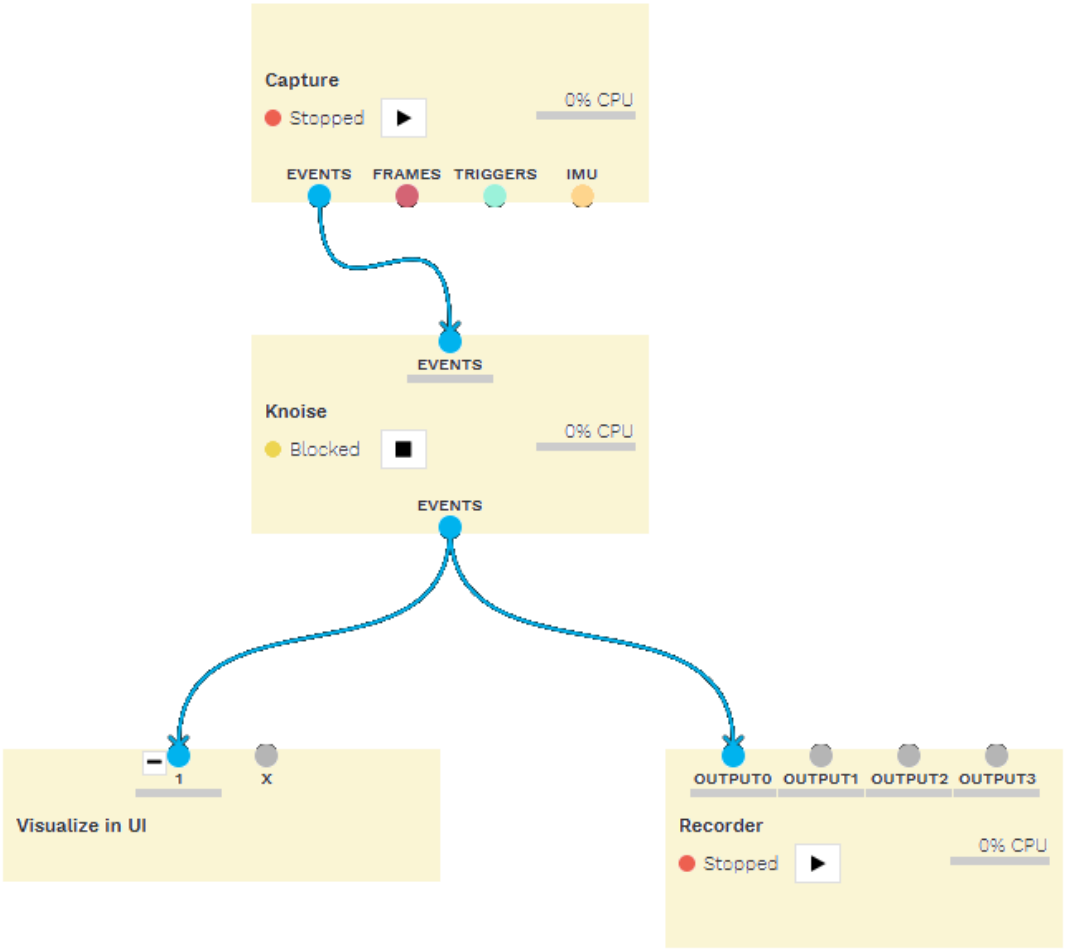

Figure 4.3: *Screen shot of the DV structure used while capturing data.*

4.1.3 Video annotation

As supervised learning was utilized, all data required labeling. Initially, the plan was to also record the electric current induced by magnets attached to the paw of the test animal, as described in Section 2.2.2. The data could then be analyzed to provide the labels for the data captured by the event camera. However, the precise labeling of scratch behaviour, which only lasts for a short duration, required accurate alignment. It turned out to be a great challenge due to the absence of timestamps from the available amperemeter. To further complicate the alignment, it also had some default denoise features resulting in a dynamic sampling rate. Given that the

controlling code was proprietary and kept by the manufacturer, limited options were available to address this challenge. Consequently, all labeling was carried out manually. The recorded event data was visualized using the DV software and displayed in x100 slow-motion. For each observed scratching episode, its start time and end time were annotated.

4.2 Dataset creation

In this section, the process of converting the event streams into frame sequences with corresponding labels to create a dataset is described. Subsequently, the dataset is randomly divided into a training set and a test set, with proportions of 85% and 15% respectively, for further training.

4.2.1 Aedat2intevalList

The primary challenge in detecting scratching or any dynamic behaviour in general lies in determining how to partition the raw data to create a dataset. This task is challenging due to the dynamic nature of the behaviour, which requires the segmented data to form a chronological data sequence. Additionally, the duration of the behaviour is usually short and irregular, which adds to the difficulty of detecting them accurately.

To address this challenge, a statistical analysis based on the amount of data received by the event camera was performed to identify time intervals with a significantly higher probability of corresponding to scratching behaviour. Figure 4.4 illustrates the typical amount of events collected per millisecond while the test animal repeatedly scratches itself. The colored region indicates the observed scratching behaviour, typically lasting between 0.3 and 0.6 seconds. The peaks marked with a cross were identified using the peak finding algorithm provided by the *scipy.signal* package.

It was noted that the observed scratching behaviour typically starts and ends near the peaks. This finding is considered reasonable since the test animal needs to lift its paw to reach the injection site, resulting in a larger movement and a subsequent peak. Similarly, another peak is generated when the test animal restores its paw to the original location after scratching. The slight mismatch between the peaks and the colored regions can be attributed to the manual labeling process, which allowed for a delay caused by human response time.

Following this finding, all time intervals that start and end at two closely adjacent peaks (less than one second apart) were extracted as such intervals are more likely to correspond to scratching behaviour. To assign a label to each interval, its start and end times are compared to the precise timing of all observed scratching behaviour, as illustrated in Algorithm 1. The labeling of the rawdata, referred to as *trueIntervalList*, is a list containing the start and end times of all instances of observed

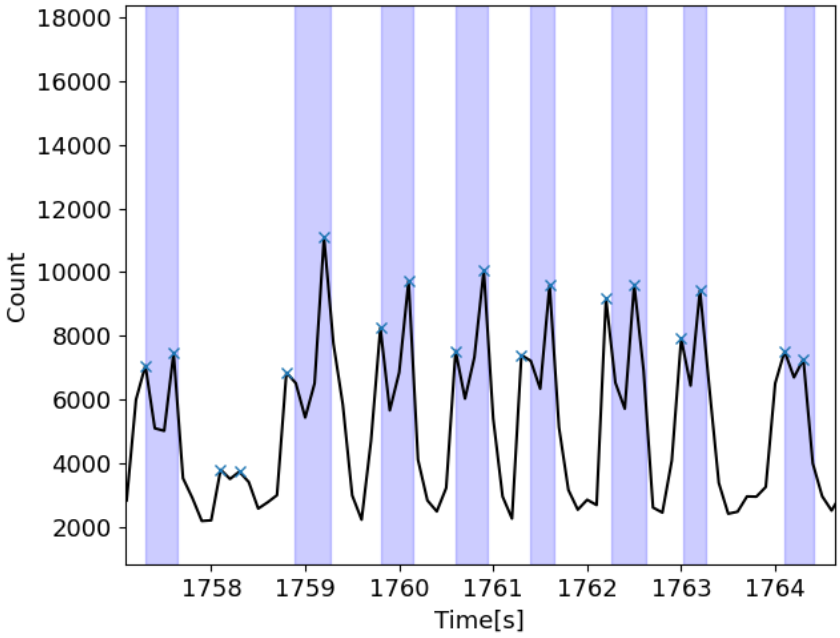

Figure 4.4: *Typical number of events received per millisecond with peaks marked as a cross. The colored regions indicate that scratch behaviour has been observed during this time.*

scratching behaviour, if the total overlap between the inputted interval and all instances in the list is more than 50%, this interval is label as it actually corresponds to scratching behaviour.

Algorithm 1 An algorithm to label the time interval = $[peak_{start}, peak_{end}]$

```
overlap = 0
for [start, end] in trueIntervalList do
    t_start = max(peak_start, start)
    t_end = min(peak_end, end)
    if t_start < t_end then
        overlap+ = (t_end − t_start)/(peak_end − peak_start)
    end if
end for
if overlap<0.5 then
    return 0
else
    return 1
end if
```

4.2.2 Event2frame

The next step was to create a corresponding list of event sequences from the labeled *intervalList*. For each interval in the list, all events that were received during this

specific time interval were extracted and the same label was kept. To utilize mature computer vision techniques and ensure a constant input dimension, all event sequences were then converted into two-channel-frames sequences using the event-to-frame integrating method [16].

Given an event sequence denoted as $E(x_i, y_i, t_i, p_i), i \in [1, N]$, represent each event's coordinate, timestamp and polarity, the N events are divided into $T = 20$ slices, with an almost equal number of events in each slice, as shown in Equation 4.1, These slices are then accumulated into sequences of frames, denoted as $F(j, p, x, y), j \in [1, 20]$. It is worth noting that T also is the simulation time-step for the model used, and the frames are finally scaled down from 640x480 to 64x48 to accelerate further training.

$$\begin{aligned} j_l &= \left\lfloor \frac{N}{T} \right\rfloor \cdot j \\ j_r &= \begin{cases} \left\lfloor \frac{N}{T} \right\rfloor \cdot (j+1), & \text{if } \ j < T-1 \\ N, & \text{if } \ j = T-1 \end{cases} \\ F(j, p, x, y) &= \sum_{i=j_l}^{j_r-1} E(p_i, x_i, y_i) \end{aligned} \tag{4.1}$$

4.2.3 Augmentation

Despite the injection of the drug, scratching remains being a sparse behaviour that rarely occurs. Hence, several data augmentation methods were employed to augment the size and diversity of the dataset.

For each frame sequence corresponding to scratching, the following augmentation techniques were applied: firstly, horizontal and vertical flipping. To introduce additional randomness, horizontal and vertical translation were applied, with a randomly selected distance ranging from 0% to 30%. In summary, four additional data instances were created for each original scratching data as shown in Figure 4.5.

4.3 Model Architecture

In terms of architecture, the model can be divided into two main components: an encoder and a classifier. The encoder is responsible for extracting high-dimensional features from the input, while the classifier utilizes these features to predict the corresponding label.

4.3.1 Encoder

The encoder architecture can be seen in Table 4.2, containing the layers used, and the input/output dimensions for each layer. It consists of convolutional layers followed

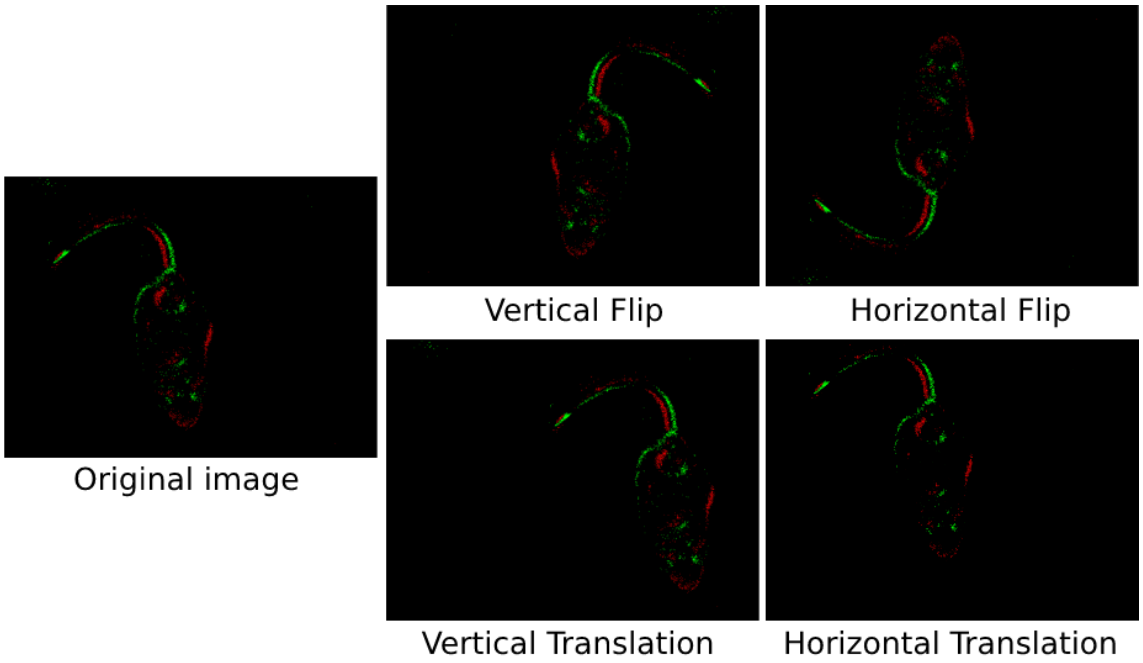

Figure 4.5: *Augmentations methods used for data corresponding to scratching behaviour.*

by Batchnorm, LIF neurons, and max pooling layers, with the layer's hyperparameters found in Table 4.1.

	kernel size	**padding**
Conv2d	3	2
	num_features	
Batchnorm	32	
	kernel_size	**stride**
MaxPool2d	2	2

Table 4.1: Hyperparameters for the encoder's different layers.

	Input shape	**Output size**
Conv2d	20x25x2x64x48	20x25x32x64x48
Batchnorm	20x25x32x64x48	20x25x32x64x48
LIF	20x25x32x64x48	20x25x32x64x48
Maxpool	20x25x32x64x48	20x25x32x32x24
Conv2d	20x25x32x32x24	20x25x32x32x24
Batchnorm	20x25x32x32x24	20x25x32x32x24
LIF	20x25x32x32x24	20x25x32x32x24
Maxpool	20x25x32x32x24	20x25x32x16x12

Table 4.2: Architecture for the encoder, with input and out given for each layer used.

4.3.2 Classifier

The encoder architecture comprises stacked dropout layers followed by Linear and LIF layers. The specific dimensions and relevant hyperparameters can be found in Tables 4.3 and 4.4. Instead of solely relying on the final LIF layer for prediction, an Avgpool layer was incorporated as the final layer. This layer computes the average activation across a group of neurons corresponding to each class, resulting in

a more accurate and robust prediction. Lastly, since the model output includes a time dimension, the final prediction is interpreted using rate encoding. This means that the average activation over time is considered as the final prediction.

	Dropout rate	
Dropout	0.5	
	kernel_size	**stride**
Avgpool1D	50	50

Table 4.3: Hyperparameters for the classifier's different layers.

	Input size	Output size
Flatten	20x25x32x16x12	20x25x6144
Dropout	20x25x6144	20x25x6144
Linear	20x25x6144	20x25x1000
LIF	20x25x1000	20x25x1000
Dropout	20x25x1000	20x25x1000
Linear	20x25x1000	20x25x100
LIF	20x25x100	20x25x100
Avgpool1D	20x25x100	20x25x2

Table 4.4: Architecture for the classfier, with input and out given for each layer used

4.4 Training using BP

The SNN with the suggested architecture was first trained using the surrogate gradient method combined with backpropagation, as described in Section 3.5. The model was optimized with Adam optimizer with a learning rate of 0.001 and ran for 70 epochs.

4.4.1 Surrogate function

For all LIF neurons involved, the arctan surrogate function was used, which is displayed in Figure 4.6 together with its derivative and the Heaviside function.

4.4.2 Loss function

The backpropagation method relies on the fundamental principle of updating the weights to minimize the loss. Therefore, a loss function is necessary to evaluate and determine the performance of the model. The loss function measures the discrepancy between the predicted output of the model and the true target values, allowing the model to learn and adjust its weights accordingly during the backpropagation process.

The loss function used is mathematically described in Equation 4.2, which is a sum between the weighted CrossEntropyLoss and firing rate. The weighted CrossEntropyLoss was applied since the dataset was still strongly unbalanced despite the data augmentation applied. In this equation, y represents the labels encoded using

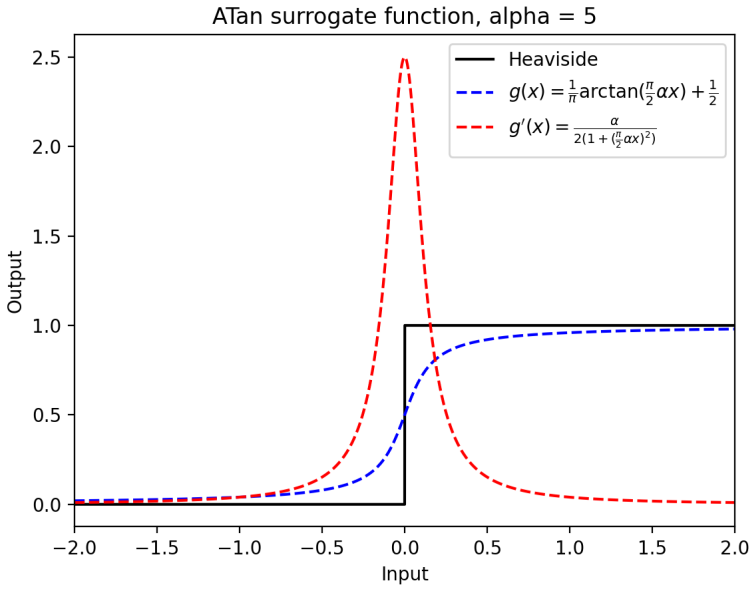

Figure 4.6: *The arctan surrogate function used and its derivative compared with the Heaviside function.*

one-hot encoding, $\hat{y}$ represents the predicted probability distribution of the target classes, w_i represents the weight assigned to class i, and N represents the total number of samples. The weight is determined by the number of samples for the class with most samples divided by the number of samples in each class, which encourages the model to accurately classify the underrepresented class.

The firing rate here is computed as the number of generated spikes divided by the maximum number of spikes that the network theoretically can generate. It is used to determine the efficiency of the model. Without generating spikes, the neuron transfers no information to further layers. Consequently, the only operation needed is to update its own state variable, resulting in nearly no energy consumption. By incorporating the firing rate into the loss function, the model is motivated to minimize its overall firing rate, thereby maximizing the energy efficiency of an SNN.

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} w_i y_i \log(\hat{y}_i) + firingRate \tag{4.2}$$

4.5 Training using BP combined with STDP

Another test was conducted with the aim of combining two well-known learning rules, STDP learning, and BP. All convolutions layers involved were trained with the STDP learning rule following the description in Section 3.6, the Fpost and Fpre functions shown in Equations 4.3 and 4.4, were intentionally designed to be asymmetric so that δW tends to be negative globally. For these layers, the weights were

then updated using the SGD optimizer with a learning rate of 0.01. And all layers were trained with the surrogate gradient method combined with backpropagation as previously section with unchanged loss function, surrogate function, and optimizer.

$$F_{pre}(w) = min(max(x, -0.7), 1) \subseteq [-0,7; 1] \tag{4.3}$$

$$F_{post}(w) = min(max(x, -1), 0.7) \subseteq [-1; 0.7] \tag{4.4}$$

4.6 Model application

Additional functions were implemented to demonstrate how the trained model can be utilized. Once the model has been trained, the user can input a single Aedat file, which will be automatically converted into an unlabeled dataset without any data augmentations. The function then outputs a CSV file containing the predicted timing and duration of all scratching behaviour. If required, it can also output a map of mp4 videos converted from the corresponding frame sequences.

5
Result

The result section presents the key findings of the project. It includes visual representations of the results, discusses the analysis methods used, and provides some insights and implications.

5.1 Training result

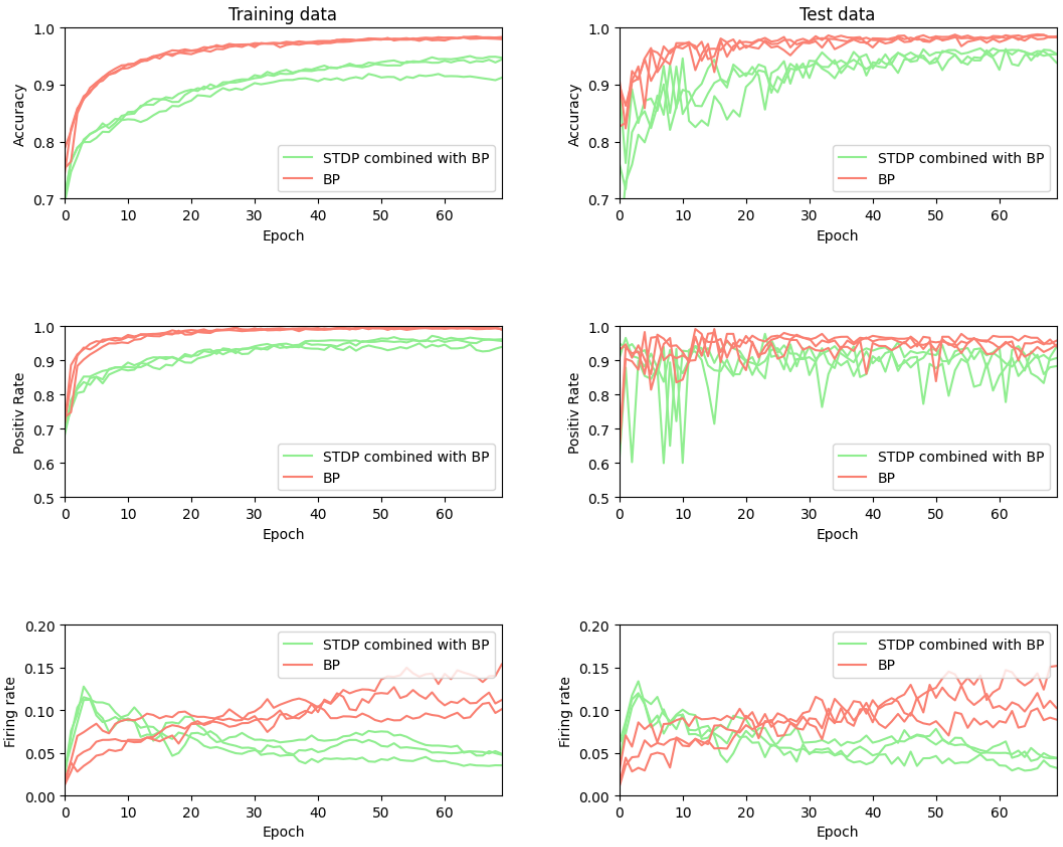

Figure 5.1: *Accuracy, positive rate and firing rate for models trained with BP or BP combined with STDP.*

Training set	Accuracy	Positive rate	Firing rate
BP	98.19%	99.07%	12.21%
STDP combined with BP	93.44%	95.22%	4.39%

Table 5.1: The average Accuracy, Positive rate, and Firing rate on the training set for models trained with BP resp. STDP combined with BP for 70 epochs.

Test set	Accuracy	Positive rate	Firing rate
BP	98.37%	94.75%	11.43%
STDP combined with BP	94.86%	90.92%	4.01%

Table 5.2: The average Accuracy, Positive rate, and Firing rate on the test set for models trained with BP resp. STDP combined with BP for 70 epochs.

To minimize randomness, three models were trained using identical settings for both training methods. Figure 5.1 illustrates the accuracy, positive rate, and firing rate of these models for the training set and test set. The models trained using BP are indicated by the color red, while the models trained using a combination of STDP and BP are depicted in green. The average results of the models trained using both methods, after 70 epochs, are presented in Table 5.1 and 5.2.

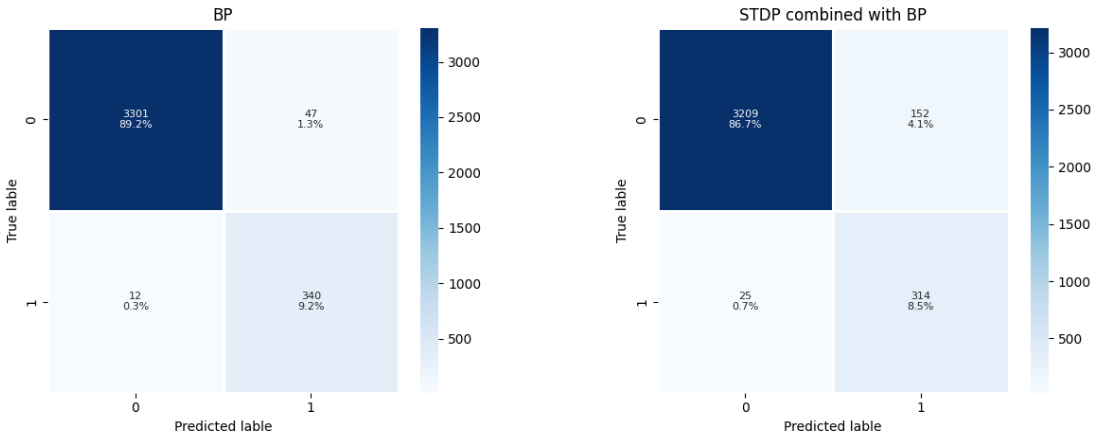

Figure 5.2: *Confusion matrix for models trained with BP, or BP combined with STDP.*

For each training method, the model with the highest average accuracy and the positive rate is saved. Their results are then displayed using a confusion matrix as seen in Figure 5.2 to provide a more intuitive representation.

Since the dataset is unbalanced, evaluating the model's performance solely based on accuracy is insufficient. There is a potential risk that the model might classify all data as the majority class, resulting in a high accuracy but an ineffective model. Therefore, the positive rate, which reflects the model's ability to accurately label positive data, is also noted.

Across all training methods, the results on the test set exhibit more fluctuation compared to the training set, but some degree of convergence can still be observed. Regarding both accuracy and positive rate, the models trained with BP demonstrate superior prediction performance, with an average increase of 3.51% in accuracy and 3.83% in positive rate. On the other hand, the models trained with the combination of STDP and BP exhibit better efficiency performance, with an average decrease of 7.43% in firing rate.

5.2 Evaluation of prepossessing

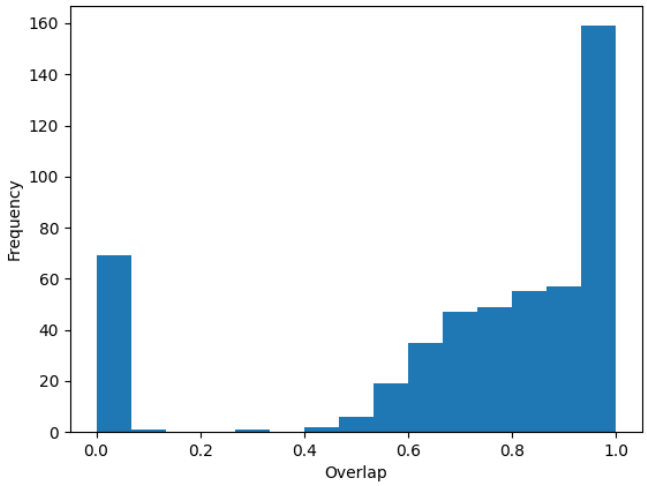

Figure 5.3: *Histogram of the overlap value between interval from the manual labeling and the extracted intervals list.*

As previously mentioned, a sudden increase in data amount can typically be observed when scratching behavior starts and ends. However, it is not always the case and thus some scratches are not included in the extracted interval list. For example, the algorithm may fail to find peaks or, in the worst case, there may be no peak to be detected, either because the movement was too little or was blocked by the body. Therefore the preprocessing method has a direct effect on the performance.

To evaluate the preprocessing method, a modified version of algorithm 1 was used. Each interval from the manual labeling was inputted and compared with all extracted intervals. The summed overlap percentage thus determines how well the interval is represented by the extracted interval list. Figure 5.3 shows a histogram of the overlap percentage. It can be seen that the absolute majority have a non-zero overlap percentage, meaning that they are somehow represented and over 85 percent of intervals have an overlap value higher than 0.5 and are therefore considered

well-represented .

5.3 Performance

It is important to note that the training results solely reflect the model's learning performance on the created dataset. To faithfully evaluate the overall accuracy, the impact of the preprocessing method should be considered. This can be done by multiplying the rate of well-represented scratches (0.85) with the test accuracy. As a result, the best model trained with BP achieves a final accuracy of 83.36%, and the best model trained with STDP combined with BP achieves a final accuracy of 80.63%.

6
Discussion

6.1 Model

The training results showed that the models solely trained with BP achieved the highest accuracy, which is expected given its widespread popularity and superior performance. Since the linear layers for the model were trained using identical methods and settings, one may conclude that the unsupervised STDP learning rule still lags behind in terms of feature extraction capability. To improve the accuracy, one approach could be to use a more advanced version of STDP as Anti-Hebbian STDP (aSTDP) or Reward Modulated STDP (R-STDP) [17] [18]. Another potential improvement is to adopt a more complex neuron model, either making the diff equations inherently more complex or incorporating additional learnable variables as potential thresholds. This has for example been done by Diehl et al., achieving a 95% accuracy on the MNIST dataset using a model solely trained on STDP [19].

Regarding the firing rate, the models trained with a combination of STDP and BP showed superior results. The superior results obtained could be attributed to the functioning of STDP. Since STDP is considered the learning rule for highly efficient biological networks, it is reasonable to assume that STDP itself possesses similar efficiency.

Although the loss function used incorporates the firing rate and thus encourages the model to minimize its firing rate, it is clearly possible to prioritize the efficiency further by multiplying the firing rate with a scalar greater than one. However, it is likely that such a modification would yield a negative impact on accuracy.

During this project, the efficiency was evaluated using the overall firing rate. An interesting idea would be to analyze the firing rate in the linear layers and convolutional layers separately to explore any potential trends.

The results demonstrate that SNNs can effectively address the problem outlined in this thesis. However, it is important to note that this does not guarantee that the training method or the trained model can be directly applied to real-world scenarios with certainty. One of the challenges posed by general neural networks is their black-box nature, which restricts everyone, including the developers from fully understanding their inner workings. This lack of understanding makes it difficult to accurately assess the risk of misclassification when applying the model to new and

unseen data.

6.2 Architecture

Most likely, adopting a more advanced architecture will have a positive effect on the model's performance. However, since the overall performance was mainly limited by the preprocessing stage, it was not considered a necessity to change the model architecture. Furthermore, as training methods were the focus of this thesis, the optimization of the architecture for energy efficiency was not pursued, although the possibility remains. For example, implementing skip connections between layers could bypass certain computations, resulting in a reduction in the overall computational load. The effectiveness of this approach would depend on the depth of the network, as skipping more layers would lead to more reduction in computation.

Another potential method is to modify the encoding method. In this thesis, rate encoding was used, meaning the spiking frequency is determined. However other types of encoding methods exist like latency encoding, which determines the timing of the spike instead. For example, the class of the neuron in the output layer that fires first is considered the prediction. Therefore, the simulation can be stopped as soon as any output neuron fires, resulting in less simulation time and less operation.

6.3 Dataset

Due to the constraints in resources and time, the dataset used in this project was limited in size, which probably had an effect on the final result. Despite the model seeming to successfully learn the pattern, the size of the dataset was on the verge. It was noted that a model trained with the dataset without any augmentations resulted in significant overfitting, meaning that the test accuracy decreased while the training accuracy increased.

It was also noted during training that the way to divide the dataset had an impact on the performance of the model. When the dataset was split chronologically instead of randomly, meaning that the first xx minutes were used as training data and the remaining as test data, both accuracy and the positive rate dropped significantly. This suggests that the patterns learned by the model are not yet generalized enough, even with data augmentation techniques applied.

6.4 Preprocessing

The method used to split the raw data into a dataset has a significant effect on the overall performance. However, it could still be argued that it is an acceptable

trade-off compared to the computationally intensive alternative.
The alternative involves using a fixed length and timestep equal to the average duration of scratching (0.42 seconds) to iterate through all 120 minutes of recordings, resulting in approximately 14,400 intervals. This is almost 30 times more than the current method's extraction of approximately 500 intervals.

6.5 Neuromorphic Hardware

The potential of SNN can be best utilized when it runs on Neuromorphic Hardware, which is specialized hardware to mimic the structure and functionality of the human brain. It enables efficient and parallel processing, emulating the brain's abilities for tasks like pattern recognition and learning.

Consequently, a promising next step for this project would involve executing the trained SNN on Neuromorphic Hardware, such as Loihi or SpiNNaker, or on a simulator that emulates such hardware devices. By doing so, not only can the model take advantage of the specialized features offered by Neuromorphic Hardware, but it would also allow a more fair comparison with other neuromorphic models or traditional computing approaches in terms of execution speed, energy consumed, and other real-time processing capabilities.

7
Conclusion

First of all, the result obtained indicates that a SNN is capable of detecting scratching behaviour despite the limitations imposed by the small size of the available data, thus showcasing the potential of SNNs in real-life applications.

It was noted that the primary bottleneck in detecting mouse scratching behavior lies in the preprocessing method, how the dataset was created from the raw data. Although chosen for its computational efficiency, the proposed method significantly impacts overall performance. Therefore, effort should be invested in developing an alternative preprocessing method that maintains computational manageability while avoiding performance degradation.

It was also noted that the SNN models trained with BP achieved higher accuracy(3.83% on average) but in return, the model trained with STDP combined with BP demonstrated a better firing rate efficiency (7.43% on average). As STDP is considered the learning rule for efficient biological networks, the results were expected.

To further enhance the overall performance, the dataset probably needs to be expanded upon, both in size and diversity. Some other improvements that can be made are for example consider exploring advanced versions of STDP, utilizing more complex neuron models, or modifying the architecture of the model.

A potential next step in a different direction is to train and deploy the model on Neuromorphic hardware, such as Loihi or SpiNNaker, with the expectation of better performance. This approach also allows more straightforward comparisons with other models.

Bibliography

[1] B. Yin, F. Corradi, and S. Bohté, "Effective and efficient computation with multiple-timescale spiking recurrent neural networks," pp. 1–8, 07 2020.

[2] M.-O. Gewaltig and M. Diesmann, "Nest (neural simulation tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[3] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, p. e47314, Aug. 2019.

[4] A. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "Pynn: a common interface for neuronal network simulators," *Frontiers in Neuroinformatics*, vol. 2, 2009.

[5] W. Fang, Y. Chen, J. Ding, D. Chen, Z. Yu, H. Zhou, T. Masquelier, Y. Tian, and other contributors, "Spikingjelly." `https://github.com/fangwei123456/spikingjelly`, 2020. Accessed: YYYY-MM-DD.

[6] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *arXiv preprint arXiv:2109.12894*, 2021.

[7] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma, "Bindsnet: A machine learning-oriented spiking neural networks library in python," *Frontiers in Neuroinformatics*, vol. 12, 2018.

[8] P. Elliott, M. G'Sell, L. M. Snyder, S. E. Ross, and V. Ventura, "Automated acoustic detection of mouse scratching," *PLoS One*, vol. 12, no. 7, p. e0179662, 2017.

[9] N. Inagaki, K. Igeta, J. F. Kim, M. Nagao, N. Shiraishi, N. Nakamura, and H. Nagai, "Involvement of unique mechanisms in the induction of scratching behavior in balb/c mice by compound 48/80," *European journal of pharmacology*, vol. 448, no. 2-3, pp. 175–183, 2002.

[10] H. Yu, J. Xiong, A. Y. Ye, S. L. Cranfill, T. Cannonier, M. Gautam, M. Zhang, R. Bilal, J.-E. Park, Y. Xue, *et al.*, "Scratch-aid, a deep learning-based system for automatic detection of mouse scratching behavior with high accuracy," *Elife*, vol. 11, p. e84042, 2022.

[11] N. Sakamoto, T. Haraguchi, K. Kobayashi, Y. Miyazaki, and T. Murata, "Automated scratching detection system for black mouse using deep learning," *Frontiers in Physiology*, p. 1466, 2022.

[12] K. Kobayashi, S. Matsushita, N. Shimizu, S. Masuko, M. Yamamoto, and T. Murata, "Automated detection of mouse scratching behaviour using convolutional recurrent neural network," *Scientific reports*, vol. 11, no. 1, p. 658, 2021.

[13] I. Park, K. Lee, K. Bishayee, H. J. Jeon, H. Lee, and U. Lee, "Machine-learning based automatic and real-time detection of mouse scratching behaviors," *Experimental Neurobiology*, vol. 28, no. 1, p. 54, 2019.

[14] ""specifications – current models ",inivation `https://inivation.com/wp-content/uploads/2023/02/2022-09-iniVation-devices-Specifications.pdf`."

[15] A. Khodamoradi and R. Kastner, "$o(n)$ o (n)-space spatiotemporal filter for reducing noise in neuromorphic vision sensors," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 1, pp. 15–23, 2018.

[16] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2661–2671, 2021.

[17] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe, and T. Masquelier, "Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks," *Pattern Recognition*, vol. 94, pp. 87–95, 2019.

[18] S. Li, "astdp: A more biologically plausible learning," *arXiv preprint arXiv:2206.14137*, 2022.

[19] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, vol. 9, p. 99, 2015.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY