DESIGN AND OPTIMIZATION OF HETEROGENEOUS FEEDFORWARD SPIKING NEURAL NETWORK FOR SPATIOTEMPORAL DATA PROCESSING

A Dissertation
Presented to
The Academic Faculty

By

Xueyuan She

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering
College of Engineering

Georgia Institute of Technology

May 2022

DESIGN AND OPTIMIZATION OF HETEROGENEOUS FEEDFORWARD SPIKING NEURAL NETWORK FOR SPATIOTEMPORAL DATA PROCESSING

Thesis committee:

Dr. Saibal Mukhopadhyay
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Arijit Raychowdhury
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Hyesoon Kim
School of Computer Science
Georgia Institute of Technology

Dr. Christopher Rozell
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Tushar Krishna
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date approved: January 11, 2022

ACKNOWLEDGMENTS

First and foremost, I want to thank my parents. My accomplishment today would not be possible without the support from my father and mother. When I look back at the time I have spent in schools, which is more than two decades till this moment, my parents are always there to provided guidance and encouragement whenever I need it. They helped me discover my potential and exceed my limit during the pursuit of knowledge. I want to thank my parents for never compromising on my education, by providing all the resources they could to support me during my study. I am deeply grateful to them and hope what I have accomplished truly makes them proud.

I want to thank Dr. Saibal Mukhopadhyay who has been a great advisor and mentor. His dedication to helping his students succeed is truly remarkable. During my study at Georgia Tech, I learned from Dr. Saibal Mukhopadhyay not only the technical skills to be a good researcher, but also important characteristics including patience, perseverance and curiosity, all of which I know will be invaluable to me for the rest of my life. I also want to thank Dr. Mircea R. Stan, who guided me, an undergraduate student knowing not what his future should be at that time, into the world of academic research.

I am glad to have the opportunity to work with many of the current and past members of the GREEN Lab, including Dr. Long Yun, Dr. Burhun A. Mudassar, Saurabh Dash, Daehyun Kim, Priyabrata Saha, Minah Lee, Biswadeep Chakrabarty and Beomseok Kang. I especially want to thank Saurabh Dash who provided highly valuable contribution to my research.

Last but not least, I want to thank Tongshu Yang, who has always been there for me during the past two years. She helped me greatly during the difficult times and I am very fortunate to have her sharing my joy during the good times. The years of 2020 and 2021 were not easy in many aspects. I am grateful that we can go through it together.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Over recent years, deep neural network (DNN) models have demonstrated break-through performance for many computer vision applications. However, such models often require a large amount of computation resources to operate, creating limiting factors for energy-constrained hardware platforms. Training DNN models also requires a high amount of labeled data, which could be difficult or expensive to acquire. The biologically inspired model of spiking neural network (SNN) is another type of network that is capable of processing computer vision data, and has the potential to achieve higher energy-efficiency than DNN due to its event-driven operations. SNN also has the capability to learn with biologically inspired algorithm that does not require training labels, i.e. unsupervised learning. While good performance has been shown for datasets with spatial-correlation, such as those in image classification tasks, the accuracy of SNN is still below that of DNN when the dataset has a higher level of complexity. This includes spatiotemporal tasks such as video classification and gesture recognition.

In this research, we tackle this problem by proposing a design methodology for feed-forward SNN that can be trained with either biologically inspired unsupervised learning algorithm or supervised statistical training algorithm, to achieve spatiotemporal data processing. The proposed model shows performance that is parallel to or better than DNN when the amount of labeled training data is limited. We derive theoretical analysis for the proposed design to help optimize network performance, and demonstrate with experimental results that the proposed design can achieve improved performance while using less trainable parameters. For event-based spatiotemporal data, we demonstrate that the efficiency of the proposed network can be further improved with a fully event-driven processing method.

CHAPTER 1
INTRODUCTION

Spiking neural network (SNN) is a type of artificial neural network (ANN) that is constructed with biologically inspired neuron and synapse models. One of SNN's main difference from conventional deep neural network (DNN) is that the neurons are dynamical systems with internal states evolving over time. In SNN, information is encoded as spike sequences with varying frequency (rate encoding) or delay (temporal encoding). The patterns contained in the spike sequences can be learned either with statistical training methods such as backpropagation-through-time (BPTT) [1], or with biologically inspired algorithms such as the unsupervised spike-timing-dependent plasticity (STDP) [2, 3, 4].

The event-driven nature of SNN operation promises high energy-efficiency during network operations [5]. Over the years, SNN has shown success in spatial data processing such as image classification. While many large scale SNN models depend on conversion from DNN [6, 7, 8] or supervised training [9, 10, 11], more recently, unsupervised learning has shown promising results [12, 13]. However, SNNs for processing temporal or spatiotemporal data are still primarily based on recurrent connections [14, 15], and networks that are trained with supervised training [16], leading to increased network complexity for processing spatiotemporal data and high demand for labeled training data. In addition, while promising performance have been demonstrated for SNN through empirical results, there lacks a theoretical understanding of the approximation capability of feedforward SNN for spike-sequence mapping functions.

One particular challenge of using SNN for spatiotemporal vision data processing originates from the complex dynamic of the changing visual patterns. For example, many computer vision applications involve observing objects in motion [17, 18], such as shown in Figure 1.1, where a robotic arm is interacting with a rolling object, and an unmanned

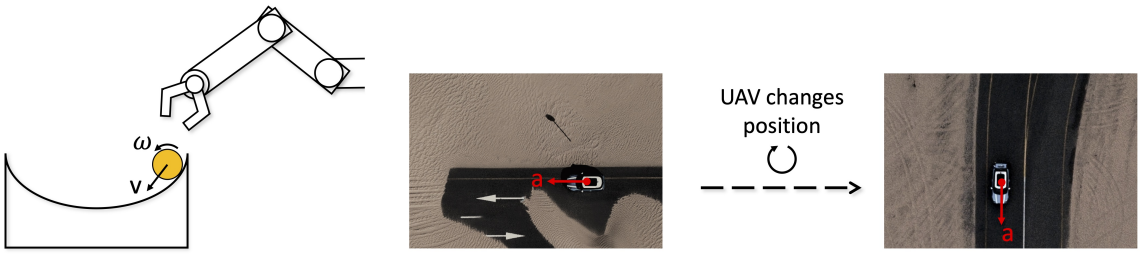


Figure 1.1: Objects with mixed dynamics in computer vision applications: robots interacting with a rolling ball in a convex (left); an UAV analyzing movement of a vehicle (right).

aerial vehicle (UAV) analyzing the motion of a moving car. In those applications, the object motion can be a mixture of translation and rotation, both with constant or changing speed. Proper operation of the visual signal processing model requires two processes to be performed: identifying class of the object and understanding dynamic of the motion. Specifically, the model needs to correctly classify known objects with unseen motion and recognize known motion of unknown objects. For spatiotemporal processing models that rely on camera-captured frame sequences as input, the constant transformation of pixel-level information makes the preceding problem challenging.

In terms of the DNN based approaches, spatiotemporal input data can be processed with 3D convolutional layers [19] or combining spatial data processing models with recurrent connections [20]. However, such DNN models do not generalize well for features that have been transformed. In order to achieve transformation invariant classification, data augmentation [21, 22] is the common approach to allow the networks to learn the transformed input features, and an increased amount of network parameters are needed to learn the extra information.

Regarding the spatiotemporal data sources, besides conventional cameras that capture frame-based images in a synchronous manner, there is another type of device, named event cameras [23], which capture the spatiotemporal information in an event-based manner. Event cameras use dynamic vision sensor (DVS) to capture brightness changes (increase and decrease) in a scene as asynchronous events. Compared to frame-based cameras, event cameras can achieve much higher temporal resolution due to the reduced data transmis-

sion, especially when observing scenes with few moving objects. Event camera also has advantages such as low power consumption and high dynamic range, making it an ideal data capturing device for applications that demand high power efficiency [24, 25, 26].

As the recent development of DNN offered state-of-the-art performance for frame-based visual data, applying similar approaches to process event camera data has received less attention. One type of approaches aims to convert the event based input to another representation that is similar to frames, and train conventional models based on the converted data [27, 28]. Such approaches take the advantage of well studied methods for processing frame-based images, but require aggregation of events over a period of time thus do not fully utilize the sparsity of event camera data. The second type focuses on designing networks that are event driven to better match the asynchronous property of event camera output[29, 30]. Such networks are asynchronous in nature, thus providing better efficiency for processing the sparse event camera output. While promising results have been achieved with both types of approaches, few has the capability of unsupervised learning. The event-driven operation of SNN makes it inherently suitable for processing event-based data. A system combining event camera and SNN has the potential to achieve a fully event-based image processing pipeline with high throughput and low latency. In addition, the unsupervised learning capability of SNN can be advantageous when the amount of training labels is limited. However, similar to processing complex frame-based spatiotemporal data, event-driven SNN processing for event camera data using STDP learning remains a challenging task.

1.1 Thesis Contribution

The objective of this research is to explore using heterogeneous neuron dynamics in feedforward SNN for spatiotemporal data processing, and study the learning performance and computation efficiency of such design. More specifically, the following contributions are made:

• We propose a novel feedforward SNN architecture consists of convolutional network layers with crossover connections between neurons with heterogeneous dynamics, which is capable of learning spatiotemporal patterns with spike-timing-dependent plasticity (STDP) unsupervised learning to achieve accuracy comparable to DNN with higher label efficiency.

• We develop approximation theorem of a single spike propagation path, referred to as a memory pathway, for any spike-sequence-to-spike-sequence mapping functions on a compact domain, and prove that using heterogeneous neurons having different dynamics and skip-layer connection increases the number of memory pathways a feedforward SNN can achieve and hence, improves SNN's capability to represent arbitrary sequences.

• We improve the proposed SNN architecture using the preceding theoretical observations and develop a dual-search-space Bayesian optimization process to experimentally demonstrate that the SNN architecture can be trained effectively for higher classification accuracy of complex spatiotemporal data.

• We develop an event-driven processing method for STDP and inference of event-based data using the proposed network architecture, which significantly reduces computation and increases network throughput compared to regular discrete-time method, while providing accuracy comparable to DNN and other SNN baselines.

• We show that the proposed event-driven SNN processing method is more robust than regular discrete-time simulation method when synaptic conductance has reduced precision, and delivers better accuracy with low-precision STDP learning.

1.2 Thesis Organization

In the following chapters of the thesis, we first present the background of SNN and DNN, and review prior works in spatiotemporal data processing using different SNN and DNN models in chapter 2. In chapter 3, we propose an architecture of heterogeneous feedforward SNN with convolutional layers that can effectively learn and predict for frame-based spatiotemporal datasets, and empirically investigate the learning performance of the proposed

network. Then, in chapter 4, we develop a theoretical basis for function approximation using feedforward SNN. Based on the derived theorem and lemmas we further improve the H-SNN design and demonstrate its performance with more complex spatiotemporal data classification task. In chapter 5, an event-driven SNN processing method designed for processing event camera data with the proposed architecture, is presented. The method is demonstrated in experiments for its advantages in computation efficiency and low precision learning. The thesis is summarized in chapter 6.

CHAPTER 2

BACKGROUND

2.1 Spiking Neural Network

The basic components of SNN are spiking neurons and synapses. Figure 2.1(a) is an illustration of a pair of neurons connected by a synapse. The two neurons are referred to as the pre-synaptic neuron, as shown on the left, and post-synaptic neuron as shown on the right. We first introduce the mathematical model of spiking neurons and then the learning rule to optimize synaptic conductance.

2.1.1 Spiking Neuron

There are different models that are developed to capture the dynamic of real biological neurons such as Leaky Integrate-and-Fire (LIF) and Hodgkin–Huxley[31]. In this thesis, we mainly study SNN constructed with the the LIF neuron model. The change of membrane potential v for a LIF neuron can be described by the following equations:

$$\tau_m \frac{dv}{dt} = a + R_m I - v \tag{2.1}$$

$$v = v_{reset}, \text{ if } v > v_{threshold} \tag{2.2}$$

Here, R_m is membrane resistance and $\tau_m = R_m C_m$ is the time constant with C_m being membrane capacitance. a is a parameter used to adjust neuron behavior during simulation. I is the sum of current from all synapses that connects to the neuron. A spike is generated when membrane potential v cross threshold and the neuron enters refractory period, during which the neuron can not spike again.

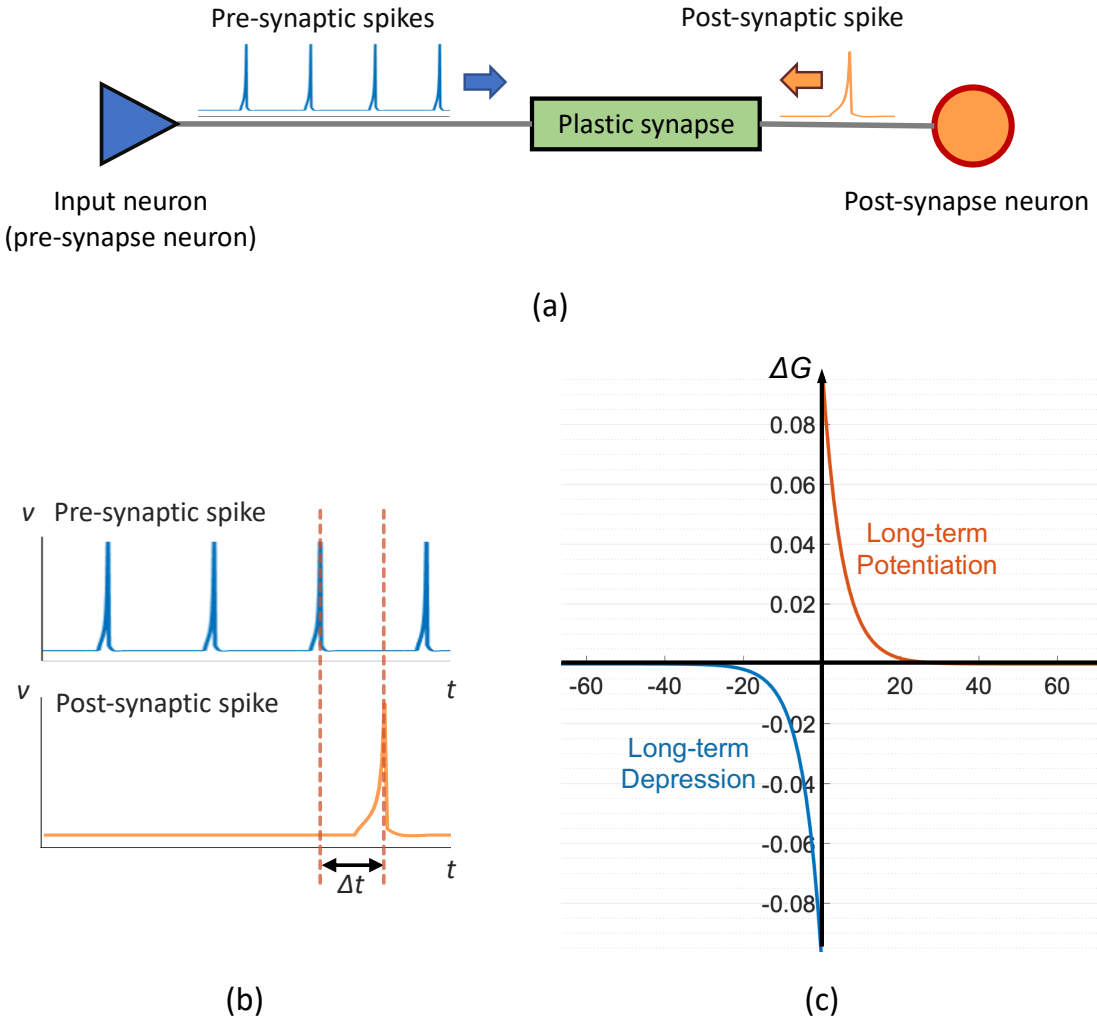


Figure 2.1: (a) Pre-synaptic and post-synaptic neurons connected by a synapse; (b) spike timing of pre-synaptic and post-synaptic neuron. The post-synaptic spike has timing different Δt with last input spike and LTP is induced; (c) STDP magnitude has an exponential relation with spike timing difference.

2.1.2 Spiking-timing-dependent Plasticity Learning Rule

Based on the biologically plausible network model of SNN, it is possible to implement learning algorithm that can exploit the causal relationship between spiking events [32, 33]. As shown in Figure 2.1(a), the pre-synaptic neuron emits spike signal when it reaches threshold, and the spiking signal is conducted by the synapse to the post-synaptic neuron. The signal excites post-synaptic neuron which accumulates membrane potential and can also emit a spike when it reaches threshold. Synaptic conductance therefore determines how strongly two neurons are connected. In the context of machine learning, synaptic conductance acts as connection weight between neurons, and learning is achieved through modulation of synaptic conductance. Meanwhile, unlike many conventional machine learning algorithms that is based on gradient descent, SNN adopts modulation rules similar to that in biological neural systems. One type of the conductance modulation rules is spiking-timing-dependent plasticity (STDP), which has been widely applied in SNN based machine learning applications [34, 35, 36, 37].

In the STDP learning algorithm, there are two types of conductance modulation behaviors: long-term potentiation (LTP) and long-term depression (LTD), which increases and decreases synapse conductance, respectively. LTP is induced when post-synaptic neuron emits a spike shortly after receiving an input spike from pre-synaptic neuron, indicating a causality between the two events; LTD is induced if the input spike is received after post-synaptic neuron spikes. As shown in Figure 2.1(b), when post-synaptic neuron spikes closely after a pre-synaptic spike, a positive time difference Δt is recorded. Querlioz[36] presents in his work an algorithm to determine the magnitude of conductance modulation and such algorithm has been tested in machine learning applications[38]. It is defined by the following two equations:

$$\Delta G_p = \alpha_p e^{-\beta_p (G - G_{min})/(G_{max} - G_{min})} \tag{2.3}$$

$$\Delta G_d = \alpha_d e^{-\beta_d (G_{max} - G)/(G_{max} - G_{min})} \tag{2.4}$$

Here, G_p is the magnitude for LTP and G_d for LTD; G is the value of synapse conductance before modulation; α_p, α_d, β_g and β_d are parameters with values above zero, G_{max} and G_{min} are network parameters. Figure 2.1(c) demonstrates the relationship between STDP magnitude with spike timing difference Δt as defined by this algorithm.

2.1.3 Approximation Theory of Feedforward SNN

While many theoretical approaches to analyze SNN [39, 40] focus on the storage and retrieval of precise spike patterns, this property is different from the approximation capability of spike-sequence-to-spike-sequence mappings functions, which is relevant to the pattern classification tasks studied in this research. Towards the approximation capability of SNN, a model that incorporates excitatory and inhibitory signal is shown for its ability to emulate sigmoidal networks [41] and is theoretically capable of universal function approximation. Feedforward SNN with specially designed spiking neuron models [42, 43] have been demonstrated for function approximation, while for networks using LIF neurons, function approximation has been shown with only empirical results [44]. On the other hand, the existing works that has developed efficient training process for SNN and demonstrated classification performance comparable to deep learning models, have mostly used simpler and generic LIF neuron models [45, 16, 46, 47, 7, 48, 49], contrasting the lack of theoretical basis for such networks.

2.2 Deep Convolutional Neural Network

Statistical machine learning models using deep convolutional neural network (CNN) and gradient based optimization process has demonstrated high performance in many computer vision application applications including image classification [50, 51, 52]. An example of

the architecture for deep convolutional neural network (CNN) used in computer vision is shown in Figure 2.2. Input images are processed with convolutional layers and pooling layers for feature extraction, and fully connected layers at the end are used to generate prediction results.

A common training algorithm for deep CNN is gradient descent. During the gradient descent process, the training images are used as the input to the feedforward network, which generates predicted probabilities of each image belonging to each class. The ground truth labels of the training images are used to compute loss values, which measure the distance between the generated prediction and ground truth. The loss function can be chosen as Weston Watkins formulation or cross-entropy loss, etc. Based on the chain-rule, gradient of loss with respect to network parameters can be back-propagated through the network, such as shown in Figure 2.2 where the gradient is back-propagated to the first convolutional layer. The gradient information is then used to update the parameter in the direction that would minimize the loss.

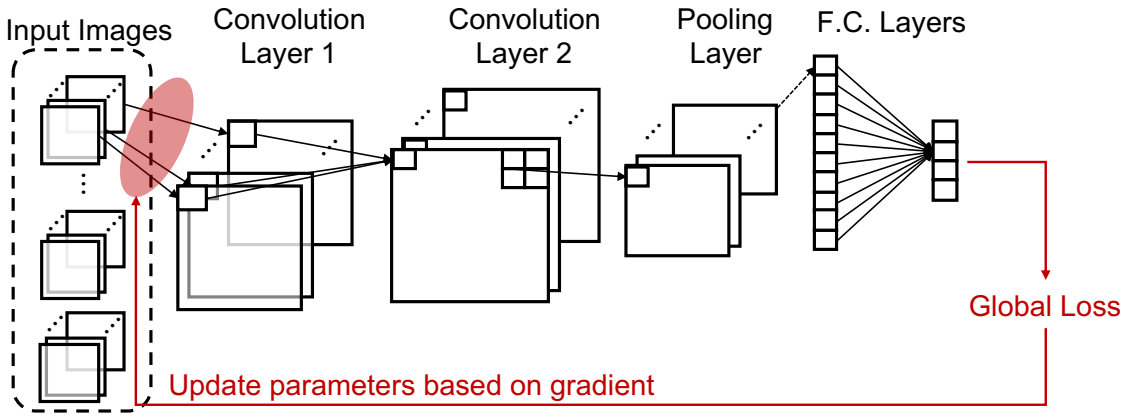


Figure 2.2: The gradient-based training process of deep CNN.

The datasets used for DNN training usually contain a large number of training samples, and it is inefficient to compute gradient over the entire training dataset. The alternative approach commonly used in DNN training is stochastic gradient descent (SGD), where a subset (batch) of the training set is used for each parameter update process. This is repeated

for all training image batches and one training epoch is complete. A network is trained over a number of epochs until certain stopping criteria is met. During inference, target images are processed by the feedforward network with optimized parameters, and prediction can be generated.

The gradient descent or stochastic gradient descent based weight update process for DNN training computes the new weight as $W' = W - \eta \nabla L$, where the gradient of loss function L is taken with respect to weight: $\nabla_w L = \langle \frac{\partial L}{\partial W_i}, ..., \frac{\partial L}{\partial W_k} \rangle$. Consider cross entropy loss as an example for L, weight optimization of element i is described by:

$$W_i' = W_i - \eta \frac{-\frac{1}{N} \partial \{ \sum_{n=1}^{N} [y_n log(\hat{y}_n)] \}}{\partial W_i} \tag{2.5}$$

Here η is the rate for gradient descent; N is the number of classes; y_n is a binary indicator for the correct label of current observation and $\hat{y}_n$ is the predicated probability of class n by the network.

2.3 Spatiotemporal Data Processing

2.3.1 Deep Neural Network

The development of deep learning over recent years leads to state-of-the-art solutions for many computer vision problems. DNN model can be designed to process spatiotemporal data with either 3D convolution layers [19] or recurrent connections [20]. However, in those spatiotemporal networks, transformation invariance is not explicitly imposed. To achieve transformation equivalent feature extraction, several DNN architectures have been proposed [22, 21, 53]. While it is possible to apply those designs in the spatiotemporal network, it has been demonstrated that DNNs are invariant only to images with very similar transformation as that in the training set [54], which indicates that DNNs generalize poorly when learning feature transformations. As a result, an increased amount of network param-

eters are required for conventional DNN to achieve transformation invariant classification. For example, in [21], each pre-defined rotation angle requires an additional set of CNN filters; in data augmentation based approaches such as [22], more parameters are needed to learn all the rotated features. Together with the combination of 3D kernel or recurrent connections, this leads to large complexity for spatiotemporal networks.

2.3.2 Spiking Neural Network

SNN with convolutional layers, trained with supervised and unsupervised methods, have shown good performance in image classification [12, 13]. SNN that can be used in time series tasks have demonstrated in [15, 10, 11, 14], which mainly focus on supervised training. Such networks are designed to predict for a single objective and do not enforce transform invariant/equivalent learning. For most STDP based SNN designed for computer vision applications, the spatial correlation are enforced with connection styles, and the dynamic of spiking neuron is exploited as an source of non-linearity. Within the network, all neurons have similar parameters that are optimized for STDP learning. This design performs well in learning spatial patterns, while for data that involve temporal dependency, change of network connectivity to recurrent is the common practice [14, 15]. A few feedforward network models have been demonstrated for spatiotemporal processing [16, 55] but are trained with gradient descent based methods. In addition, as empirical results are promising, a lack of theoretical understanding of sequence approximation using feedforward SNN makes it challenging to optimize network performance for complex spatiotemporal datasets.

CHAPTER 3
SPIKING NEURAL NETWORK WITH HETEROGENEOUS NEURON DYNAMICS

3.1 Overview

Biological study of nervous system has shown that neuron heterogeneity is an intrinsic property of brains [56, 57] and potentially acts as an important role in supporting physiological brain functions [58]. On the other hand, among the prior SNN models designed to process spatial or spatiotemporal datasets [45, 16, 46, 47, 7, 48, 49], most are constructed with homogeneous neurons. Based on this observation, we aimed to investigate how neurons with heterogeneous dynamics can be combined in a feedforward network structure and whether using heterogeneous neurons are beneficial to spatiotemporal data processing.

In this chapter, we first discuss the development of simulation platform used to support this study, then present heterogeneous spiking neural network (H-SNN) [59] as a novel, feedforward SNN structure with two neuron dynamics capable of learning complex spatiotemporal patterns with STDP based unsupervised training. Within H-SNN, hierarchical spatial and temporal patterns can be constructed with convolution connections and memory pathways containing spiking neurons with different dynamics. The proposed network is tested on visual input of moving objects to simultaneously predict for object class and motion dynamics and compared with baselines including DNN and SNN models.

3.2 H-SNN Simulation Platform

Based on the original *ParallelSpikeSim* (refered to as *ParallelSpikeSim-v1.0*) presented in our prior work [38], which was primarily used for shallow fully-connected spiking neural network, we made several key developments to improve the simulator as listed below. We

refer to the new version as *ParallelSpikeSim-v2.0*.

3.2.1 Convolutional SNN

While *ParallelSpikeSim-v1.0* can be used to process SNN with any connection schemes, it is primarily designed for fully-connected networks. The input signal to each spiking neuron is calculated based on a customized CUDA kernel for matrix multiplication and reduction. During development, we check two approaches of implementing convolution operations: customized CUDA kernel, and functions from the cuDNN library. We observe that in order to achieve the most efficient convolutional layers, using the cuDNN library, which contains highly optimized convolution operation implemented by NVIDIA, is the better approach. Therefore, we integrate the convolution operation workflow of cuDNN with the spiking neuron simulation process of *ParallelSpikeSim* to achieve fast and efficient convolutional spiking neuron layers.

Meanwhile, cuDNN does not support biologically inspired learning algorithms such as STDP. Due to the more complex network connections in convolutional SNN than the fully-connected SNN used in *ParallelSpikeSim-v1.0*, the original STDP implementation from *ParallelSpikeSim-v1.0* is no longer efficient. We therefore develop a new GPU accelerated STDP learning process which is discussed in more details below.

3.2.2 Simulation Process Optimization

We improve the computation efficiency of the simulation process in terms of memory consumption and simulation speed. *ParallelSpikeSim-v1.0* uses a design of unified data type (UDT) that combines all network parameters for neurons and synapses in one data structure. This design has the advantage of simplified software development process and fits the memory requirement of shallow, fully-connected network architectures. However, for convolutional SNN, the same conductance matrix is shared by neurons in the same depth. Number of neurons are also much higher than shallow, fully-connected networks. Using the

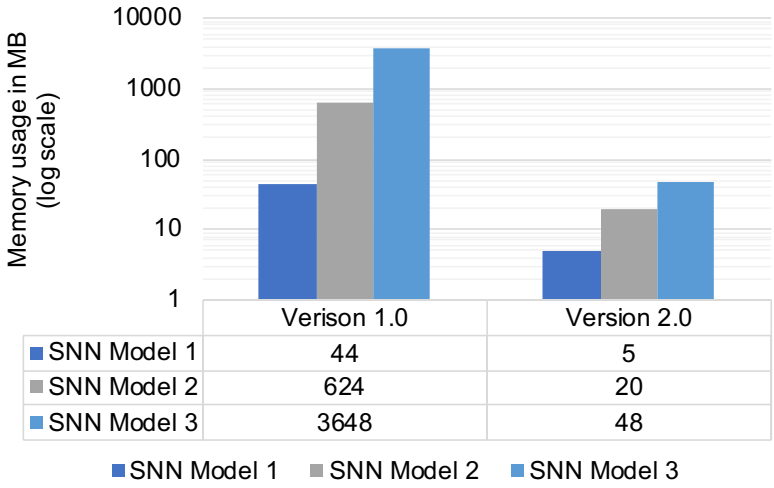


Figure 3.1: Comparison of memory usage for three networks simulated with two versions of *ParallelSpikeSim*

original UDT design is therefore not memory efficient as conductance matrices are stored repetitively. In *ParallelSpikeSim-v2.0*, new data structures are used for different components of the network. Instead of storing all conductance of synapses connected to each neuron separately as in *ParallelSpikeSim-v1.0*, we use a new pointer-to-pointer GPU memory structure that is optimized for the weight sharing scheme of convolutional networks. The structure is designed to be compatible with both the cuDNN convolution workflow as well as the STDP learning process of ParallelSpikeSim.

We test memory consumption of the two versions of *ParallelSpikeSim* based on three network structures. Since *ParallelSpikeSim-v1.0* does not support convolutional layers, the tested networks all contain fully-connected (F.C.) layers only, as shown in Table 3.1. Here the notation of [784x1000] indicates a layer with 784 input and 1000 output neurons.

Table 3.1: Network configurations

Model	Layer Configuration	Number of Synapses
SNN Model 1	F.C.{[784x1000],[1000x10]}	793K
SNN Model 2	F.C.{[3072x1000],[1000x10]}	3.08M
SNN Model 3	F.C.{[3072x1600],[1600x1400],[1400x10]}	7.17M

Memory usage values of the networks are shown in Figure 3.1. It can be observed that, from SNN Model 1 to SNN Model 2, number of network connections increases by 3.88 times, while memory usage in *ParallelSpikeSim-v1.0* increases by around 14 times. For *ParallelSpikeSim-v2.0*, the increase is much more linear with respect to the scaling up of the processed network, as memory usage is increase by around 4 times. The same trend can also be observed for SNN Model 3. *ParallelSpikeSim-v2.0* is more memory efficient particularly in larger networks: memory usage is around 8x less than the old version for Model 1, and 76x less for SNN Model 3. This result reflects a crucial improvement to the platform, which allows us to fit more complex networks on GPU memory.

In terms of the learning process, *ParallelSpikeSim-v1.0* implements STDP by launching CUDA kernels for all neurons at each timestep to check for spikes that induce conductance modulation. If a post-synaptic spike is detected, the kernel checks all pre-synaptic neurons serially. For deep convolutional SNN, launching kernels for the large amount of neurons inside the network is time consuming, and checking all pre-synaptic connection serially is inefficient. To solve this problem, during spiking neuron simulation, spike events are recorded in a spike-indicator array stored inside GPU memory. During STDP learning, a technique called dynamic parallelism is used: learning process is launched for spiked neurons only, and a second level of parallel processing is initiated by launching STDP kernels for all pre-synaptic neurons.

Other minor improvements include optimization of memory control that reduces data transfer between GPU and CPU memory; the lateral inhibition process, which was implemented within STDP kernel, is now separated as layer-by-layer operation to support local and cross-depth inhibition used by convolutional SNN.

3.2.3 Dynamic Network Structure

ParallelSpikeSim-v1.0 is designed to simulate SNN with static network structure during each simulation process. In *ParallelSpikeSim-v2.0*, dynamic network structure is sup-

ported. Specifically, configurations of each network layer, such as depth, kernel size and stride, and hyperparameters such as the spiking neuron parameters, can be changed while the platform is running network simulation. To achieve this, we develop a conversion function that changes the network structure based on the current and target network configurations. The process creates a new network object by re-using the conductance matrix and neuron object array from the original network structure in designated parts of the new network object, and fills the remaining data structure according to the new network configuration. The conversion process then automatically determines and updates the simulation parameters in all sub-modules, including the cuDNN workflow, neuron simulation, inhibition and STDP learning modules, such that they operate according to the new network structure. The support for dynamic network structure is another crucial development that enables us to implement the heterogeneous spiking neural network with convolutional layers.

3.3 The Proposed H-SNN Architecture

3.3.1 Heterogeneous Neuron Dynamics

In SNN, spiking neurons can be used as the basic element of information retention. Long and short retention length can be achieved if neurons have different membrane potential decay rates. Consider the LIF neuron model, which is reproduced here:

$$\tau_m \frac{dv}{dt} = a + R_m I - v;$$

$$v = v_{reset}, \text{ if } v > v_{threshold}$$

With this model, consider a spiking neuron with membrane potential at v_{reset}, which receives input current I at $t = 0$, the time t_{decay} for the neuron membrane potential to decay to v_{reset} after receiving the input, can be derived by solving the differential equation

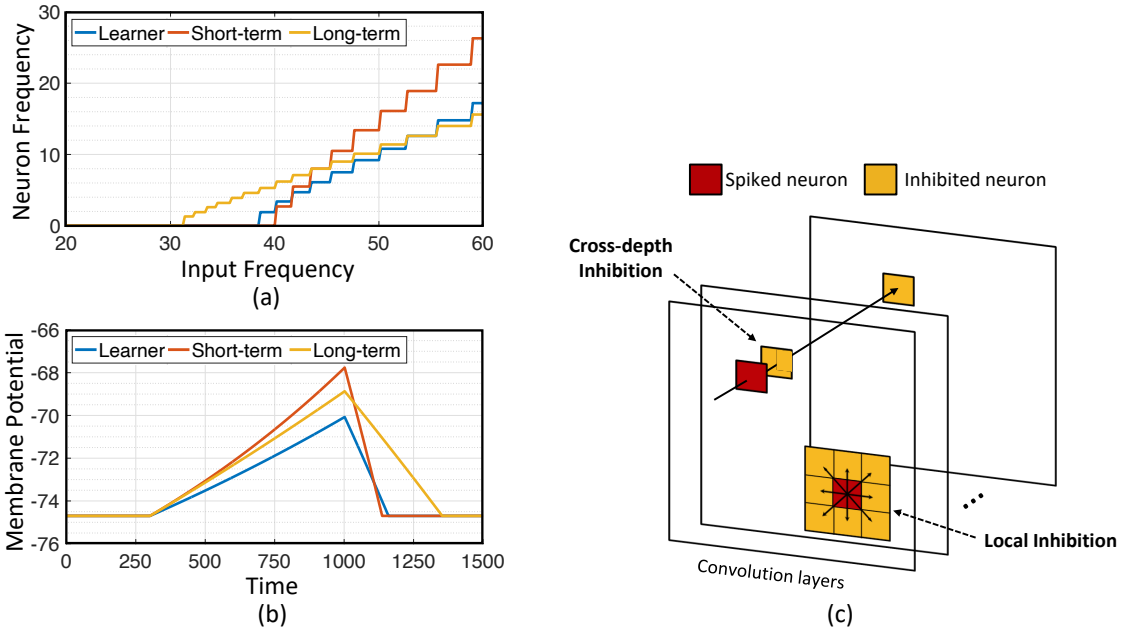


Figure 3.2: Neurons and inhibition: (a) Neuron response to input frequency; (b) neuron decay rate; (c) illustration of cross-depth and local inhibition.

Equation 2.1, leading to:

$$t_{decay} = \tau_m \ln(v_{reset} - a + \frac{R_m}{\tau_m} I) - \tau_m \ln(v_{reset} - a) \tag{3.1}$$

Equation 3.1 suggests that, by adjusting the parameters a, τ_m and R_m in Equation 2.1, different information retention period can be achieved. Particularly, in the proposed H-SNN architecture, three types of spiking neurons are used:

Learner neuron, which has a balanced decay rate and input response designed to optimize STDP learning, is similar to neurons used in previous works [36, 13, 12]. Its parameters are referred to as $\{a_{ln}, \tau_{ln}, R_{ln}\}$.

Short-term neuron, with parameters $\{a_{stn}, \tau_{stn}, R_{stn}\}$, has $\frac{a_{stn}}{\tau_{stn}} < \frac{a_{ln}}{\tau_{ln}}$ to create higher decay rate. It is used to extract short term patterns from input features.

Long-term neuron, with parameters $\{a_{ltn}, \tau_{ltn}, R_{ltn}\}$, has lower decay rate than learner neuron, as $\frac{a_{ltn}}{\tau_{ltn}} > \frac{a_{ln}}{\tau_{ln}}$. It is designed for long term pattern recognition.

Since membrane potential of long term memory neuron decays slower, under the same

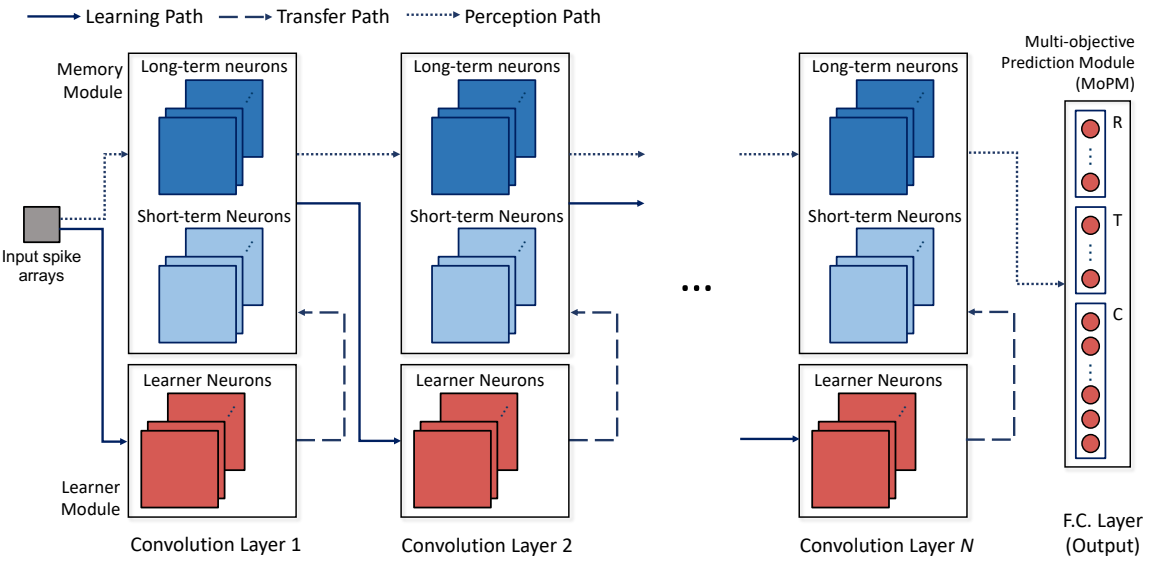


Figure 3.3: Architecture of the proposed network; network operation involves data flow of three paths: learning path where STDP learning is used to modulate synapse conductance, transfer path where learned features are transferred to long and short term neurons, and perception path, where perceived input spatiotemporal features are encoded in spikes.

input signal, it can potentially produce output spike frequency that dominates the short term memory neuron when the two are placed in parallel. To prevent this, $\frac{R_{stn}}{\tau_{stn}} > \frac{R_{ltn}}{\tau_{ltn}}$ is used. Figure 3.2 (a) shows responses of different neurons to pre-synaptic frequency. Long-term neuron is able to respond to lower frequency input, due to its lower decay rate, but its post-synaptic frequency increases slowly compared to the short-term neuron. Figure 3.2 (b) shows that short-term and long-term neurons gain membrane potential faster than learner neuron with a given input current, but they also exhibit different decay rates when input current is zero.

3.3.2 Network Architecture

The architecture of H-SNN is shown in Figure 3.3. Three types of modules are connected by three types of data flow paths between network layers. Spike signal from each layer's memory module is sent through perception path to deeper layers. Learning path connects memory module to learner module in the next layer to enable STDP learning. The learned

synapse conductance is transferred from learner module to memory module in the same layer. Modules are built with neurons of specific dynamics as mentioned before. Each convolution layer contains a learner module and a memory module. Two inhibition schemes are implemented within the convolutional layers: cross-depth, and local (Figure 3.2 (c)). Cross-depth inhibition is implemented to create competition between neurons with the same receptive field. This prevents more than one kernel from learning the same pattern. Local inhibition, where the spike of one neuron inhibits surrounding neurons in the same depth, is used to help the network to better detect and learn translation invariant features.

Learner module is responsible for facilitating STDP learning. All the spiking neurons in this module are learner neurons, and local inhibition is combined with cross-depth inhibition. For memory module, a combination of long-term and short-term neurons are used. Synapses in memory module are used only for perception thus not modified by STDP learning. Cross-depth inhibition is not implemented to accelerate neuron response and mitigate the diminishing spike frequency issue. The last layer is a multi-objective prediction module (MoPM) with each neuron fully connected to the previous layer. As will be discussed later, MoPM is fine-tuned with supervision. To facilitate the common conversion process, MoPM consists of all standard learner neurons. Inside the MoPM, neurons are indexed as $N_{i,j}$ where i represents one objective (section) and each j represent represent one label (class) within that section. Here, section-lateral inhibition is implemented, which means that spiking of neuron $N_{i,j}$ sends inhibition signal to all neurons in section i while other sections are unaffected. This enables H-SNN to simultaneously generate prediction for independent objectives.

3.3.3 Learning Process

For H-SNN to process spatiotemporal information in a dataset, the memory modules needs to transform the input sequences to output space that contains spatial features of target classes and spatiotemporal features of feature motion dynamics. The last layer of H-SNN

trained with stochastic gradient descent (SGD) can statistically correlate those attributes in the reduced dimension to likelihood of the input belonging to different classes.

During the learning process, each frame in the input sequences are converted to a 2D array of spike trains; frequency of the spike train for a pixel is proportional to the pixel's intensity (rate encoding). The network receives one frame at a time and observes it for a period (t_{train} chosen based on input frequency range) to generate sufficient spiking events. After all frames are learned the process repeats for the next sequence.

STDP learning of the network proceeds in a layer-wise manner. During learning of layer 1, neurons in the learner module receive input spikes and perform STDP learning. The learned conductance matrix is transferred to long-term and short-term neurons. Next, learning of layer 2 begins. In this step, memory neurons in layer 1 receive input spikes and generate spikes for the learner neurons in layer 2. After the learner neurons complete learning of all training sequences, the conductance matrix is transferred to layer 2 memory neurons. This process repeats for all convolution layers. While in one layer, the learner neurons exhibit different neuron activity with the long-term and short-term neurons, such layer-wise learning process allows the next layer to learn the changed spiking pattern using STDP. Since multi-layer SNN experiences diminishing spiking frequency along network layers, threshold of neurons in the memory module are scaled down to produce higher output spiking frequency. The scaling factor for each layer is tuned as network hyper-parameter, and its value is kept uniform within one layer to prevent distortion of output pattern. During training of the final fully connected layer (MoPM), the perception path produces spikes for each training sequence and spike frequency of all memory neurons in convolution layer n is calculated. The last layer is trained with conventional SGD algorithm to predict for multiple targets. The objective function of SGD is to minimize binary cross entropy loss between label vectors that are one-hot encoded for each prediction target and the layer output.

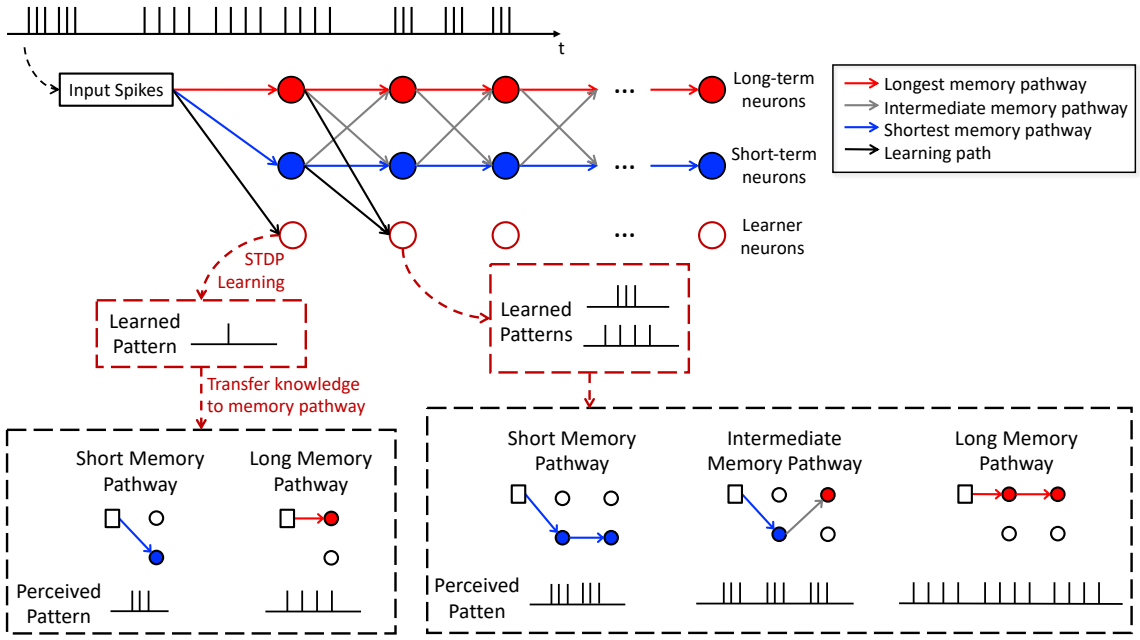


Figure 3.4: Illustration of memory pathways and hierarchical learning in H-SNN.

3.3.4 Memory Pathway - Hierarchical Memory Formation in H-SNN

Within the perception path, long-term and short-term neurons in different layers are connected with crossover connections. This establishes different memory pathways as shown by the red, grey and blue lines in Figure 3.4. More specifically, we refer to one trace of stacked connection of long-term and short-term neurons from the first memory module to the last, as a memory pathway. The memory pathways in H-SNN have a wide range of time scales. Connections consist of entirely short-term neurons, long-term neurons, and mixture of both types of neurons create memory pathways of shortest, longest, and intermediate time scales, respectively.

To better illustrate this, an example is shown in Figure 3.4. The memory pathways enable hierarchical learning of temporal patterns for the target spike train shown at the top of the figure. Learner neurons in the first layer extract correlation information from the immediate past (single spike in Figure 3.4) and transfer such knowledge to neurons in memory pathway (two different compositions of a single spike in Figure 3.4). The memory of layer 1 creates a higher level of temporal abstraction of input features that are transmit-

ted as spike inputs to the learner neurons in layer 2. Hence, learner neuron in layer 2 learns compositions of the higher level temporal patterns perceived by memory modules in layer 1 (see bottom of Figure 3.4 for illustration). This process is repeated throughout network learning, creating a hierarchical learning scheme of temporal features over different time-windows. In other words, the equivalent STDP learning window expands along network layers, with long memory pathways creating the faster expansion and short memory pathways creating the slower expansion. Temporal features of different scales can therefore be learned with the unsupervised STDP algorithm.

3.4 Results

3.4.1 Parameters and Simulation Configurations

The manually tuned neuron parameters are: for learner neurons, a is -75, τ_m is 100 and R_m is 31; for short-term neurons, a is -93, τ_m is 50 and R_m is 23; for long-term neuron, a is -34, τ_m is 100 and R_m is 16. Values of STDP parameters are: $\alpha_p = 0.1$, $\alpha_d = 0.03$, $G_{max} = 1.0$, $G_{min} = 0$, $\tau_{pot} = 10\,\mathrm{ms}$ and $\tau_{dep} = 80\,\mathrm{ms}$. To prevent early convergence of synaptic conductance and allow the network to effectively learn the entire dataset, the values of α_p and α_d are chosen to be relatively small, and training data is shuffled for both class and motion categories. In terms of the simulation process, unit timestep is set to 1 ms. Input frames are converted to spike trains with pixel intensity proportional to spike frequency range $f_{min}^{input} = 0\,\mathrm{Hz}$ and $f_{max}^{input} = 100\,\mathrm{Hz}$. Time spent on each frame is $t_{train} = 300\,\mathrm{ms}$. All neuron states are reset to default value after learning of each sequence.

3.4.2 Baseline Networks

Five DNNs are implemented to represent baselines for spatiotemporal processing, namely, (i) a simple 3D CNN [19] referred to as 3D CNN-α, which has similar layer configurations

Table 3.2: Network configurations and number of parameters

Model	Convolution Layer Configuration	Total Parameter
3D CNN-α	Conv3D {[3x3x3,20],[3x3x3,32], [5x5x5,64],[5x5x5,64]}	0.83M
3D CNN-β	Conv3D{[3x3x3,32],[5x5x5,64], [3x3x3,96],[3x3x3,128]x2}	4.5M
3D MobileNetV2	[60]	1.5M
3D ShuffleNetV2	[60]	1.2M
CNN+LSTM	Conv2D{[3x3,64],[3x3,128],[5x5,256]}	3.7M
BP-SNN/BP-SNN-LS	Conv2D{[3x3,32],[3x3,64],[5x5,128],[7x7,40]}	0.74M
H-SNN	Conv2D{[3x3,32],[3x3,64],[5x5,128],[7x7,40]}	0.74M

as H-SNN, (ii) 3D CNN-β, a more complex 3D CNN with more layers and parameters, (iii) 3D MobileNetV2 and (iv) 3D ShuffleNetV2 as implemented in [60]. The fifth baseline for comparison is an implementation of CNN+LSTM [20]. To prevent overfitting, for 3D CNN-α and 3D CNN-β dropout layers are applied; for all DNN baselines, early stopping for training are used. In [61], spatiotemporal back-propagation is shown for SNN and the trained network is tested for dynamic dataset. We implement this design with two variants as additional bio-inspired baseline networks. The first variant is referred to as BP-SNN, which has the same convolution layer configuration as H-SNN but does not use neurons of different dynamics. Based on the original BP-SNN structure, we implement refractory period to the neurons, and modified each layer to include neurons with long and short memory, similar to H-SNN. This second variant is referred to as BP-SNN-LS.

3.4.3 Network Complexity and Energy Dissipation

The configurations of convolution layers and network parameter number are shown in Table Table 3.2. Note, the representation of:

$$\{[5\text{x}5, 32], [5\text{x}5, 64], [5\text{x}5, 128]\}$$

Table 3.3: {R,T} prediction accuracy for unknown classes with aerial footage dataset

Model	$\sigma_{ts} = 1$ {R,T}	$\sigma_{ts} = 3$ {R,T}	$\sigma_{ts} = 5$ {R,T}
3D CNN-α	86.4, 88.7	60.7, 64.9	52.3, 56.3
3D CNN-β	95.2, 91.4	70.3, 65.3	61.4, 59.3
3D MobileNetV2	91.0, 83.6	67.1, 67.4	56.0, 61.6
3D ShuffleNetV2	97.5, 81.9	68.2, 62.2	67.6, 57.8
CNN+LSTM	93.1, 87.6	72.5, 66.7	63.1, 57.7
BP-SNN	84.7, 85.9	57.5, 64.2	51.4, 58.6
BP-SNN-LS	86.5, 94.8	62.3, 78.2	57.5, 60.4
H-SNN (1xU)	88.6, 91.0	68.0, 73.4	64.0, **63.2**
H-SNN (5xU)	92.3, **98.4**	**72.7**, **80.7**	66.3, **70.7**

denotes a network with one convolutional layer with 5x5 filter and 32 depth followed by a layer of 5x5 filter and 64 depth followed by a layer of 5x5 filter and 128 depth. The tested H-SNN has 0.74 million parameters. 3D CNN-α has similar complexity as H-SNN with 0.83 million parameters and BP-SNN has the same number of parameters as H-SNN. On the other hand, 3D CNN-β and CNN+LSTM contains 4.5 million and 3.7 million parameters. 3D MobileNetV2 and 3D ShuffleNetV2 has 0.5x complexity [60] and contains more parameters than H-SNN. For CNN+LSTM, parameters in the CNN encoder is 2.7M, while that in the LSTM decoder is 1.0M.

For H-SNN, network memory consist of two main parts: (i) synapse conductance, which are trainable parameters, use 0.474M, and (ii) neuron state, which are non-trainable variables, use 0.267M to store all membrane potential. The number of H-SNN's trainable parameters is thus significantly less than 3D CNN-β and CNN+LSTM. Moreover, it is well-known that the event-driven nature of SNN assists in reducing network activation which in turn reduced energy dissipation of computation. Using method presented in [62], we compute the energy advantage of H-SNN over DNN baselines during inference as follows: 1.0 for H-SNN, 1.55 for 3D CNN-α and 3.37 for 3D ShuffleNetV2; 3D MobileNetV2, 3D CNN-β and CNN+LSTM consumes $9.01\times$, $11.80\times$ and $28.88\times$ higher energy than H-SNN, respectively. BP-SNN and BP-SNN-LS uses similar energy as H-SNN.

Table 3.4: Accuracy result of event camera dataset for networks trained with 100%, 50% and 30% of labeled training data

Model	100%	50%	30%
3D CNN-α	92.8	90.5	86.3
3D MobileNetV2	97.0	94.2	90.4
3D ShuffleNetV2	97.3	95.4	90.1
H-SNN	96.2	93.8	**90.9**
H-SNN (full data)	96.2	**95.8**	**93.7**

3.4.4 Single-objective Prediction

Experimental Details

To test the effectiveness of H-SNN in learning spatiotemporal patterns, both single-objective and multi-objective experiments are conducted. In the single-objective experiment, an event camera dataset of human gesture [63] is used. Here, individual events are superimposed onto frames with resolution of 128x128 over 20 ms window. Each generated sequence contains one hundred frames and the network learns to predict the type of action in each sequence. Three baseline networks are tested: 3D CNN-α as well as the more complex 3D MobilNetV2 and 3D ShuffleNetV2.

To study the feasibility of learning with less labeled data and benefit of unsupervised learning, three sets of experiments are performed on DNN baselines using different training set sizes: 100%, 50% and 30% of the full training data set. In terms of H-SNN, two training configurations are tested: one that uses the same training set as DNN for STDP learning and SGD-based final layer tuning, referred to as **H-SNN**; the other, referred to as **H-SNN (full data)**, uses the full training set for STDP unsupervised learning, while SGD-based tuning uses the same set as DNN.

Results

As shown in Table Table 3.4, accuracy results from the three sets of experiments are listed. With full training dataset, accuracy of H-SNN is on a comparable level with 3D MobileNetV2 and 3D ShuffleNetV2 and outperforms 3D CNN-α. With less amount of labeled training data, all networks experience performance degradation. Among tested networks, H-SNN (full data) has the lowest accuracy decrease, while other networks lose around 7% from 100% training data to 30%. Such difference leads to the higher accuracy of H-SNN (full data) in low training data conditions, which indicates that unsupervised STDP learning of unlabeled data is effectively improving spatiotemporal pattern recognition in this particular task.

3.4.5 Multi-objective Prediction

Experimental Details

The second set of experiments is designed as a multi-objective computer vision task: the network observes a moving object as visual input to predict the class of the object and dynamic of its motion. We generate dataset with controlled motion dynamics by extracting objects from an aerial image dataset [64]. A subset (20%) of the original training data for each object class is used with label to test the benefit of unsupervised learning. In order to generate transformation sequences, objects are placed on canvas and applied with translation and rotation, each with five possible dynamics: static, constant speed, accelerating, decelerating and oscillating. For the training sequence, transformation dynamics are generated with parameters P_{train}. For the test sequence, objects are taken from the test set of [64] and transformation dynamic parameters are drawn from Gaussian distribution with mean P_{train} and standard deviation σ_{ts}. As a second, non-aerial test case, we have performed similar experiments on Fashion-MNIST dataset which are 10 classes of apparel items with the dimension of 28 by 28, as objects. Training and test sequences are generated follow-

ing the same process as discussed above, except that 10% of the original training data is used. For all generated sequences, training data is shuffled for both object class and motion dynamic.

In addition to regular training/inference setup, to understand the network's capability to learn class and motion independently, a total of three sets of experiments are conducted:

• ***Experiment 1: all-objective prediction*** In this regular training/inference setup, training set contains all classes and all possible transformation dynamics, and networks are tested for prediction of object class and its rotation/translation dynamics, which is either static, constant speed, accelerating, decelerating or oscillating.

•***Experiment 2: class-agnostic motion prediction*** In this experiment, networks are trained with sequences containing objects belonging to half of all classes, and tested for transformation dynamics prediction on the other half classes.

•***Experiment 3: motion-agnostic class prediction*** Training sequences contain all object classes, but with only a subset of motion dynamics. Networks are tested for accuracy of class prediction but with unknown dynamics.

Experiment 1 is conducted on both the aerial and Fashion-MNIST dataset, while the other two are tested on the aerial dataset only. Examples of training/test sequences for the three experiments are shown in Figure Figure 3.5. All baseline networks as discussed in Section subsection 3.4.2 are tested.

Training Configurations

We test H-SNN with two training schemes for the aerial dataset. The first one, referred to as **H-SNN (1xU)**, uses 20% of the training set mentioned before for STDP learning and SGD-based final layer tuning. To study the advantage of training with unlabeled data, we create a second network, **H-SNN (5xU)**, that uses $5\times$ more unlabeled data during STDP learning in SNN while the SGD-based tuning of the last year still uses the original labeled training set same as H-SNN (1xU). For Fashion-MNIST, **H-SNN (1xU)** uses sequences

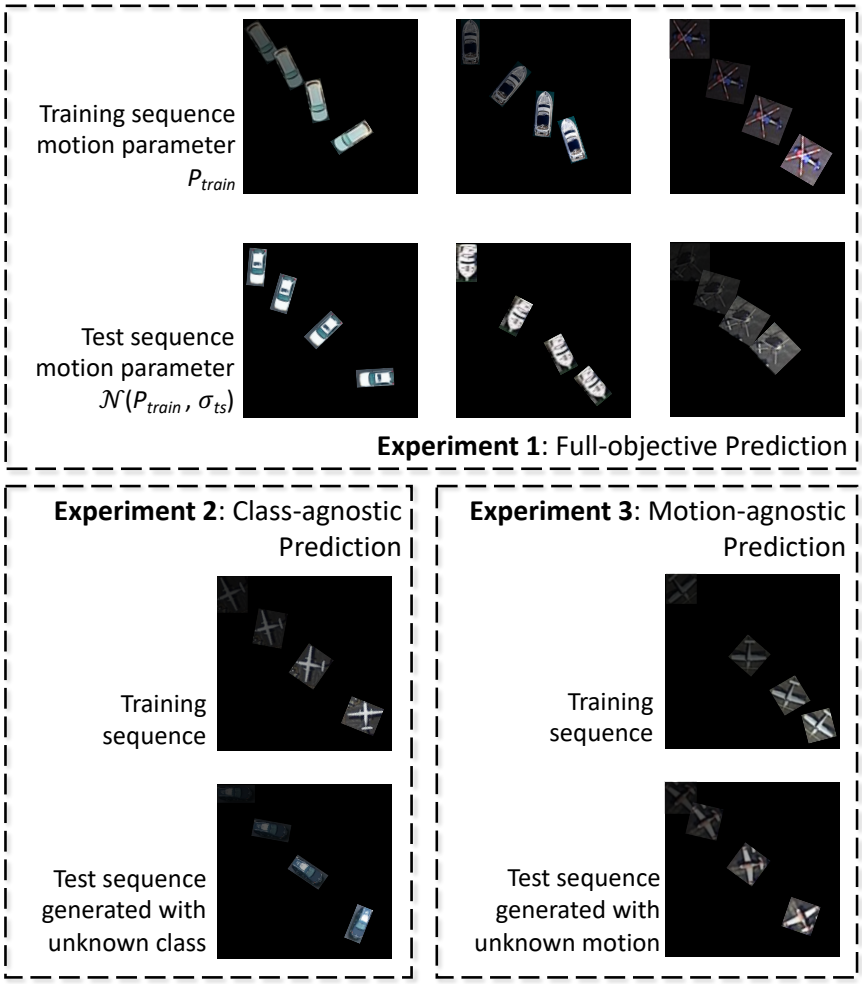


Figure 3.5: Illustrations of training and test sequences for the three experiments of the aerial footage dataset.

Table 3.5: Training accuracy of all-objective prediction with aerial footage dataset

Model	Joint	{C,R,T}
3D CNN-α	80.5	90.8, 94.9, 93.4
3D CNN-β	84.8	91.3, 96.5, 96.3
3D MobileNetV2	88.4	92.4, 97.7, 97.9
3D ShuffleNetV2	87.8	92.6, 96.8, 97.9
CNN+LSTM	86.7	90.7, 97.5, 98.1
BP-SNN	87.4	89.5, 98.9, 98.7
BP-SNN-LS	85.8	90.4, 97.5, 97.3
H-SNN (1xU)	84.8	91.4, 96.5, 96.1
H-SNN (5xU)	89.6	92.3, 98.1, 99.0

generated with 10% of original training set per class for unsupervised learning and final layer tuning; **H-SNN (10xU)** uses the whole training set for unsupervised learning and the same 600 objects per class sequences as used in H-SNN (1xU) for final layer supervised tuning. All baselines, including DNNs and SNN trained with back-propagation, are trained with the dataset used by H-SNN (1xU).

Aerial Footage Results

All-objective Prediction In all-objective prediction, results are measured by four metrics: accuracy for three separate targets and joint accuracy, which accounts for predictions that are correct for all three separate targets. Each objective's individual accuracy: class, rotation and translation, is referred to as C, R and T. Training accuracy for all networks are shown in Table 3.5. From the test accuracy results in Table 3.6, it can be observed that 3D MobileNetV2 and 3D ShuffleNetV2 show better accuracy than 3D CNN-α while they both fall behind 3D CNN-β and CNN+LSTM. BP-SNN-LS demonstrates better performance than BP-SNN in rotation and translation targets, while its class prediction is similar with BP-SNN.

With $\sigma_{ts} = 1$, H-SNN (1xU) predicts with good accuracy for motion dynamics and achieves a reasonable level of class prediction accuracy. This indicates that H-SNN is able

Table 3.6: Test accuracy of all-objective prediction with aerial footage dataset

Model	$\sigma_{ts} = 1$		$\sigma_{ts} = 3$		$\sigma_{ts} = 5$	
	Joint	{C,R,T}	Joint	{C,R,T}	Joint	{C,R,T}
3D CNN-α	51.3	64.5, 87.9, 90.4	23.2	58.9, 61.9, 63.5	16.0	56.1, 51.1, 55.8
3D CNN-β	58.6	67.1, 94.7, 92.2	31.6	64.9, 66.1, 73.7	25.3	62.8, 60.3, 66.9
3D MobileNetV2	54.7	66.9, 88.3, 92.7	24.2	64.1, 53.8, 70.1	18.0	61.9, 47.1, 62.0
3D ShuffleNetV2	53.8	64.8, 89.8, 92.5	26.1	63.2, 55.8, 73.9	19.9	60.8, 49.7, 65.9
CNN+LSTM	56.1	66.2, 92.8, 91.3	28.4	63.0, 68.3, 66.0	23.3	61.3, 63.8, 59.5
BP-SNN	49.5	65.1, 86.7, 88.3	23.6	61.7, 59.2, 64.6	17.4	60.4, 50.1, 57.3
BP-SNN-LS	58.1	66.7, 92.6, 94.1	32.5	60.1, 68.8, 78.7	26.7	58.3, 65.3, 70.2
H-SNN (1xU)	56.2	66.8, 91.0, 92.4	**34.0**	64.4, **70.5**, 74.8	**29.0**	**63.2**, **66.1**, 69.7
H-SNN (5xU)	**68.0**	**72.8**, 94.4, **98.8**	**44.6**	**69.5**, **78.4**, **81.8**	**32.9**	**66.4**, **68.3**, **72.6**

to learn spatiotemporal patterns from moving objects and predicts for separate objectives based on the learned patterns. Comparing H-SNN (5xU) to H-SNN (1xU), unsupervised learning provides considerable performance increase for all targets. With increasing σ_{ts}, accuracy for class prediction does not experience drastic degradation, showing that visual features learned by the network have a high degree of transformation invariance.

In comparison with baseline networks, accuracy values where H-SNN exceeds all baselines are marked bold in Table 3.6. For $\sigma_{ts} = 1$, H-SNN (1xU) outperforms BP-SNN and 3D CNN variants except for 3D CNN-β, while H-SNN (5xU) shows accuracy on a par with 3D CNN-β and CNN+LSTM. The advantage of H-SNN (1xU) is more evident in predicting motion with high deviation, as it achieves better results than baseline networks. This indicates that H-SNN is able to generalize more effectively the transformation invariant/equivariant patterns. With extra unlabeled dataset used for SNN learning, H-SNN (5xU) outperforms all baselines networks in most metrics. BP-SNN shows comparable accuracy as H-SNN (1xU) for class prediction, while its performance for motion prediction is noticeably lower than H-SNN (1xU), especially at higher σ_{ts}. BP-SNN-LS has similar performance with H-SNN (1xU) while still outperformed by H-SNN (5xU).

Confusion matrices for $\sigma_{ts} = 5$ are shown in Figure 3.6 (a). For each of the two variants of H-SNN, results are presented with three matrics. The matrix on the left is for class

prediction, the top right is for rotation and the bottom right is for translation. Horizontal axis is predicted label and vertical axis is target label, each marked with a number; for class prediction, 0-9 represents the 10 classes of objects; for rotation and translation, 0 is static, 1 is constant speed, 2 is acceleration, 3 is deceleration and 4 is oscillation. Lighter color represents more instances. It can be observed that, in terms of class prediction, confusion matrices of H-SNN (1xU) and H-SNN (5xU) share similarities, while H-SNN (5xU) predicts with more consistency across all classes. For motion prediction, learning unlabeled data has different effect: errors of H-SNN (5xU) for rotation prediction is more concentrated in one dynamic, while its errors for translation prediction spreads out more evenly, compared to H-SNN (1xU).

Class-agnostic Motion Prediction Table 3.3 lists accuracy of class-agnostic motion prediction. Each cell in the table contains accuracy for rotation (R) and translation (T). Result shows that the two H-SNN implementations are able to successfully predict motion dynamics of objects from unknown classes. Compared to motion dynamic accuracy from all-objective prediction, we observe that H-SNN experiences some degree of performance degradation. However, the decrease in accuracy is not drastic, especially for H-SNN (5xU). This shows that H-SNN is able to learn and predict motion dynamics independent of object's visual features on a certain level.

Among conventional deep networks, 3D CNN-β and CNN+LSTM show better accuracy than 3D CNN-α by a significant lead. 3D ShuffleNetV2 performs well in predicting rotation dynamic, while 3D MobileNetV2 shows good accuracy for translation dynamic. BP-SNN-LS has good performance for translation prediction, while the spatiotemporal patterns learned by BP-SNN are less generalizable, as its performance lags behind H-SNN and other baselines in this test. With increasing variation in transformation parameters, as observed in the $\sigma_{ts} = 3$ and $\sigma_{ts} = 5$ cases, accuracy of all networks degrade considerably.

Table 3.7: Motion-agnostic class prediction: configurations for class prediction test with unknown transformations where $\{s, c, a, d, o\}$: static, constant speed, accelerating, decelerating and oscillating.

	Training Dynamics	Test Dynamics
Pair I	T: $\{s, a, d, o\}$ R: $\{a, d\}$	T: $\{c\}$ R: $\{c, s, o\}$
Pair II	T: $\{a, d\}$ R: $\{s, a, d, o\}$	T: $\{c, s, o\}$ R: $\{c\}$
Pair III	T: $\{a, d\}$ R: $\{a, d\}$	T: $\{c, s, o\}$ R: $\{c, s, o\}$

Accuracy of H-SNN (1xU) is on similar level with the more complex DNNs and BP-SNN-LS, and higher than 3D CNN-α and BP-SNN. H-SNN (5xU) shows comparable or better performance than best baseline performance. Similar to all-objective prediction, the advantage of H-SNN (5xU) is more evident for high σ_{ts} cases.

Motion-agnostic Class Prediction The three training/test pairs of motion dynamics are shown in Table 3.7, and results for each pair is shown in Table 3.8. H-SNN is able to learn motion invariant spatial patterns as it predicts object classes with reasonable accuracy. For this task, accuracy of H-SNN (1xU) can again be improved further using more unlabeled data as shown in the H-SNN (5xU) results. This indicates a better generalization ability of H-SNN (5xU) for objects with unknown transformations.

In this test, 3D ShuffleNetV2 provides better performance than all other baselines. Compared to other networks, H-SNN (1xU) performs well for Pair I while keeping its advantage over 3D CNN-α and BP-SNN for all test cases. For Pair II, it is on a par with the best baseline results. H-SNN (5xU) achieves considerable improvement for all pairs by providing the best accuracy results except for Pair III, where it slightly falls behind 3D ShuffleNetV2. Hence, we observe that patterns learned from unlabeled data by STDP

Table 3.8: Motion-agnostic class prediction: accuracy of class prediction for different test cases with aerial footage dataset.

Model	Pair I	Pair II	Pair III
3D CNN-α	52.7	50.1	45.4
3D CNN-β	57.5	52.5	51.9
3D MobileNetV2	51.9	47.1	50.6
3D ShuffleNetV2	59.1	56.2	59.3
CNN+LSTM	56.2	56.6	53.1
BP-SNN	49.3	48.2	43.0
BP-SNN-LS	51.2	53.6	55.4
H-SNN (1xU)	**60.4**	54.0	53.4
H-SNN (5xU)	**64.7**	**60.6**	58.9

reduces the impact of unknown transformations to class prediction. For BP-SNN, degradation from all-objective prediction is significant, which indicates that the network has difficulty in learning spatial patterns invariant to unknown transformations. Compared to BP-SNN, BP-SNN-LS shows improvement in Pair II and Pair III while performs similarly in Pari I.

It is also worth noting that, compared to previously shown class prediction results, combinations of training/test dynamics affect network performance differently. For example, Pair II causes more degradation to 3D CNN-β than to CNN+LSTM while Pair I has similar influence to the two networks. For BP-SNN, Pair III shows to be most challenging. This indicates that the spatial patterns learned by networks generalize differently for unseen motion dynamics.

Fashion-MNIST Results

As shown in Table 3.9, both variants of H-SNN provide good accuracy for the three targets, indicating that the network is able to effectively learn the spatiotemporal patterns in moving apparel items, which have different visual features from the aerial footage dataset. H-SNN (10xU) shows higher accuracy than H-SNN (1xU) for class and translation motion prediction, while the improvement it achieves for rotation prediction is smaller.

Table 3.9: Accuracy result for sequences generated with Fashion-MNIST

Model	$\sigma_{ts} = 1$		$\sigma_{ts} = 3$		$\sigma_{ts} = 5$	
	Joint	{C,R,T}	Joint	{C,R,T}	Joint	{C,R,T}
3D CNN-α	54.3	70.4, 86.4, 89.2	19.0	50.2, 57.0, 66.5	10.9	43.9, 45.9, 54.4
3D CNN-β	63.1	72.5, 92.6, 94.0	28.4	61.0, 61.3, 75.9	18.2	58.9, 53.8, 57.4
3D MobileNetV2	57.6	70.0, 94.2, 87.3	22.6	58.3, 57.4, 67.6	13.8	53.8, 48.2, 53.1
3D ShuffleNetV2	61.0	68.7, 92.3, 96.2	23.9	53.4, 56.0, 79.8	12.1	43.2, 47.5, 59.0
CNN+LSTM	55.7	69.2, 89.6, 89.8	24.2	55.3, 74.7, 58.5	18.4	53.2, 64.1, 54.1
BP-SNN	54.5	68.1, 85.7, 93.4	19.5	47.0, 56.5, 73.5	14.3	50.6, 47.5, 59.3
BP-SNN-LS	70.0	76.8, 97.1, 94.1	37.2	66.0, 71.6, 78.7	26.0	62.5, 67.3, 61.7
H-SNN (1xU)	65.6	74.1, 96.2, 92.2	**38.9**	64.8, **83.7**, 71.8	**27.6**	59.2, **71.0**, **65.7**
H-SNN (10xU)	**73.9**	**79.4**, 96.8, **96.2**	**47.5**	**70.0**, **84.9**, **79.9**	**31.7**	**65.2**, **71.9**, **67.6**

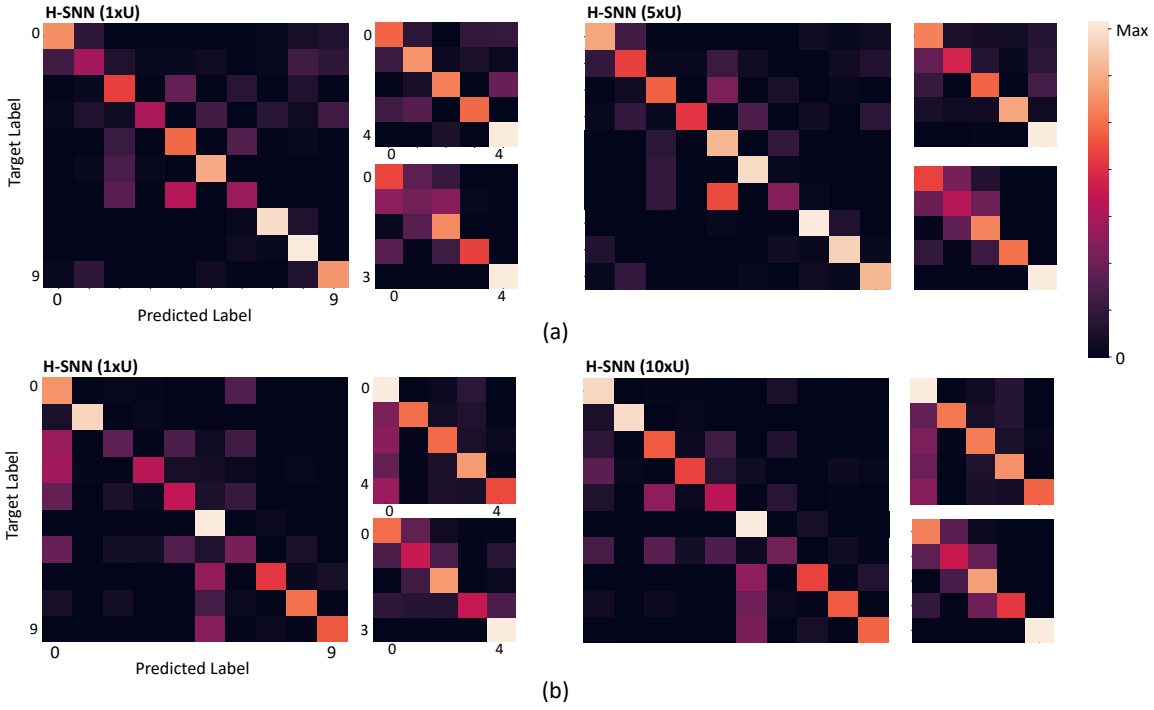


Figure 3.6: Confusion matrix for (a) aerial image sequences and (b) Fashion-MNIST sequences; for each network, the left matrix is for class prediction, the top right for rotation and the bottom right for translation; darker color represents less instances.

Table 3.10: Impact of training data size for aerial dataset (top 3 rows) with scaling unsupervised learning data size and Fashion-MNIST (bottom 3 rows) with fixed unsupervised learning data size.

	$\sigma_{ts} = 1$		$\sigma_{ts} = 3$		$\sigma_{ts} = 5$	
Training Set	Joint	{C,R,T}	Joint	{C,R,T}	Joint	{C,R,T}
20%	56.2	66.8, 91.0, 92.4	34.0	64.4, 70.5, 74.8	29.0	63.2, 66.1, 69.7
60%	76.5	81.5, 95.2, 98.6	49.0	73.7, 81.7, 81.3	34.6	70.4, 69.0, 71.2
100%	83.5	86.3, 97.2, 99.5	54.7	77.1, 84.9, 83.5	37.8	72.9, 70.5, 73.5
10%	73.9	79.4, 96.8, 96.2	47.5	70.0, 84.9, 79.9	31.7	65.2, 71.9, 67.6
50%	77.8	81.1, 97.6, 98.3	51.5	72.3, 86.5, 82.3	34.7	68.7, 73.6, 68.6
100%	82.3	85.3, 97.9, 98.6	57.4	76.8, 87.1, 85.9	38.1	74.2, 74.0, 69.4

From the confusion matrix in Figure 3.6 (b), the two H-SNN variants show similar profile for rotation and translation predictions while their class prediction differentiates. With unsupervised learning, H-SNN (10xU) is able to predict more accurately for classes that have high error rate, while the improvement on classes with lower error rate is less noticeable.

Among the baseline DNNs, 3D CNN-β shows good result at low σ_{ts} as it performs better than other deep networks and BP-SNN, and shares similar performance with CNN+LSTM in high σ_{ts} in terms of joint accuracy while each individual target differs. BP-SNN-LS shows considerable gain from BP-SNN and has the best performance among baseline networks. H-SNN (1xU) demonstrates similar accuracy as 3D CNN-β for $\sigma_{ts} = 1$ while its accuracy is lower than BP-SNN-LS. At higher σ_{ts}, H-SNN (1xU) shows comparative advantage: the prediction accuracy for multiple targets exceeds all baseline networks. This indicates that, for Fashion-MNIST, H-SNN is able to more effectively learn spatiotemporal patterns generalizable to different transformation dynamics. With unsupervised learning of extra unlabeled sequences, H-SNN (10xU) is able to make better prediction in almost all individual targets, and achieves joint accuracy considerably higher than baseline networks.

Impact of Training Data Size

In this section, we investigate the impact of scaling labeled training data on H-SNN, with two types of unsupervised learning setups. For the aerial dataset, unsupervised learning and supervised training of H-SNN both use sequences generated with 20%, 60% and 100% of the original dataset. For Fashion-MNIST, three levels of supervised training: 10%, 50% and 100%, are tested on a network that learns 100% data without supervision. The results are shown in Table 3.10. For the aerial dataset, it can be observed that by increasing training data size the network experiences considerable improvement on performance. The gain in class prediction accuracy is higher than that in motion prediction and the improvement in general is higher for lower σ_{ts} cases. When the network always learns 100% unlabeled data, similar trend can also be observed as shown in the Fashion-MNIST result. However, the benefit from increasing training data size is smaller than in the aerial dataset, e.g. for $\sigma_{ts} = 1$, joint accuracy increased by around 27% for aerial image, while for Fashion-MNIST the gain is around 9%.

3.5 Summary

In this chapter, we present H-SNN as a novel spiking neural network design that is capable of learning spatiotemporal information with STDP. For H-SNN, no recurrent connection is needed to process temporal patterns since memory of different retention length can be formed with crossover connections of heterogeneous neurons. The separation of memory module and learning module makes it possible to implement a feedforward convolutional network that can be learned with STDP unsupervised training.

We test H-SNN in computer vision tasks predicting for both single and multiple objectives, and demonstrate the effectiveness of H-SNN on datasets with different visual features and varying motion dynamics. H-SNN is compared with conventional DNN approaches including multiple variants of 3D-CNN, CNN with recurrent connections and SNN trained

with back-propagation. Results show two main advantages of H-SNN. First, H-SNN has comparable accuracy with DNN using the same amount of training data. Meanwhile, with the addition of unlabeled data, H-SNN can be further optimized with unsupervised STDP learning and provides higher accuracy than conventional DNN and BP-SNN. The advantage over baselines is most significant when motion dynamic has high deviation from training dataset. This trend is observed for H-SNN with and without extra unlabeled data to learn. The second advantage is that, with unsupervised learning, H-SNN demonstrates better generalization ability to unknown motion or classes in motion-agnostic and class-agnostic tests. Compared to BP-SNN, H-SNN more effectively learns transformation invariant spatial patterns as well as the general spatiotemporal patterns in the dataset. The combination of long and short term neurons in BP-SNN-LS produces considerable improvement over the original BP-SNN in terms of motion dynamics prediction accuracy. However, the supervised training method of BP-SNN-LS does not have the ability to learn unlabeled data, thus cannot benefit from the same technique used for H-SNN (5xU) and H-SNN (10xU) to further improve SNN performance.

In addition to the advantage in prediction accuracy, the improved performance of H-SNN is achieved with much lower network complexity than conventional deep networks. In conclusion, the design of using heterogeneous neuron in feedforward SNN provides an appealing solution for learning spatiotemporal patterns encountered in computer vision applications that have limited training data and/or constrained computing resources.

CHAPTER 4

SEQUENCE APPROXIMATION USING FEEDFORWARD-ONLY SNN

The empirical studies in chapter 3 show promising results of using H-SNN for spatiotemporal data processing, but the theoretical basis is not well understood for such networks. This makes it difficult to optimize the network configurations of H-SNN. In this chapter, we develop a theoretical framework for analyzing and improving spike-sequence-to-spike-sequence approximation using feedforward SNN. We consider a feedforward connections of spiking neurons as a spike propagation path that maps an input spike train with an arbitrary frequency to an output spike train with a target frequency. Consequently, we argue that an SNN with many memory pathways can approximate a temporal sequence of spike trains with time-varying unknown frequencies using a series of pre-defined output spike trains with known frequencies.

In the following sections, we discuss the motivation behind this study, and then present the main development as well as experimental results. Our theoretical framework aims to first establish SNN's ability to map frequencies of input/output spike trains within arbitrarily small error; next, we aim to derive the basic principles for adapting neuron dynamics and SNN architecture to improve spike-sequence mapping function approximation. The theoretical derivations are then investigated with experimental studies on using feedforward SNN for spatiotemporal data processing. We adopt the basic design principles for improving sequence approximation to optimize SNN architectures and study whether these networks can be trained to improve the performance for spatiotemporal classification tasks.

4.1 Motivation: Experiments on H-SNN Configuration and Performance

The original H-SNN uses two types of neuron dynamics for the memory modules. We first investigate how network performance changes in correlation with the configuration of

heterogeneity in the network. The native event-based data from DVS Gesture is tested. To better contrast the change of learning performance, the dataset of UCF-11, which is a frame-based action recognition dataset that is more challenging than the DVS Gesture, is also tested.

Based on the original H-SNN design, we test networks with an additional neuron dynamic creating different memory retention length from the original long-term and short-term neurons. The configurations of the three tested networks are shown in the top of Table 4.1. Accuracy results suggest that, in general, by increasing the level of heterogeneity, H-SNN can achieve higher performance. The particular amount of accuracy increase is dependent on the chosen neuron parameters as well as the target dataset. For example, *H-SNN 3-dynamic-iii* shows the best performance for both datasets, while *H-SNN 3-dynamic-i* and *H-SNN 3-dynamic-ii* performs better for DVS Gesture and UCF-11, respectively.

Table 4.1: H-SNN Scaling Test Results

Network	a	τ_m	R_m	Conv. Layer Number	Accuracy (DVS Gesture)	Accuracy (UCF-11)
H-SNN Original	-	-	-	4	94.3	61.3
H-SNN 3-dynamic-i	-150	50	35	4	95.1	65.1
H-SNN 3-dynamic-ii	-150	100	50	4	94.5	65.7
H-SNN 3-dynamic-iii	-50	200	80	4	95.6	66.0
H-SNN 3-dynamic-iii-3	-50	200	80	3	92.7	58.3
H-SNN 3-dynamic-iii-4	-50	200	80	4	95.6	66.0
H-SNN 3-dynamic-iii-5	-50	200	80	5	95.9	69.8
H-SNN 3-dynamic-iii-6	-50	200	80	6	96.0	71.2
H-SNN 3-dynamic-iii-7	-50	200	80	7	95.7	70.3
H-SNN 3-dynamic-iii-8	-50	200	80	8	95.1	68.7

In the original H-SNN design, there are 4 convolutional layers in the network. Next, we aim investigate the performance scaling properties of H-SNN with respect to the number of convolutional layers. Since the best performing neuron dynamic configuration from the previous test is *H-SNN 3-dynamic-iii*, in this experiment, we use the same neuron parameters from this network and change the number of convoluntional layers. The result is shown in the lower-half of Table 4.1. It can be observed that by changing the layer number

from 4 to 3, the network performance for both DVS Gesture and UCF-11 datasets experiences degradation. When the network is changed from 4 to 5 layers and from 5 to 6 layers, accuracy is improved while the amount of accuracy increase diminishes for higher layer numbers. From 6 layers, network performance no longer increases with additional layers and even shows slight decline.

The preceding results suggest that the configuration of H-SNN, including network structure and neuron parameters, has a considerable impact to network performance, and it is not always beneficial to have deeper networks. This raises two questions: first, how to decide the optimal configuration for H-SNN given a certain dataset to learn, and second, given the large amount of hyper-parameters in the network configuration settings, how to tune them efficiently. Regarding those questions, we start from the approximation theory of feedforward H-SNN, then derive lemmas that correlates network structures to the function approximation capability. Based on this insight, we develop an efficient Bayesian optimization process to determine H-SNN configurations for a given task.

4.2 Approximation Theory of Feedforward SNN

4.2.1 Definitions and Notations

We first provide the definitions and notations used in this chapter as below.

Definition 1 *Neuron Response Rate* γ For a spiking neuron n with membrane potential at v_{reset} and input spike sequence with period t_{in}, γ is the number of input spike n needs to reach v_{th}.

Definition 2 *Neuron Delay* t_{nd} The time for a spike from pre-synaptic neuron to arrive at its post-synaptic neurons.

Definition 3 *Minimal-layer-size Network* A minimal-layer-size network is a feedforward spiking neural network with a finite number of layers and one neuron in each layer.

Definition 4 *Distinct Memory Pathways* For a feedforward SNN with m layers, a memory pathway is defined as a spike propagation path connected by neurons in m-tuple

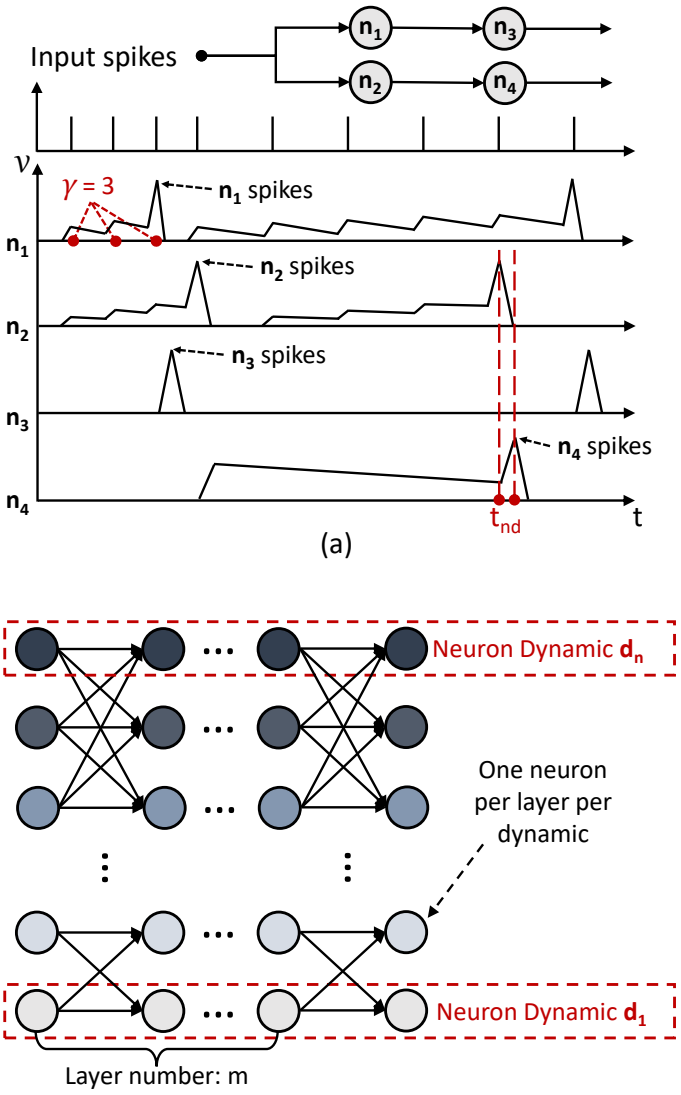


Figure 4.1: (a) A time-varying input spike sequence received by two memory pathways: neuron membrane potential plots show the different response from the neurons to the given input. (b) A minimal multi-neuron-dynamic (mMND) network with m layers and n neuron dynamics.

$\mathbb{P} = \{D_1, D_2, D_3, ..., D_m\}$ where D_i is the set of neurons included in layer i. $\mathbb{P}$ and $\mathbb{P}'$ are considered to be distinct if

$$\forall\ D_i \in \mathbb{P} \text{ and } D_i' \in \mathbb{P}',\ \exists\ i\ s.t.\ D_i \neq D_i'$$

Definition 5 *Skip-layer Connection* For a feedforward SNN with m layers, a skip-layer connection is defined with source layer and target layer pair (l_s, l_t), such that $l_s \in \{1, 2, 3, ..., (m-2)\}$, and $l_t \in \{(l_s+2), (l_s+3), (l_s+4), ..., m\}$. The output feature map from source layer is concatenated to the original input feature map of the target layer.

Definition 6 *Minimal Multi-neuron-dynamic (mMND) Network* A densely connected network in which each layer has an arbitrary number of neurons that have different neuron parameters. All synapses from one pre-synaptic neuron have the same synaptic conductance.

Figure 4.1(a) shows two memory pathways receiving an input spike sequence with time-varying periods. As the neurons have different dynamics, the two memory pathways have different response to the input spike sequence. An example of mMND network with m layers and n neuron dynamics is shown in Figure Figure 4.1(b). SNN with multilayer perceptron (MLP) structure can be considered a scaled-up mMND network with multiple neurons for each dynamic. A network with convolutional structure can be considered a scaled-up mMND network with duplicated connections in each layer. We analyze the correlation of network capacity and structure based on mMND networks, as for MLP-SNN and Conv-SNN network the analysis can be extended according to the specific layer dimensions.

Notations For the analysis of spike sequence in temporal space, the notation of T_{max} and T_{min} are defined as positive real numbers such that $T_{max} > T_{min}$. $\epsilon > 0$ is the error of approximation.

4.2.2 Modeling of Spiking Neuron

The spiking neuron model studied here is leaky integrate-and-fire (LIF) presented in chapter 2. Its equations are reproduced here:

$$\tau_m \frac{dv}{dt} = a + R_m I - v;$$

$$v = v_{reset}, \text{ if } v > v_{threshold}$$

R_m is membrane resistance, $\tau_m = R_m C_m$ is time constant and C_m is membrane capacitance. a is a parameter used to adjust neuron behavior during simulation. I is the sum of current from all input synapses that connect to the neuron. A spike is generated when membrane potential v cross threshold and the neuron enters refractory period r, during which the neuron maintains its membrane potential at v_{reset}. The time it take for a pre-synaptic neuron to send a spike to its post-synaptic neurons is t_{nd}. Neuron response rate γ is a property of a spiking neuron's response to certain input spike sequence. We show how the value of γ can be evaluated below.

Remark For any input spike sequence, individual spike can be described with Dirac delta function $\delta(t - t^i)$ where t^i is the time of the i-th input spike. For the membrane potential of a spiking neuron receiving the input before reaching spiking threshold, with initial state at $t = 0$ with $v = v_{reset}$, solving the differential equation (Equation 2.1) leads to:

$$v(t) = v_{reset} e^{-\frac{t}{\tau_m}} + a(1 - e^{-\frac{t}{\tau_m}}) + \frac{R_m}{\tau_m} e^{-\frac{t}{\tau_m}} \sum_i G \int_0^t \delta(t - t_n^i) e^{\frac{t}{\tau_m}} dt \tag{4.1}$$

Here, G is the conductance of input synapses connected to the neuron. From (Equation 4.1), there exists a timestep u such that $v_m(t^{(u-1)}) < v_{threshold}$ and $v_m(t^u) >=$

$v_{threshold}$. By evaluating (Equation 4.1) for u given neuron parameters and input spike sequence, the neuron response rate γ can be found.

Remark For a sequentially connected neuron list with m neurons all with $\gamma = 1$ and neuron delay t_{nd}, an input spike at time t leads the neuron list to generate an output spike at time $t + mt_{nd}$

Remark For any input sequence with period t_{in} to a spiking neuron with response rate γ such that $\gamma > 1$, if refractory period is set to $r < t_{in}$, the neuron can exit refractory period before the next spike arrives.

4.2.3 Approximation Theorem of Feedforward SNN

To develop the approximation theorem for feedforward SNN, we first aim to understand the range of neuron response rate that can be achieved. We show with Lemma 1 that for any input spike sequence with periods in a closed interval, it is possible to set the neuron response rate γ to any positive integer. Based on this property, we show with Theorem Theorem 1 that by connecting a list of spiking neurons with certain γ sequentially and inserting skip-layer connections, approximation of spike-sequence mapping functions can be achieved. To understand whether this capability of feedforward SNN relies on skip-layer connections, we develop Lemma 2 to prove that skip-layer connections are indeed necessary.

After the approximation theorem is established, in section 4.3 we investigate the correlation between approximation capability and network structures by analyzing the neurons' behavior based on their cutoff property, which can change the network's connectivity. In our analysis, we focus on two particular designs: heterogeneous network (Lemma 4) and skip-layer connection (Lemma 5), and show their impact on the number of distinct memory pathways in a network.

Lemma 1 *For any input spike sequence with period t_{in} in range $[T_{min}, T_{max}]$, there exist a spiking neuron n with fixed parameters v_{th}, v_{reset}, a, R_m and τ_m, such that by changing synaptic conductance G, it is possible to set the neuron response rate γ_n to be any positive*

integer.

Proof. For a given input spike sequence period t_{in}, consider the maximum possible membrane potential decay that can be reached within a period of t_{in}. From (Equation 2.1), when $I = 0$, $\frac{dv}{dt} < 0$ and $|\frac{dv}{dt}|$ increases with higher v. Hence the maximum decay of v is reached when initial membrane potential $v(t = 0) \to v_{th}^{-}$ and the neuron decays for period $t_{in} = T_{max}$. The decayed membrane potential $v(t = T_{max})$ can be derived by solving the differential equation (Equation 2.1) for $v(t) = v_{th}$ at $t = 0$:

$$v(t = T_{max}) = v_{th}e^{-\frac{T_{max}}{\tau_m}} - ae^{-\frac{T_{max}}{\tau_m}} + a \tag{4.2}$$

It is possible to have a spiking neuron with R_m, a and τ_m such that Δv, defined as

$$\Delta v = v(t = T_{max}) - v(t = 0) = v_{th}e^{-\frac{T_{max}}{\tau_m}} - ae^{-\frac{T_{max}}{\tau_m}} + a - v_{th} \tag{4.3}$$

,

tends to zero. With this configuration, since the the highest possible decay of membrane potential is negligible, for any target γ, it is possible to set G such that

$$G = \frac{v_{th} - v_{reset}}{\gamma} \tag{4.4}$$

The proof is complete.

Theorem 1 *For any input and target output spike sequence pair with periods* $(t_{in}, t_{out}) \in [T_{min}, T_{max}] \times [T_{min}, T_{max}]$*, there exist a minimal-layer-size network with skip-layer connections that has memory pathway with output spike period function* $P(t)$ *such that* $|P(t_{in}) - t_{out}| < \epsilon$.

Proof. For any given (t_{in}, t_{out}), first consider the condition where $t_{in} > t_{out}$. It is possible to construct a minimal-layer-size network N connecting m spiking neurons with neuron response rate $\gamma = 1$ sequentially, denoted as a m-tuple of neurons $\{n_1, n_2, ..., n_m\}$.

Since any configuration of skip-layer connection with source layer and target layer pair (l_s, l_t), such that $l_s \in [1, m-2]$, and $l_t \in [l_s+2, m]$, can be added, it is possible to add a $(m-2)$-tuple of skip-layer connections

$$S_{sl} = \{(i, m) \ \forall \ i \in \{1, 2, 3, ..., m-2\}\} \tag{4.5}$$

Denote the synaptic conductance for all the skip-layer connections as a $(m-2)$-tuple

$$S_{G^{sl}} = \{G_1^{sl}, G_2^{sl}, G_3^{sl}, ..., G_{m-2}^{sl}\} \tag{4.6}$$

For any $t_{out} < t_{in}$, it is possible to find a k-tuple of synaptic conductance

$$S'_{G^{sl}} = \{G_i^{sl}, G_{2i}^{sl}, G_{3i}^{sl}, ..., G_{ki}^{sl}\} \text{ s.t. } i = \lfloor \frac{t_{out}}{t_{nd}} \rfloor \text{ and } k = \lfloor \frac{m-2}{i} \rfloor \tag{4.7}$$

Set synaptic conductance in $S_{G^{sl}} \setminus S'_{G^{sl}}$ to 0. Then set the conductance of synapse connecting n_{m-1} and n_m to 0. In such way, The output spikes from network N has period

$$P(t_{in}) = \lfloor \frac{t_{out}}{t_{nd}} \rfloor \cdot t_{nd} \tag{4.8}$$

For given ϵ, it is possible to choose t_{nd} such that $t_{nd} < 2\epsilon$, therefore satisfying $|P(t_{in}) - t_{out}| < \epsilon$. m can be chosen as

$$m = \frac{T_{max} - T_{min}}{t_{nd}} \tag{4.9}$$

or equivalently:

$$m = \frac{T_{max} - T_{min}}{2\epsilon} \tag{4.10}$$

Since $\frac{T_{max}-T_{min}}{2\epsilon}$ is finite, m is finite.

For $t_{in} < t_{out}$, using N as described above, it is possible to achieve output spike with

period within ϵ of any period in $(0, t_{in}]$. For a given t_{out}, assume the configuration in neuron list N has output spike interval t'_{int} such that $kt'_{int} = t_{out}$, where k is a positive integer. From Lemma 1, it is possible to set G for a neuron n_{m+1} such that its neuron response delay satisfies $\gamma_{n_{m+1}} = k$ for input spike period t'_{int}. A new network, denoted as N', can be formed by connecting n_{m+1} to the output of N. N' has output spike with period $P(t_{in}) = kt'_{int} = t_{out}$. Hence, to reach the given ϵ, it requires neuron list N to have output spike interval t_{int} such that

$$|t_{int} - t'_{int}| < \frac{\epsilon}{k} \tag{4.11}$$

Since k is finite, (Equation 4.11) can be achieved.

For $t_{in} >= t_{out}$, it is possible to configure network N' such that t_{int} satisfies $|t_{int} - t_{out}| < \epsilon$, and the value of $\gamma_{n_{m+1}}$ set to 1, hence $|P(t_{in}) - t_{out}| < \epsilon$ can be achieved. The proof is complete.

Lemma 2 *With no skip-layer connection, there does not exist a minimal-layer-size network that has output spike period function $P(t)$ such that for any input and target output spike sequence pair with periods $(t_{in}, t_{out}) \in [T_{min}, T_{max}] \times [T_{min}, T_{max}]$, $|P(t_{in}) - t_{out}| < \epsilon$.*

Proof. A minimal-layer-size network N with m layers can be denoted as a m-tuple of neurons $\{n_1, n_2, ..., n_m\}$ connected sequentially. Since no skip-layer connection exists, there is only one distinct memory pathway that contains all neurons $\{n_1, n_2, ..., n_m\}$.

Denote the set of neuron response rate corresponding to each neuron in N as

$$\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_m\} \tag{4.12}$$

For a given input sequence with t_{in}, denote the timing of the first spike as t_1^{in}, consider the output spike sequence for network with

$$\gamma_i = 1 \ \forall \ \gamma_i \in \Gamma \tag{4.13}$$

The first output spike from N has timing $\tilde{t}_1^{out} = t_1^{in} + mt_{nd}$, and the second output spike has timing $\tilde{t}_2^{out} = t_1^{in} + t_{in} + mt_{nd}$. It can be easily derived that the period of the output spike sequence is

$$P(t_{in}) = t_{in} \tag{4.14}$$

Also consider the output spike sequence for network with

$$\gamma_j = 2 \text{ for any } j \in \{1, 2, 3, 4, ..., m\} \text{ and } \gamma_i = 1 \ \forall \ i \in (\{1, 2, 3, 4, ..., m\} \setminus \{j\}) \tag{4.15}$$

Following the same process, the period of the output spike sequence is

$$P(t_{in}) = 2t_{in} \tag{4.16}$$

Since the smallest increase to any γ_i is by 1, there is no set of values for γ such that the network output spike sequence has period $P(t_{in})$ satisfying $t_{in} < P(t_{in}) < 2t_{in}$. Since within the range $(t_{in}, 2t_{in})$, there exists values of t_{out} such that $|P(t_{in}) - t_{out}| < \epsilon$ does not hold. The proof is complete.

4.3 Network Structure and Memory Pathways

Based on Theorem 1, it is possible to approximate an input to output spike sequence mapping function using a minimal-layer-size network with specific configuration, which can be considered as a memory pathway. Since any continuous bounded function on a compact interval can be approximated to arbitrary accuracy using a piece-wise constant function, and it is possible to use a memory pathway to approximate each of the piece-wise constant function, with increasing number of distinct memory pathways, a feedforward SNN can achieve approximation of continuous functions with less error. In this section, we show

that two SNN structural designs: heterogeneous network i.e. a network having neurons with different dynamics and adding skip-layer connections, a feedforward SNN has the capability to achieve more distinct memory pathways.

In this subsection, we first prove the existence of a property of spiking neurons: cutoff period, and then analyze two SNN structural designs: heterogeneous network and skip-layer connection, for their impact on the number of achievable distinct memory pathways in mMND networks.

4.3.1 Neuron Cutoff Period

We first show the correlation of cutoff period and spiking neuron parameters with Lemma 3.

Lemma 3 *A spiking neuron has cutoff period* $\omega_c = \tau_m \ln(\frac{v_{reset}-a}{v_{reset}-a+\frac{R_m}{\tau_m}G})$ *above which input spike sequence cannot cause the spiking neuron to spike.*

Proof. Consider (Equation 4.1), since the membrane potential increases at time of t^i and decays otherwise, solving for $t = t^i$ and the equation can be expanded:

$$v_m(t^i) = v_{reset}e^{\frac{t^i}{\tau_m}} + a(1 - e^{\frac{t^i}{\tau_m}}) + \frac{R_m}{\tau_m}Ge^{\frac{t^i-t^1}{\tau_m}} + \frac{R_m}{\tau_m}Ge^{\frac{t^i-t^2}{\tau_m}} + ... + \frac{R_m}{\tau_m}G \quad (4.17)$$

For input with frequency f, $t^{i+1} - t^i = \Delta t = \frac{1}{f}$, subtracting membrane potential values at two consecutive t_i provides:

$$\Delta v_m = v_m(t^{i+1}) - v_m(t^i) = v_{reset}(e^{\frac{t^{i+1}}{\tau_m}} - e^{\frac{t^i}{\tau_m}}) - a(e^{\frac{t^{i+1}}{\tau_m}} - e^{\frac{t^i}{\tau_m}}) + \frac{R_m}{\tau_m}Ge^{\frac{t^{i+1}-t^1}{\tau_m}} \quad (4.18)$$

setting time of first input spike t^1 to zero leads to:

$$\Delta v_m = e^{\frac{t^i}{\tau_m}}((e^{\frac{\Delta t}{\tau_m}} - 1)(v_{reset} - a) + \frac{R_m}{\tau_m}Ge^{\frac{\Delta t}{\tau_m}}) \quad (4.19)$$

As $e^{\frac{t^i}{\tau_m}} > 0$, and the term $((e^{\frac{\Delta t}{\tau_m}}) - 1)(v_{reset} - a) + \frac{R_m}{\tau_m} G e^{\frac{\Delta t}{\tau_m}})$ does not depend on t^i, the polarity of Δv_m does not change with time. v_m is either strictly increasing, staying the same or decreasing with higher t^i. This indicates that, when $\Delta v_m \leq 0$ the post-synaptic neuron can never spike regardless of how many pre-synaptic spike it receives. $\Delta v_m \leq 0$ when input spike period t_{in} satisfies

$$t_{in} \geq \tau_m \ln(\frac{v_{reset} - a}{v_{reset} - a + \frac{R_m}{\tau_m} G}) \tag{4.20}$$

Therefore, the cutoff period of the neuron is

$$\omega_c = \tau_m \ln(\frac{v_{reset} - a}{v_{reset} - a + \frac{R_m}{\tau_m} G}) \tag{4.21}$$

The proof is complete.

Remark From Lemma 3, it can be observed that the cutoff period ω_c of a neuron can be configured to any positive real number by changing the neuron parameters and synaptic conductance G. Further, with fixed G, ω_c can be configured to any positive real number by changing the neuron parameters. Neurons that are in cutoff change the spike propagation path in a network as they send no output spikes. This creates different memory pathways without changing the connections in a network. In the following proof, we consider cutoff frequency, $f_c = \frac{1}{\omega_c}$ of spiking neurons.

4.3.2 Heterogeneous Network

If an mMND network has the same parameters for all neurons in each layer, the majority of the neurons are included in the same memory pathway, leading to the upper bound of number of distinct memory pathways to be limited. With Lemma 4, we show the relationship between the upper bound of the number of distinct memory pathways and the number of different neuron dynamics in an mMND network.

Lemma 4 *For an mMND network with m layers and $\{\lambda_1, \lambda_2, ...\lambda_m\}$ number of different neuron dynamics in each layer, the least upper bound of the number of memory pathways is $\prod_{i=1}^{m} \lambda_i$.*

Proof. Denote the set of neurons in layer l with distinct neuron dynamics as

$$S_n^l = \{n_1^l, n_2^l, n_3^l, ..., n_{\lambda_l}^l\} \tag{4.22}$$

Since the network is mMND, $|S_n^l| = \lambda_l$. Denote the set of cutoff frequency corresponding to each neuron in S_n^l as

$$F_c^l = \{f_c^{n_1^l}, f_c^{n_2^l}, f_c^{n_3^l}, ..., f_c^{n_{\lambda_l}^l}\} \tag{4.23}$$

Since neurons in S_n^l can have different neuron parameters, from Lemma 3, it is possible to set the parameters such that all entries of F_c^l are distinct. Hence, there exists a permutation π such that

$$f_c^{n_{\pi(1)}^l} < f_c^{n_{\pi(2)}^l} < f_c^{n_{\pi(3)}^l} ... < f_c^{n_{\pi(\lambda_l)}^l} \tag{4.24}$$

Denote the input spike frequency to layer l as f_{in}^l, neuron n_i^l is a part of a valid memory pathway in the network if

$$f_c^{n_i^l} \leq f_{in}^l \tag{4.25}$$

If the input spike frequency is $f_{in}^l \geq f_c^{n_{\pi(\lambda_l)}^l}$, all neurons in S_n^l can be a part of a valid memory pathway of the network. For input spike frequency such that $f_c^{n_{\pi(i)}^l} \leq f_{in}^l < f_c^{n_{\pi(i+1)}^l}$, i neurons can be a part of a valid memory pathway of the network.

For any input to the network $f_{in} \in [F_{min}, F_{max}]$, denote the number of ways different neurons in S_n^l can be part of valid memory pathways as k_l. The total number of distinct memory pathways $\mathcal{K}$ in the network is

$$\mathcal{K} = \prod_{l=1}^{m} k_l \tag{4.26}$$

Since $0 \leq k_l \leq \lambda_l$,

$$\mathcal{K} \leq \mathcal{K}_{max} = \prod_{l=1}^{m} \lambda_l \tag{4.27}$$

For any layer $l \in \{1, 2, 3, ..., m\}$, the input f_{in}^l is bounded by $[F_{min}^l, F_{max}^l]$. $k_l = \lambda_l$ can be achieved by setting $f_c^{n_{\pi(1)}^l}$ and $f_c^{n_{\pi(\lambda_l)}^l}$ such that:

$$f_c^{n_{\pi(1)}^l} = F_{min}^l \text{ and } f_c^{n_{\pi(\lambda_l)}^l} = F_{max}^l \tag{4.28}$$

Hence the bound is tight for (Equation 4.27). The proof is complete.

Compared to a network with homogeneous neuron parameters, in which the upper bound of number of distinct memory pathways is λ_m, Lemma 4 indicates that heterogeneous network increases the maximum achievable number of distinct memory pathways in a feedforward SNN.

4.3.3 Skip-layer Connection

We show that adding skip-layer connection increases the upper bound of the number of memory pathways in a network with Lemma 5.

Lemma 5 *For a mMND network with m layers and $\{\lambda_1, \lambda_2, ...\lambda_m\}$ different neuron dynamics in each layer and a skip-layer connection made between layer l_a and l_b, s.t. $a, b \in \{1, 2, ...m\}$ and $(b - a) > 1$, the least upper bound of the number of memory pathways is $\prod_{i=1}^{m} \lambda_i + (\prod_{i=1}^{a} \lambda_i \cdot \prod_{i=b}^{m} \lambda_i)$*

Proof. Denote the mMND network with skip-layer connection between layer l_a and

layer l_b as P. Denote the set of all neurons in P as

$$S_P = \{n_1^1, n_2^1, n_3^1, ..., n_1^2, n_2^2, n_3^2, ...\} \tag{4.29}$$

where n_i^1 is a neuron in layer l_1, and n_i^2 is a neuron in layer l_2, etc. The activation state of a neuron $o_{n_i^j}$ can be denoted with binary values 0 and 1 with $o_{n_i^j} = 1$ representing that n_i^j receives input frequency that is above its cutoff frequency $f_c^{n_i^j}$.

The set of all possible neuron activation states O in P that generates non-zero network output feature vector can be partitioned into two subsets denoted as O_A and O_B.

The set O_A contains all states where the input frequency f_{in}^k to any layer l_k such that $a < k < b$ satisfies

$$f_{in}^k < f_c^{n_i^k} \;\; \forall\, i \in \{1, 2, 3, ..., \lambda_k\} \tag{4.30}$$

The set O_B contains all the remaining neuron activation states in O, where all layers receive input frequency higher than cutoff frequency of at least one neuron in each layer.

For all the states in O_A, no spike signal is sent from layer $b-1$ to layer b, since at least one layer between l_a and l_b generates no output. Hence, output from P is not affected if connections between layer l_i and l_{i+1}, such that $i \in \{a, a+1, ..., b-1\}$, are removed. Network P is therefore equivalent to network P' that has layers $\{l_1, l_2, ..., l_a, l_b, l_{b+1}, ..., l_m\}$ connected sequentially.

According to Lemma 4, it can be derived that the least upper bound of the number of distinct memory pathways in P' is $\prod_{i=1}^{a} \lambda_i \cdot \prod_{i=b}^{m} \lambda_i$.

Hence, for all states in set O_A, the least upper bound of the number of distinct memory pathways in P is also $\prod_{i=1}^{a} \lambda_i \cdot \prod_{i=b}^{m} \lambda_i$. For all states in set O_B, since the activation of neurons in the source layer of the skip-layer connection is already accounted for when considering layer l_a, the least upper bound of the number of distinct memory pathways is the same as network P that has no skip-layer connection, which is $\prod_{i=1}^{m} \lambda_i$ according to

Lemma 4.

For the set of memory pathways from states in O_A, denoted as M_A, and the set of memory pathways from states in O_B, denoted as M_B, the number of memory pathways of network P is

$$\mathcal{K} = |M_A \cup M_B| \tag{4.31}$$

Since

$$|M_A \cup M_B| \leq \prod_{i=1}^{m} \lambda_i + (\prod_{i=1}^{a} \lambda_i \cdot \prod_{i=b}^{m} \lambda_i) \tag{4.32}$$

,

$$\mathcal{K} \leq \prod_{i=1}^{m} \lambda_i + (\prod_{i=1}^{a} \lambda_i \cdot \prod_{i=b}^{m} \lambda_i) \tag{4.33}$$

.

From Lemma 5, the bound is tight for $|M_A| \leq \prod_{i=1}^{a} \lambda_i \cdot \prod_{i=b}^{m} \lambda_i$ and for $|M_B| \leq \prod_{i=1}^{m} \lambda_i$. It also satisfies that $M_A \cap M_B = \emptyset$, since all elements in M_A have $(m-(b-a-1))$ layers, and all elements in M_B have m layers. Hence the bound is tight for (Equation 4.33). The proof is complete.

4.4 Time-varying Function Approximation

A time-varying function f(t) can be approximated using piece-wise constant function, such as illustrated in Figure 4.2. It is therefore possible to approximate the time-varying function with a feedforward SNN by approximating each of the constant function with a memory pathway. In this section, we test the approximation capability of feedforward SNN for time-varying functions using this principle. The target functions to approximate has the form of:

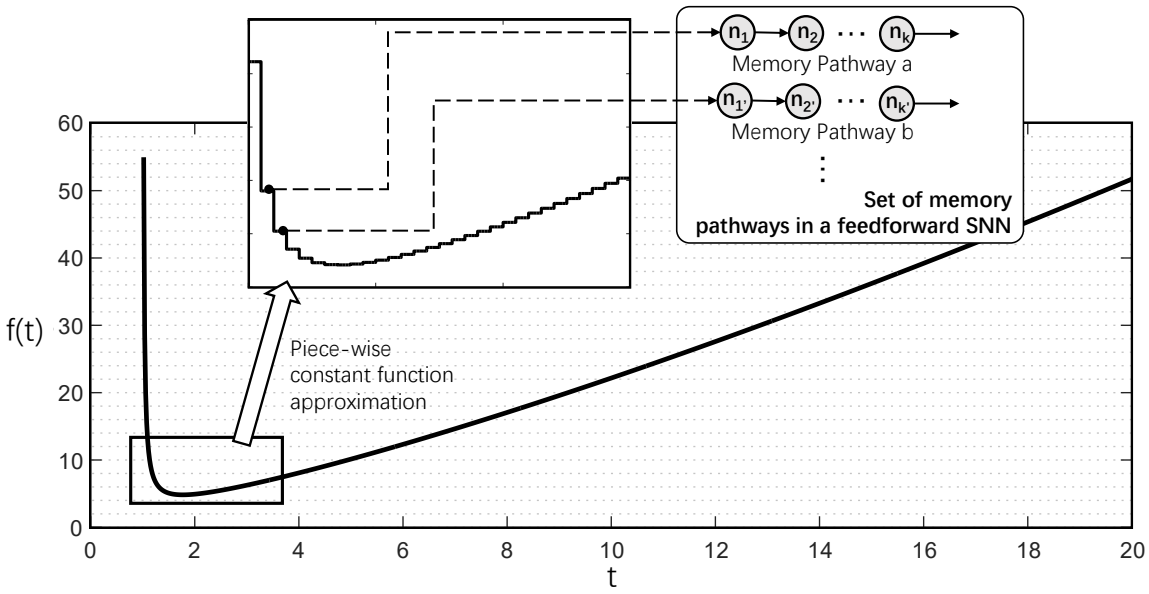


Figure 4.2: Using a set of memory pathways to map a piece-wise constant function for approximating a time-varying function.

$$f(x) = \frac{x^n}{x - m} \tag{4.34}$$

Here, x is variable; n and m are function parameters. For discrete-time simulation of the network, we approximate the target function with

$$x = t_{in} \text{ and } f(x) = t_{out} \tag{4.35}$$

where t_{in} is the input spike period and t_{out} is the output spike period. We test approximation performance of a small-scale feedforward SNN with 6 fully-connected layers, skip-layer connections $\{(2, 5), (3, 5)\}$ and 4 neuron dynamics. The network is trained with BPTT to minimize mean squared error (MSE) loss between the spike period of network output t'_{out} and target spike period t_{out}.

To evaluate the network's approximation performance, we construct 6 networks with the same structure as discussed above, and change each to have different trainable parameter numbers by scaling the layer size. For baseline comparison, networks with 6 layers, no

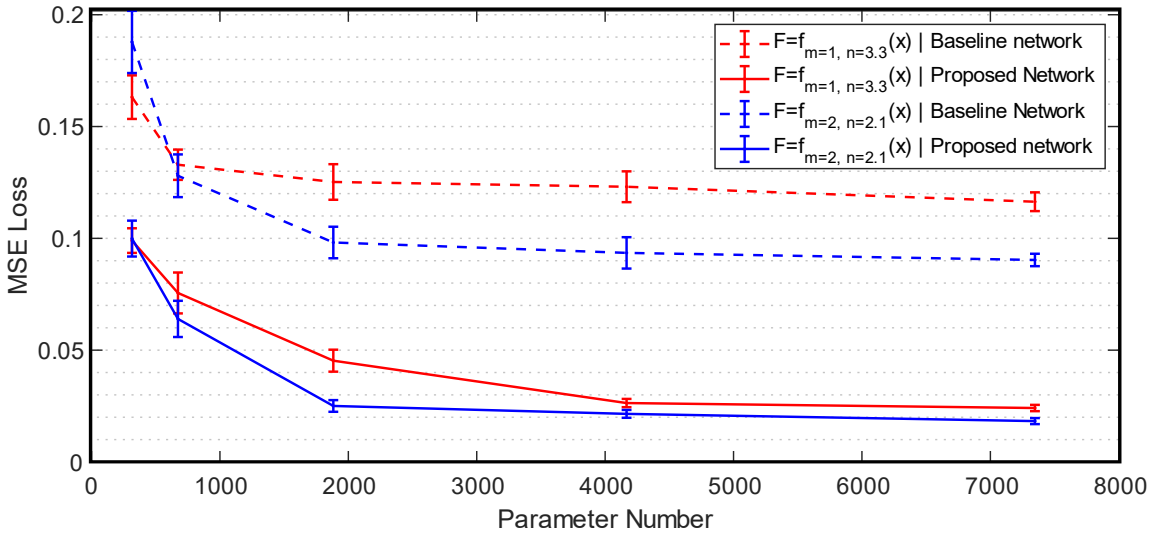


Figure 4.3: MSE loss vs. number of trainable parameters for the function approximation experiments.

skip-layer connection and using homogeneous neurons, configured to have the same trainable parameter numbers as the proposed networks, are tested. All networks are trained with BPTT. The loss function measures the difference between the network output spike trains and the target spike trains with MSE. Two sets of function parameters: $(m = 1, n = 3.3)$ and $(m = 2, n = 2.1)$ are tested for the target functions $f(x)$ on domain $[3, 10]$. The resulting MSE loss for different network scales are shown at the bottom of Figure 4.3. It can be observed that the proposed networks can approximate target functions with less error than the baseline networks at all network scales. The smallest tested networks have relatively high losses while performance increase quickly with more trainable parameters. The rate of performance improvement decreases when trainable parameter numbers is above 4000.

To understand the impact of the target function parameters to approximation performance of SNN, we test the baseline and proposed network with 4167 trainable parameters for different pairs of function parameters m and n. The resulting MSE loss is shown in Figure 4.4 (a) and in Figure 4.4 (b). It can be observed that for all m and n value pairs, the proposed network can achieve lower loss than the baseline network. Another observation is that, there is no clear correlation between the value of m and approximation error. On

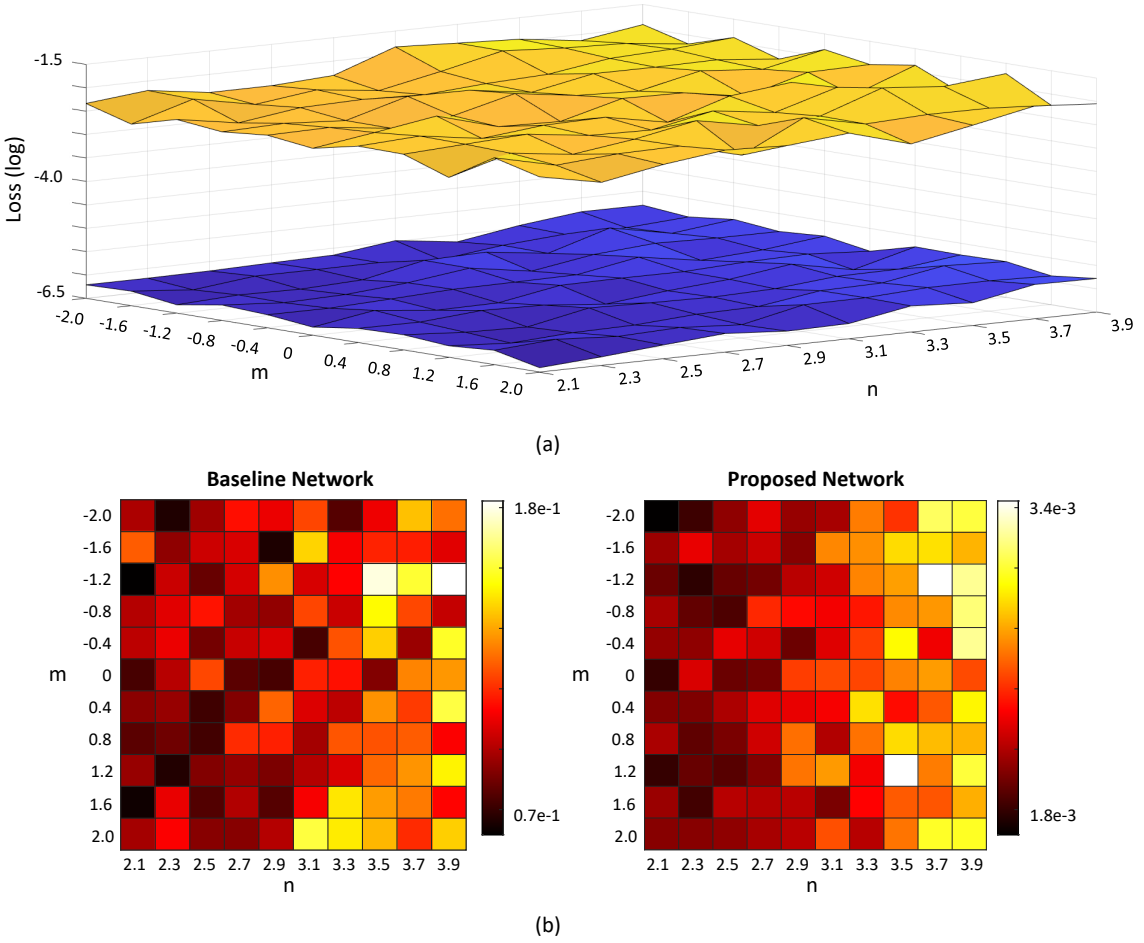


Figure 4.4: (a) MSE loss (log) of baseline network (top) and the proposed network (bottom) for approximating functions with different parameters m and n. (b) Heat plots of MSE loss for approximating functions with different parameters m and n.

the other hand, for higher values of n, the approximation error generally increases for both baseline and the proposed networks.

4.5 H-SNN Optimization Using Approximation Theory

In this section, we discuss the process to optimize H-SNN as inspired by the developed approximation theory for feedforward SNN with an efficient dual-search-space Bayesian optimization process. First, network templates for H-SNN, trainable with BPTT and STDP, are defined, based on which specific network configurations, including network structure and neuron parameters, can be optimized with the proposed Bayesian optimization process.

4.5.1 Network Template for BPTT Training

For H-SNN that can be trained with BPTT, the network template is shown in Figure 4.5. Each multi-neuron-dynamic layer, which can be either convolutional or fully connected, has heterogeneous neuron dynamics to generate each feature map. There are two types of synapses between layers: transferred synapses marked as black dashed arrows and learned synapses marked as red solid arrows. The conductance of learned synapses is optimized by the BPTT algorithm during training, and the transferred synapses have the same conductance as the learned synapses from the same pre-synaptic neuron. For example, the synapses connecting neurons with dynamic d_m to neurons with dynamic $\{d_2, d_3, d_4, ..., d_{m-1}\}$ in the next layer have conductance transferred from synapses connecting neurons with dynamic d_m to neurons with dynamic d_1.

The skip-layer connection is implemented with the output spike matrix from the source layer concatenated to the original input spike matrix of the target layer. The skip-layer connection has the same implementation as the regular connection between consecutive layers, with both learned and transferred synapses (Figure 4.5). The last layer of the network is a full-connected layer with homogeneous dynamic trained to generate prediction labels from the output feature map of the last multi-neuron-dynamic layer.

4.5.2 Network Template for STDP Learning

The template of networks trainable with STDP is different from the one for BPTT, as shown in Figure 4.6. Here, each multi-neuron-dynamic layer contains a learner module and a memory module. Learner modules use homogeneous neuron dynamic that is suitable for STDP learning, and memory modules consist of neurons with different dynamics. Similar to BPTT training, there are two types of synapses: transferred synapses and learned synapses. Between two layers, memory modules are connected with transferred synapses and memory modules are connected to learner modules with learned synapses. Leaner modules between layers are not directly connected.

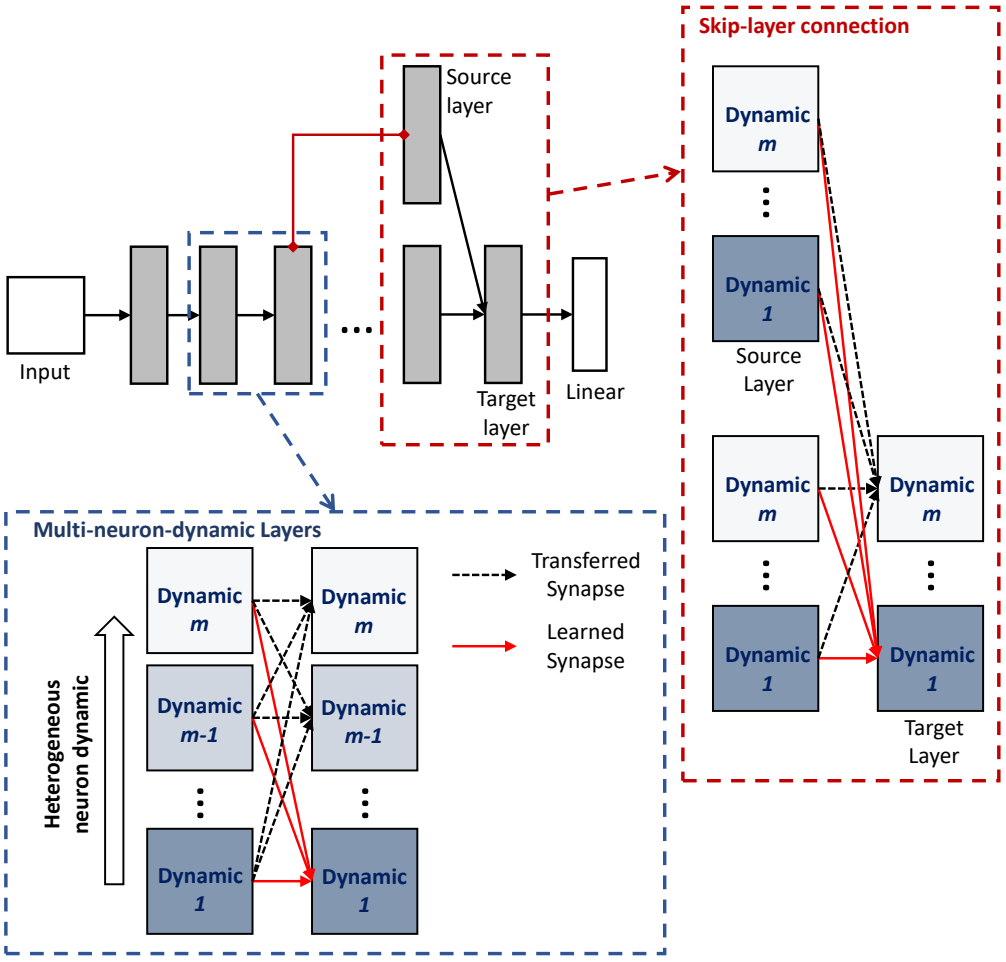


Figure 4.5: H-SNN with multiple neuron dynamics and skip-layer connections trainable with BPTT; each multi-neuron-dynamic layer contains a set of neuron dynamics from d_1 to d_m; neurons with different dynamics are connected either with synapses with trained conductance (learned synapses) or synapses with transferred conductance from learned synapses.

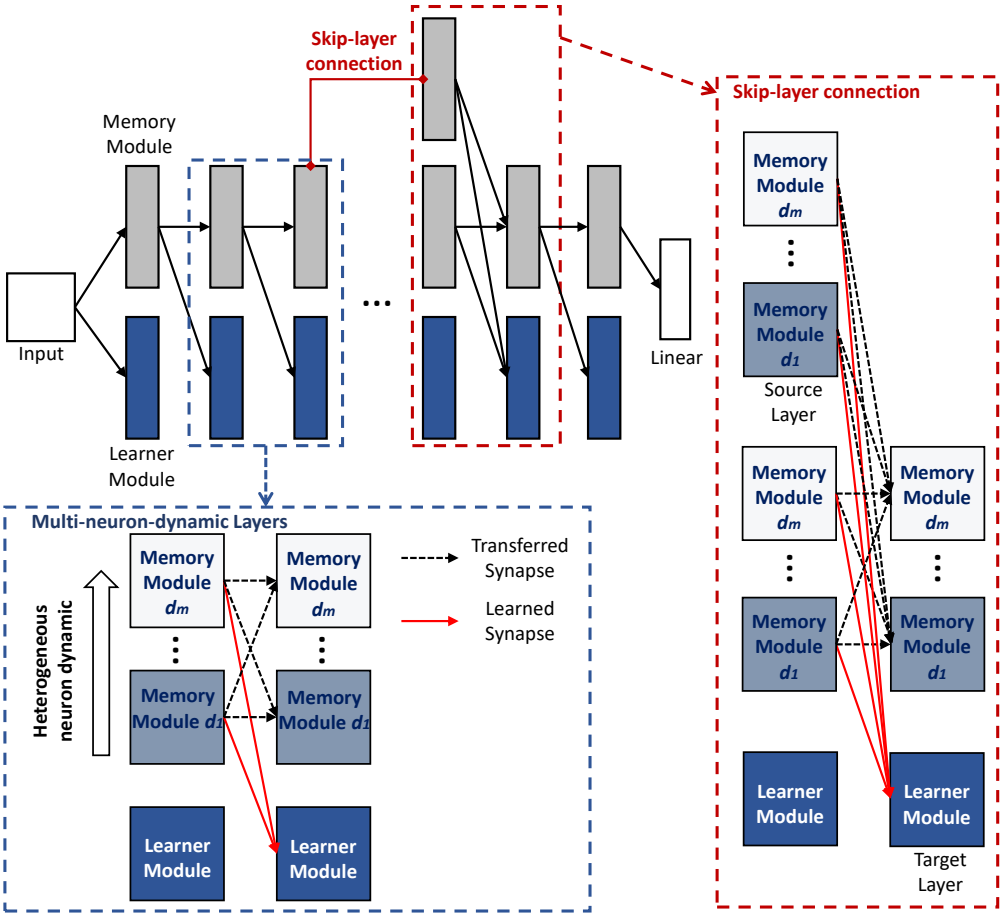


Figure 4.6: H-SNN with multiple neuron dynamics and skip-layer connections trainable with STDP.

STDP training proceeds as a layer-by-layer process. During training of the first layer, conductance of synapses connecting neurons in memory module to neurons in learner modules, referred to as learned synapses, is learned with STDP using all training data without labels. Then, the learned conductance is transferred to the corresponding transferred synapses. The memory module is then used to perceive input patterns and generate spikes during training of the next layer. This lay-by-layer process is repeated until the layer before the final layer finishes learning. The final linear layer is then fine-tuned using stochastic gradient descent (SGD) based on spike frequency array from the last multi-neuron-dynamic layer generated based on the labeled data. Skip-layer connection is implemented by connecting the memory module of the source layer to the target layer. The connections are

made with the two types of synapses and follow the same training process as the consecutive layers.

4.5.3 Implementation of Heterogeneous Conv-SNN

Multi-neuron-dynamic (MND) networks with convolutional layers can be considered a scaled-up version of the mMND network, where each neuron dynamic now contains a matrix of neurons that receive input features from different spatial locations. To implement heterogeneous Conv-SNN, first consider a regular Conv-SNN layer with no heterogeneity. The spiking neuron matrix has dimension $\{W, H, D\}$, where D is the depth of the layer. The convolution filter has dimension $\{C, w, h, D\}$ where C is the number of input channels and D is the number of output channels.

Based on this, a heterogeneous layer l can be constructed, by concatenating heterogeneous neuron matrices with the same W and H along layer depth. The resulting spiking neuron matrix has dimension $\{W, H, \lambda D\}$, where λ is the number of neuron dynamics. Hence, layer depth $i \in \{k+1, k+2, k+3, ..., k+D\}$ have the same neuron parameters, where $0 \leq k \leq \lambda - 1$ is an integer representing the index of neuron dynamics. The convolution filter has the same dimension as the regular Conv-SNN layer: $\{C, w, h, D\}$, which is shared by layer depth with different k values. During forward pass of this layer, a convolution operation is applied to generate an input signal matrix with dimension $\{W, H, D\}$. For each value of k, neurons in depth $\{k+1, k+2, k+3, ..., k+d\}$ are simulated based on the neuron parameters for such neuron dynamic index k using the signal matrix as input. For the next convolutional layer l' receiving input from layer l, the filter dimension is $\{\lambda D, w', h', D'\}$ with λD input channels.

4.6 Dual-search-space Bayesian Optimization

With the addition of multiple neuron dynamics and skip-layer structure, the new H-SNN models contain a relatively high number of hyperparameters, which could lead to difficulty

in the manual tuning process for better performance. We therefore develop a dual-search-space Bayesian optimization process as an algorithmic approach to network optimization. Bayesian optimization uses Gaussian process to model the distribution of an objective function, and an acquisition function to decide points to evaluate. For data points in a target dataset $x \in X$ and the corresponding label $y \in Y$, an SNN with network structure $\mathcal{V}$ and neuron parameters $\mathcal{W}$ acts as a function $f_{\mathcal{V},\mathcal{W}}(x)$ that maps input data x to predicted label $\tilde{y}$. The optimization problem in this work is defined as

$$\min_{\mathcal{V},\mathcal{W}} \mathcal{P} \quad \text{where } \mathcal{P} = \sum_{x \in X, y \in Y} \mathcal{L}(y, f_{\mathcal{V},\mathcal{W}}(x)) \tag{4.36}$$

$\mathcal{V}$ contains the number of layers N_{layers}, the number of memory dynamics $N_{dynmaic}$ and skip-layer connection configuration variables N_{skip}, L_{start} and L_{end}, each controlling the number of skip-layer connections, the first layer and last layer to implement skip-layer connections. All of the values are discrete. $\mathcal{W}$ contains the values for a, τ_m and R_m in (Equation 2.1), which are continuous. We separate the discrete and continuous search spaces by implementing a dual-search-space optimization process, as shwon in Figure 4.7. In this process, the network structural design is first optimized with fixed, manually selected neuron parameters. After the network structure optimization is finished, neuron parameters are optimized based on the selected network structure. This separates the search space of network structure, which is discrete, from the continuous search space of neuron parameters to reduce time consumption for the Bayesian optimization process.

It is also necessary to set constraints to the search spaces to guarantee the chosen configurations are feasible for network construction. For example, the target layer index of a skip-layer connection needs to be smaller than the number of layers in the network. To achieve Bayesian optimization with constraints, we implement a modified expected improvement (EI) acquisition function similar to the one shown by Gardner [65], which uses a Gaussian process to model the feasibility indicator due to its high evaluation cost. In this work, since the constraint function can be explicitly defined, we use feasibility indicator

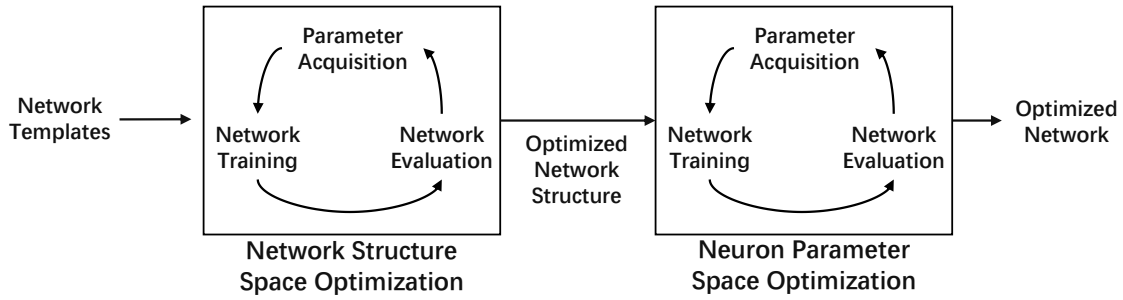


Figure 4.7: The proposed dual-search-space Bayesian optimization process.

that is directly evaluated. The modified EI function is defined as:

$$I_c(\mathbb{W}) = \Delta(\mathbb{W}) \cdot \max\{0, \mathcal{P}(\mathbb{W}) - \mathcal{P}(\mathbb{W}^+)\} \tag{4.37}$$

where $\mathbb{W}$ is the network configuration containing $\mathcal{W}$ and $\mathcal{V}$. $\mathbb{W}^+$ is the test point that provided the best result. $\Delta(\mathbb{W})$ is the explicitly defined indicator function that takes the value of 1 when all constraints are satisfied and 0 otherwise.

4.7 Experiments

4.7.1 Experiment Settings

Datasets tested in the experiment include the DVS Gesture dataset [63], which is an event-based human gesture classification dataset captured by DVS cameras, and the N-Caltech101 [66], which is an event-based version of the Caltech101 dataset. The proposed method is also tested for MLP-style SNN on the sequential MNIST dataset, in which the original MNIST images are presented row-by-row sequentially. We also vary the amount of *labeled data used during training* ranging from using 100% labeled data for training down to 10% labeled data (30% for N-Caltech101) during training. Note, during STDP training networks always uses the entire but *un-labeled* training dataset; however, only the fraction of the labeled data is used for supervised fine-tuning of the last layer. Comparison is made for DVS Gesture and N-Caltech101 with prior works including ConvLSNN, which is a combination

of convolutional SNN and recurrent SNN with long and short-term neurons trained with BPTT [67], DECOLLE [16], which uses surrogate gradient to train a convolutional feed-forward SNN, HATS [68], which implements time surfaces and SVM for classification and the original H-SNN as presented in chapter 3, which uses STDP to train a convolutional SNN with two neuron dynamics.

4.7.2 Optimization Process

During the first stage of the dual-search-space optimization process, the parameters to optimize include: N_{layer}, L_{start}, L_{end}, N_{skip}, $N_{dynamic}$, all of which are positive integers. Specifically, N_{layer} is the number of convolutional layers. For skip-layer connection, there are three configuration parameters to optimize: starting layer L_{start}, which is the source layer of the first skip-layer connection; ending layer L_{end}, which is the target layer of the last skip-layer connection; skip-layer connection number N_{skip}, which defines how many connections to implement. The source layer of the N_{skip} skip-layer connections are placed evenly between L_{start} and L_{end}, each with skip length of $\lfloor (L_{end} - L_{start})/N_{skip} \rfloor$, in case $((L_{end} - L_{start})/N_{skip}) \neq \lfloor (L_{end} - L_{start})/N_{skip} \rfloor$, the value of L_{end} is reduced to the maximum value that satisfies $((L_{end} - L_{start})/N_{skip}) = \lfloor (L_{end} - L_{start})/N_{skip} \rfloor$. For heterogeneity, the number of different dynamic $N_{dynamic}$ in all layers are optimized jointly. The constraint for the parameters is that, $N_{layer} \in [4, 15]$, $2 \leq L_{start} < (N_{layer} - 1)$, $(L_{start} + 1) < L_{end} \leq N_{layer}$, and $0 \leq N_{skip} \leq (L_{end} - L_{start})/2$ and $N_{dynamic} \in [1, 10]$. The manually configured neuron parameters are, $\tau_m = 100$ and $R_m = 300$ for all neuron dynamics, and $a \in [-30, -5]$ is distributed evenly for each neuron dynamics.

Due to the exponential increase of search space with the number of neuron dynamics in each layer, it is highly inefficient to search for every neuron parameters in each dynamic. In the second stage of the optimization process, we choose to apply Bayesian optimization for the parameter a of each neuron dynamic separately, while τ_m and R_m are optimized jointly with the same values shared by all neuron dynamics. a values are taken to the

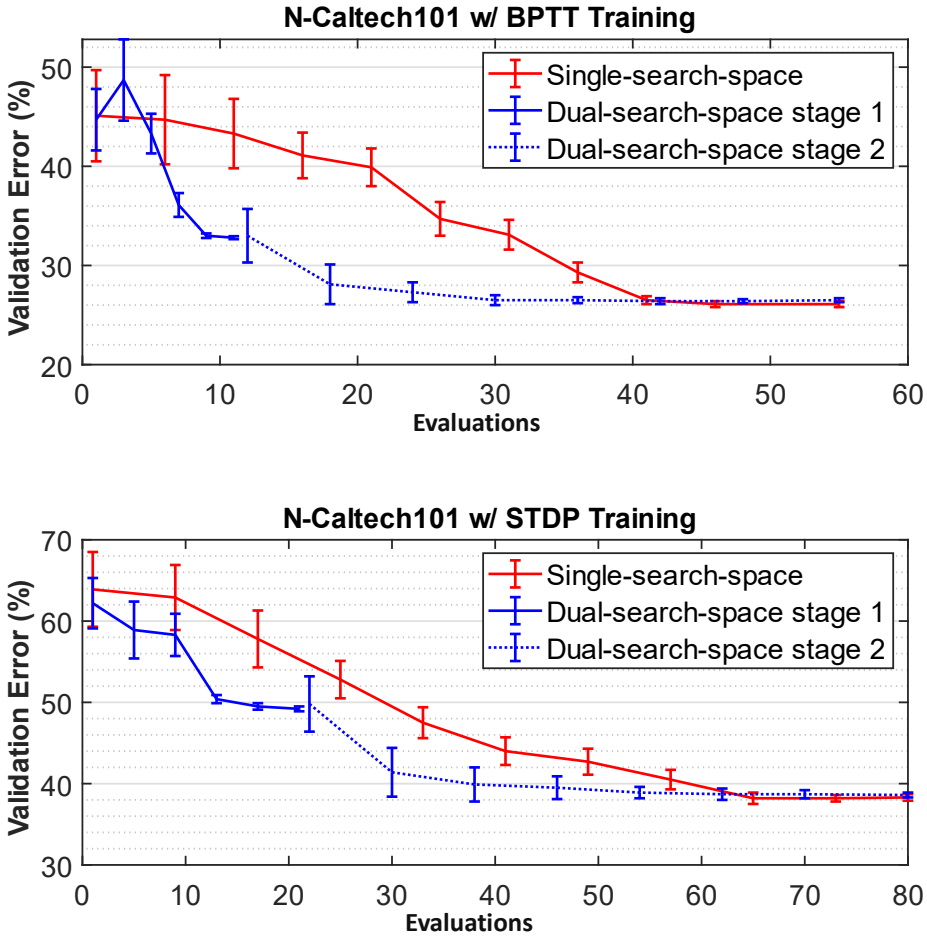


Figure 4.8: Validation error over optimization evaluations for the proposed dual-search-space Bayesian optimization compared to the normal single-search-space Bayesian optimization.

precision of 10^0, and τ_m and R_m values are taken to the precision of 10^1. The constraints are $a \in [-30, -5]$, $\tau_m \in [50, 200]$ and $R_m \in [200, 400]$. The value of t_{nd} for all networks is set to 1. The parameters of each optimized networks are shown in Table 4.2. Note, the skip-layer connections are listed as source and target layer pairs.

4.7.3 Effect of Dual-search-space Bayesian Optimization

We compare the proposed dual-search-space Bayesian optimization with regular Bayesian optimization using a single search space for network validation error over 5 runs. The result from the N-Caltech101 dataset is shown in Figure 4.8. It can be observed that the

two optimization approaches achieve similar minimum validation error after convergence. By separating the search spaces, the proposed optimization process reaches convergence faster than regular single-search-space optimization. It is also worth noting that, between the two stages in the optimization process for BPTT training, the first stage accounts for more reduction in validation error than the second stage. This indicates that optimizing network structure causes more impact to BPTT training than optimizing neuron parameters, which is potentially due to the reason that network structures more heavily affects the number of memory pathways in the network than neuron parameters. On the other hand, for STDP training where learning behavior is sensitive to the dynamic of spiking neurons, the reduction of validation error is more equally shared between the two optimization stages. Over the 5 runs, among all network configurations achieved after the dual-search-space optimization converges, we compare the configuration with the lowest number of trainable parameters against baseline models. The specific configurations for the optimized networks are listed in Table 4.2. It can be observed that for BPTT algorithm, the optimized networks have more layers than the STDP trained networks, and the optimal values found for neuron parameters are highly distinct for the two training methods.

Table 4.2: Configuration of optimized network models

Network	Conv. Layer Number	Skip-layer Connection	Number of Different Neuron Dynamics and a	Neuron Parameters τ_m	Neuron Parameters R_m
BPTT, Gesture	9	(2,7)	4, (-24,-17,-12,-9)	120	340
BPTT, N-Caltech	12	(2,5), (5,8), (8,11)	5, (-23,-16,-14,-11,-8)	70	300
STDP, Gesture	6	(2,4), (4,6)	4, (-26,-24,-15,-9)	110	260
STDP, N-Caltech	8	(3,5), (5,7)	6, (-21,-19,-17,-13,-9,-7)	140	240

4.7.4 Ablation Studies

To investigate the effect of using multiple neuron dynamics, for both BPTT and STDP training, we apply the same dual-search-space Bayesian optimization process for networks that have homogeneous neuron dynamic for the same number of evaluations as the proposed design. Such networks are referred to as *Homogeneous-BPTT* and *Homogeneous-STDP*.

Table 4.3: Ablation studies of optimization approaches: configuration of tested networks and accuracy results (%)

Model	Heterogeneity	Skip-layer	DVS Gesture	N-Caltech101	S-MNIST
Homogeneous-BPTT	N	Y	95.0	65.3	95.5
No-skip-layer-BPTT	Y	N	96.5	63.5	94.8
H-SNN Optimized (BPTT)	Y	Y	98.0	71.2	97.3
Homogeneous-STDP	N	Y	91.3	37.0	94.3
No-skip-layer-STDP	Y	N	93.1	51.9	95.5
H-SNN Optimized (STDP)	Y	Y	96.6	58.1	96.1

Similarly, to study the contribution to performance gain from skip-layer connections, the Bayesian optimization process is used for network templates without skip-layer connections. The optimization process runs for the same number of evaluations as the proposed design and the networks are referred to as *No-skip-layer-BPTT* and *No-skip-layer-STDP*.

From the results shown in Table 4.3, it can be observed that compared to baselines, the proposed networks achieve the best accuracy for all datasets. Specifically, when the network contains only homogeneous neuron dynamic, the performance of STDP trained network is noticeably lower than the proposed method for DVS Gesture and N-Caltech101 datasets, while removing skip-layer connections shows less impact. For BPTT training, using heterogeneous network and skip-layer connections show different level of benefit for each dataset. For sequential MNIST which has less complexity, the improvement from using heterogeneous neurons and skip-layer connections is not as significant.

4.7.5 Comparison with Prior Works

DVS Gesture Accuracy results and the number of trainable parameters for the tested models are listed in Table 4.4. With 100% labels available during training, the proposed network trained with BPTT demonstrates higher accuracy than all tested networks with the least number of trainable parameters. The proposed network trained with STDP has slightly lower accuracy than ConvLSNN and DECOLLE when 100% labels are used. With reduced label number, STDP training shows the advantage of unsupervised learning, and

Table 4.4: Accuracy (%) and trainable parameter number of tested models for DVS Gesture dataset with varying amount of training labels

Model	Labeled Data % In Training 100%	50%	30%	10%	Parameter No.
ConvLSNN [67]	97.1	95.3	92.0	84.3	2.9M
DECOLLE [16]	97.5	95.0	91.2	83.9	1.3M
HATS [68]	95.2	94.1	91.6	83.7	-
H-SNN Original	96.2	95.8	93.7	88.2	0.74M
H-SNN Optimized (STDP)	96.6	**96.0**	**94.1**	**91.2**	0.81M
H-SNN Optimized (BPTT)	**98.0**	95.3	91.1	82.4	1.1M

Table 4.5: Accuracy (%) and trainable parameter number of tested models for N-Caltech101 with varying amount of training labels

Model	Labeled Data % In Training 100%	70%	50%	30%	Parameter No.
ConvLSNN [67]	63.1	58.7	51.3	45.4	3.0M
DECOLLE [16]	66.9	61.9	56.2	50.6	2.0M
HATS [68]	64.2	61.0	54.3	48.8	-
H-SNN Original	42.8	41.9	37.0	34.6	1.7M
H-SNN Optimized (STDP)	58.1	57.8	**57.2**	**54.6**	1.4M
H-SNN Optimized (BPTT)	**71.2**	**65.4**	56.0	52.5	1.7M

outperforms all baselines including H-SNN, which also uses STDP unsupervised learning. The accuracy improvement from the STDP trained network is more significant with less labels. The proposed STDP network also has the advantage of less trainable parameters compared to supervised models.

N-Caltech101 As shown in Table 4.5, the proposed network trained with BPTT outperforms all other tested networks with both 70% and 100% training labels. It also has less number of trainable parameters than all baselines. The un-supervised learning models i.e., the original H-SNN and the proposed network with STDP, show considerably lower performance, compared to what was observed for DVS Gesture, than the supervised networks when 100% labels are available. On the other hand, the proposed network with STDP learning shows better performance than the original H-SNN at all label amounts, and outperforms all other networks when available labels are below 50%. Its trainable parameter

number is also the lowest amoung all tested networks.

4.8 Network Spiking Activity

To investigate neuron spiking activity in the proposed network, for the function approximation task as described in section 4.4, with parameters $(m = 1, n = 2.3)$ and a randomly selected approximation point $f(x = 19)$, we plot the timing of neuron spike at the last layer over training epochs in Figure 4.9 (a). It can be observed that the network initially generates spikes at widely distributed timings. After the first 50 training epochs spike timing starts to converge and remains stable at around 200 epochs. The final output spike period is $t'_{out} = 48$, which matches the target output period.

Next, we investigate spiking activity of the BPTT trained network as described in Section subsection 4.7.5 for N-Caltech101 classification task. Three different test data points are presented to the network, and the spikes from neurons in the first depth of layer 8 and neurons in the final layer, are recorded and shown in Figure 4.9 (b). Here, (i), (iii) and (v) are from layer 8; (ii), (iv) and (vi) are from the final layer. (i) and (ii) are from the observation of a data point with class label "5"; (iii) and (iv) are from the observation of another data point also with class label "5"; (iii) and (iv) are from the observation of a data point with class label "1".

It can be observed that neurons in layer 8 exhibit similar activity in (i) and (iii) as the network is presented with data points from the same class. (ii) and (iv) show that most spiking activity is from the neuron with index 5, indicating correct prediction for the two data points. When a data point from a different class is presented, spiking activity in (v) shows different patterns than those in (i) and (iii). In the final layer, neuron with index 1 generates the most spike, leading to correct prediction. However, for this data point, neuron with index 58 is also generating a considerable number of spikes, indicating that the network is more likely to mis-classify this data point, compared to the other two tested data points.

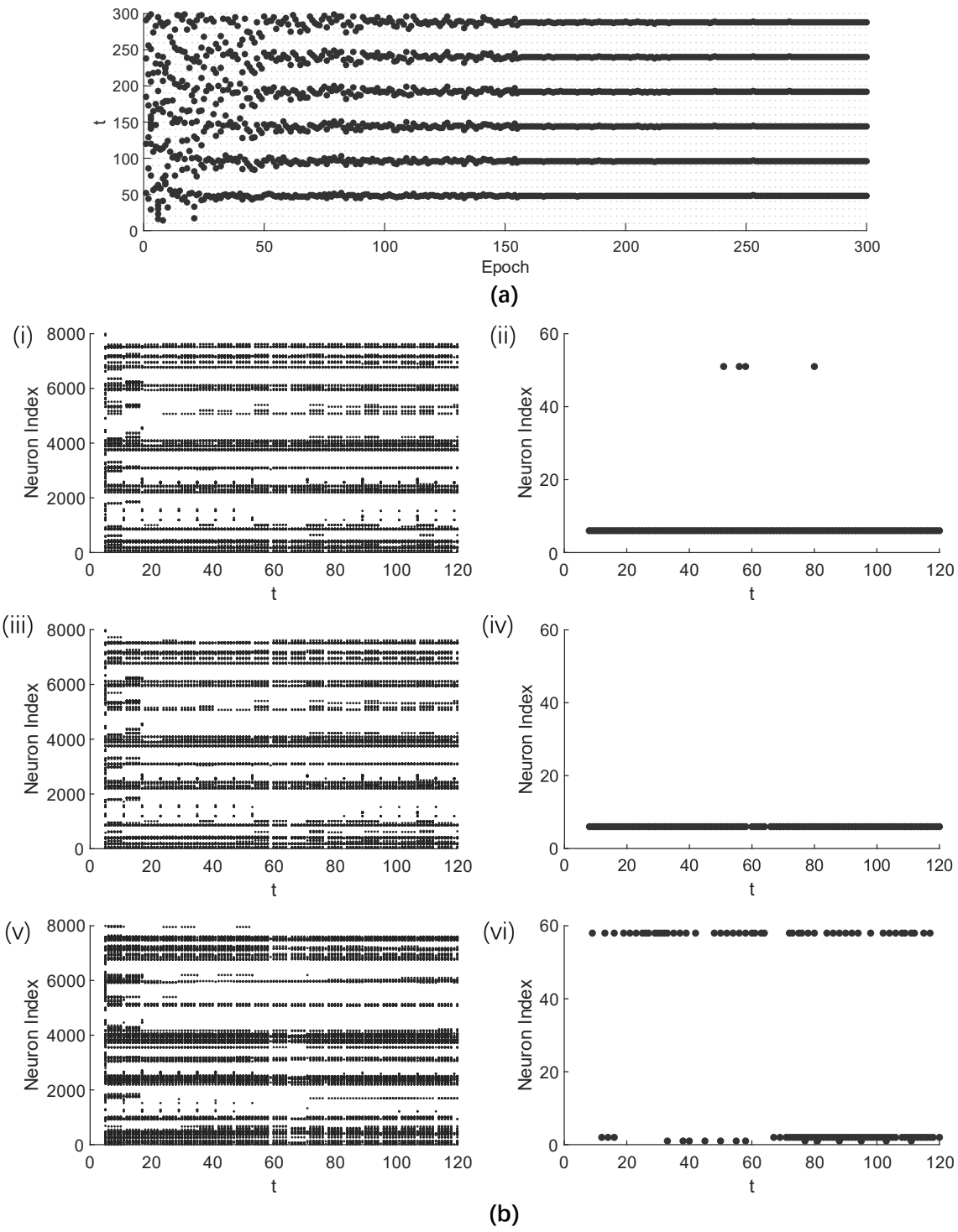


Figure 4.9: (a) Raster plot for function approximation experiment; t is the simulation time step. (b) Raster plot for the proposed SNN trained for N-Caltech101 with BPTT; t is the simulation time step; (i), (iii) and (v) are from layer 8; (ii), (iv) and (vi) are from the final layer.

4.9 Summary

To achieve better understanding of the properties of feedforward SNN when used in spatiotemporal data processing, in this chapter, we develop a theoretical basis for the capability of feedforward SNN to approximate mapping functions of input-to-output spike sequence pairs. On top of this, we analytically show how heterogeneity of neuron dynamic and skip-layer connections can improve the function approximation ability of H-SNN. Based on the theoretical results, the two structural designs: neuron heterogeneity and skip-layer connections, are jointly implemented for two network architectures trainable with BPTT and STDP, respectively. Compared to baseline models, the number of hyperparameters to tune could be relatively higher for the proposed design. We therefore develop an efficient dual-search-space Bayesian optimization process to search for H-SNN configurations that can achieve lower validation error for given tasks. Experimental results show that the learning performance of H-SNN can be successfully improved using the proposed methods. The optimized H-SNN shows advantages over baseline models in learning spatiotemporal datasets of different complexity.

CHAPTER 5

EVENT-DRIVEN SNN PROCESSING OF SPATIOTEMPORAL DATA

In the previous chapters, experimental results and theoretical basis of using H-SNN for processing spatiotemporal data are presented, and the primary focus is on the network's learning performance. In this chapter, we present a fully event-based processing pipeline using neuromorphic vision sensors as the input source and H-SNN as the data processing model aiming to improve computation efficiency. This design is named SPEED [69]: a processing method for spiking neural network with event-driven learning and inference of event camera data. SPEED has the potential to achieve high throughput and low latency computer vision data processing. Figure 5.1 shows an overview of the proposed image processing pipeline: an event camera captures brightness-change events in the original scene and data is transmitted to an on-board event processing engine in its native, event-by-event format. The processing platform, such as a CPU or GPU, runs unsupervised STDP learning as well as object classification by processing the input events in real-time. The proposed event-driven processing method significantly improves H-SNN learning and inference speed, and delivers higher robustness when the network has reduced-precision.

In the following sections, we first discuss the background on event-based vision sensor and existing processing methods for spatiotemporal dataset. Details of the proposed event-driven SNN processing method is then presented and experimental results are shown.

5.1 Background on Event-based Spatiotemporal Data Processing

5.1.1 Neuromorphic Vision Sensor

Neuromorphic vision sensor [70] is a type of event-based vision sensors inspired by biological retinas. An example of such sensors is the dynamic vision sensor (DVS) used in

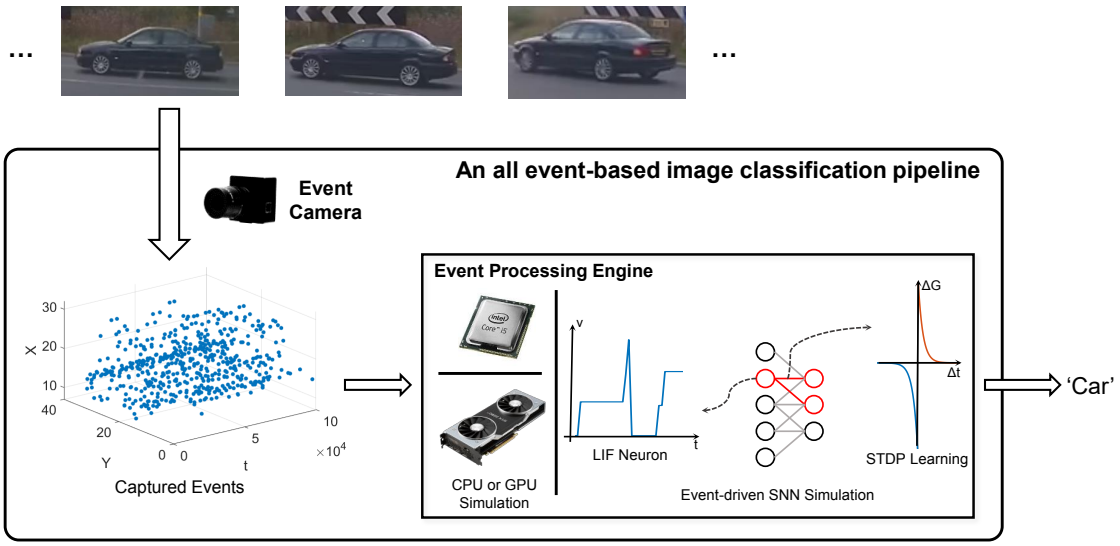


Figure 5.1: The all event-based classification pipeline proposed in this chapter, which consists of event camera as input source and CPU or GPU running event-driven SNN processing. The two polarities of the output events, representing increase and decrease of brightness, are marked with red and blue dots, and separated into two input channels to the network.

event cameras [71]. Event camera operates by capturing brightness changes (increase and decrease) in a scene as asynchronous events. Compared to conventional image sensors that capture frame-based data, neuromorphic vision sensors have advantages including higher temporal resolution [72] and higher dynamic range [73]. This makes event camera an ideal device for applications such as space-based platforms [25], driver assistance [26] and robot navigation [74]. Processing event camera data has been achieved with different approaches [27, 68, 75]. In particular, spiking neural network (SNN) [76] is an appealing solution for its spike-based information representation and transmission that are inherently suitable for event-based data processing, and the unsupervised learning capability [2] that enables learning of unlabeled data.

5.1.2 Conventional Machine Learning

As the recent development of artificial neural networks (ANN) offered state-of-the-art performance for frame-based visual data, applying similar approaches to process neuromor-

phic vision sensor data has received less attention. Existing methods for learning event-based visual information with conventional machine learning models can be categorized into two types of approaches. The first type aims to convert the event based input to another representation that is similar to frames, and train conventional models based on the converted data, such as the work by Gehrig et al. [27] where events are mapped to a grid-based representation. Another example is DART by Ramesh et al. [28], in which a log-polar structure is used for conversion. Such approaches take the advantage of well studied methods for processing frame-based images, but requires aggregation of events over a period of time thus do not fully utilize the sparsity of event-based input. The second type focuses on designing networks that are event driven to better match the asynchronous property of event camera output. Examples include work by Cannici et al. [29], and by Messikommer et al. [30], where networks based on event-driven convolution operations are shown. Such networks are asynchronous in nature, thus providing better efficiency for processing data from event cameras. While promising results have been achieved with both types of approaches, few has the capability of unsupervised learning.

5.1.3 Spiking Neural Network

Compared to conventional machine learning approaches, using SNN to process neuromorphic vision sensor data has several advantages. First, SNN is capable of transmitting information and achieve learning using spike signals, which are asynchronous events similar to the output from neuromorphic vision sensors. Such alignment has the potential to achieve an all-event-based neuromorphic vision system. Another advantage of SNN is its ability to learn the temporal correlation of spiking events [32] based on causality. This can be achieved with an algorithm called spike-timing-dependent plasticity (STDP) [2]. Since the STDP learning rule is unsupervised, it is possible for SNN to learn the spatial or temporal correlation from unlabeled data. This capability is useful when the amount of labeled training data is limited. Event-based data representation in SNN simulation is demonstrated to

Table 5.1: Comparison of different methods

Network	Model & Learning Rule	Input Type	Event-driven Process
SCRNN[75]	Recurrent SNN (BPTT)	Frame	No
SpArNet[55]	Converted SNN (SGD)	Frame	Yes
[81]	Converted SNN (SGD)	Frame	No
H-SNN Original	Conv. SNN (STDP)	Frame	No
H-SNN w/ SPEED	Conv. SNN (STDP)	Event	Yes

have improved efficiency while maintaining accuracy [77]. Simulating SNN with event-driven approach has been shown in many prior works, but most cannot be directly used for learning of neuromorphic vision sensor data. For example, Lytton et al. [78] presents a variable-step integration method for local neuron update and Naveros et al. [79] implements a combined look-up table for high complexity network simulation. The two prior works show improved spiking neuron simulation efficiency but do not consider SNN learning. NEVESIM [80] is an event-driven SNN simulator with the support of Hebbian learning, but its neuron simulation and learning process are not designed for network architectures optimal for vision-based data, such as convolutional SNN.

While some recent work, such as shown in Table 5.1, has shown promising results on using SNN to perform classification for event camera data, many of them do not consider unsupervised learning or event-driven simulation. For example, Xing et al. [75] shows SNN with convolutional and recurrent connection for recognition of hand gesture captured by event camera. Learning is achieved with temporal back-propagation, which is a supervised training method also used in several other work [47, 82] for event-based data. Another approach is to convert SNN from stochastic gradient descent (SGD) trained CNN as shown by Massa et al. [81]. Besides using supervision, the network operations in those models are not event driven, making them inefficient for processing event camera input. Khoei et al. [55] shows event-driven SNN converted from CNN, but the input is frame-based data instead of sparse events, and the network is trained with supervision. STDP based

unsupervised learning of spatiotemporal data can be achieved[83, 59], but the networks are not event driven.

5.2 The Proposed Event-driven SNN Processing Method

In this section, based on the LIF neuron model and STDP learning rule, we present details of the proposed event-driven inference and unsupervised learning for event-based input.

5.2.1 Neuron and Synapse Model

The networks considered here are feedforward SNN in which leaky integrate-and-fire (LIF) neurons are connected with synapses. Two neurons connected with a synapse are called pre-synaptic and post-synaptic neurons, and the strength of connection is determined by synaptic conductance. For a synapse with conductance G, the change in G given the time difference between pre-synaptic and post-synaptic spikes $\Delta t = t_{post} - t_{pre}$, is determined with the STDP algorithm as reproduced here:

$$\Delta G_p = \alpha_p e^{-\Delta t(G-G_{min})/(\tau_{pot}(G_{max}-G_{min}))}$$

$$\Delta G_d = \alpha_d e^{-\Delta t(G_{max}-G)/(\tau_{dep}(G_{max}-G_{min}))}$$

Here, ΔG_p and ΔG_d are magnitude of LTP and LTD operation, respectively. α_p and α_d are parameters that controls the magnitude of LTP and LTD operation. τ_{dep} and τ_{pot} are time constant parameters. G_{max} and G_{min} are hyperparameters tuned based on specific network configurations.

5.2.2 Event-driven Neuron Simulation

In a simulation process with global updates to all neurons at each time step, referred to in the rest of the chapter as discrete-time simulation, the value of spiking neuron's membrane

potential can be directly calculated with the current value and the change over each timestep using differential equation defined by Equation 2.1. The data structure for discrete-time simulation of spiking neurons in a STDP learned network thus requires three variables: *membrane_potential*, *spike_timer* which keeps the elapsed time from last spike for STDP process, and *inactive_timer* which keeps the remaining time of an inactive neuron. Inactivation occurs when a neuron receives inhibitory signal from other spiked neurons or enters refractory period. *Membrane_potential* is updated each time step, and *inactive_timer* and *spike_timer* are also updated if needed.

When it comes to event-driven neuron simulation for sparse input, updating the state of each spiking neuron at each time step is inefficient due to the low activity inside the network. Ideally, only neurons that receive input spikes are updated. To achieve learning and inference with this principle, the event-driven spiking neuron is designed with a data structure that contains four variables: *last_membrane_potential*, *inactive_end_time*, *last_update_time* and *last_spike_time*. The last three variables keep track of the exact timestamp of the end of current inactive period, the timestamp of the last update to the neuron and the timestamp of its last spike, respectively. The variable *last_membrane_potential* records the value of v_m at *last_update_time*. To process event-driven update to membrane potential of LIF neurons, consider that in the absence of input signal, Equation 2.1 is an ordinary differential equation with solution:

$$v(t) = -e^{(-t+C)/\tau_m} + a \tag{5.1}$$

Here, C is the integration constant. With initial membrane potential value of the zero-input period $v(t_{last}) = v_{old}$, Equation 5.1 can be rewritten as:

$$v(t) = (v_{old} - a)e^{-(t-t_{last})/\tau_m} + a \tag{5.2}$$

With *last_update_time* as t_{last}, and value of pre-update *membrane_potential* as v_{old},

membrane potential at time t can be derived using the stored variables. In the actual software implementation, a skip time t_{skip}, which is the longest possible time for a neuron to decay, is used. This is the time for a neuron to decay from spiking threshold to membrane potential reset value without receiving any input, and is calculated at the beginning of simulation with the following equation:

$$t_{skip} = \tau_m ln(v_{threshold} - a) - \tau_m ln(v_{reset} - a) \tag{5.3}$$

As t_{skip} is determined based on the neuron parameters, it is used as a global parameter that does not need to be stored for each individual neuron. During simulation, to update a neuron that receives input spike at time t, $v(t)$ is set to v_{reset} if $t - t_{last} > t_{skip}$, otherwise, Equation 5.2 is evaluated. In other words, if the gap between current time and last update time is larger than t_{skip}, the neuron is set to membrane potential floor without needing extra calculation; otherwise the decayed value is calculated. This process provides the current membrane potential if no input has been received. Now with consideration of input signal I, if current time is larger than *inactive_end_time* meaning that the neuron is not in inactive period, the value of $R_m I$ is added to $v(t)$ to produce the final membrane potential after receiving the input signal. At last, variables of the updated spiking neuron: *last_update_time* and *last_membrane_potential*, are updated. If threshold is crossed, *last_spike_time* is also updated to the current time and the neuron enters refractory period by setting the *inactive_end_time* to a future timestamp. If the input signal is inhibitory, the *inactive_end_time* variable is set to a future timestamp. The detailed process of event-driven update represented in pseudocode form is shown in algorithm 1.

The event-driven neuron simulation as implemented in H-SNN is illustrated in Figure 5.3(a): when a neuron, marked in red, in layer $n-1$ spikes, only the three post-synaptic neurons that are connected to the spiked neuron, marked in yellow, are updated; one updated neuron in layer n crosses threshold and initiates update of its three post-synaptic neurons in layer $n+1$. Other neurons in the network, marked in gray as inactive neurons,

Algorithm 1: Event-driven Neuron Update

```
Global Data: refractory_period, inhibition_period, current_time,
      membrane_resistance, t_skip, v_reset, v_threshold
Neuron Data: last_membrane_potential, last_update_time, inactive_end_time,
      last_spike_time, input_current, inhibition_signal
begin
    if current_time − last_update_time > t_skip then
        last_membrane_potential = v_reset
    else
        Solve (Equation 5.2)
    end
    if inhibition_signal then
        inactive_end_time = max((current_time + inhibition_period),
          inactive_end_time)
    else
        if current_time > inactive_end_time then
            last_membrane_potential += membrane_resistance · input_current
            if last_membrane_potential > v_threshold then
                last_spike_time = current_time
                inactive_end_time = max((current_time + refractory_period),
                  inactive_end_time)
                Initiate STDP
                last_membrane_potential = v_reset
            end
        end
    end
    last_update_time = current_time
end
```

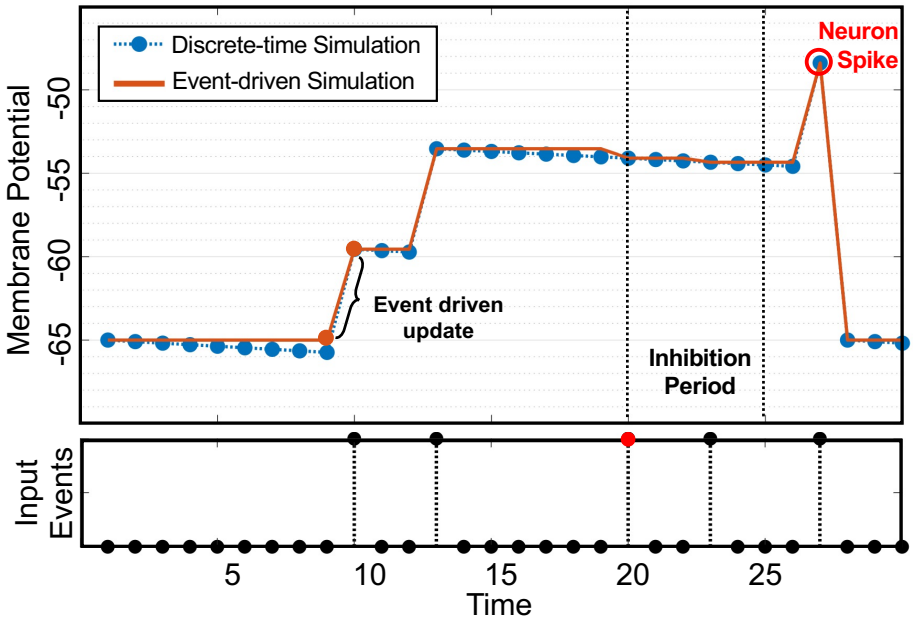


Figure 5.2: Membrane potential tracked over time for discrete-time simulation and event-driven simulation (top) and input events (bottom) with black dots as input spikes and red dot as inhibition signal.

are not updated. Since each neuron needs to receive one or multiple spikes before generating one, when the input is sparse, exponential increase of signal that needs to be processed in a multi-layer convolutional network can be prevented.

To better illustrate the difference between event-driven simulation and regular discrete-time simulation, membrane potential of two identical spiking neurons simulated with the two different methods is shown in Figure 5.2. The two neurons receive the same input event sequence, which contains input spikes and inhibition signal, and the input synapses to the two neurons have the same conductance. It can be observed that for discrete-time simulation, membrane potential of neuron is changed at each timestep. The event-driven neuron is updated only at time of input events, and its membrane potential at input time closely tracks that of the discrete-time simulated neuron, while during other time its membrane potential is unchanged. The two neurons cross threshold at the same time, therefore producing the same spike timing, which is crucial to the STDP learning algorithm.

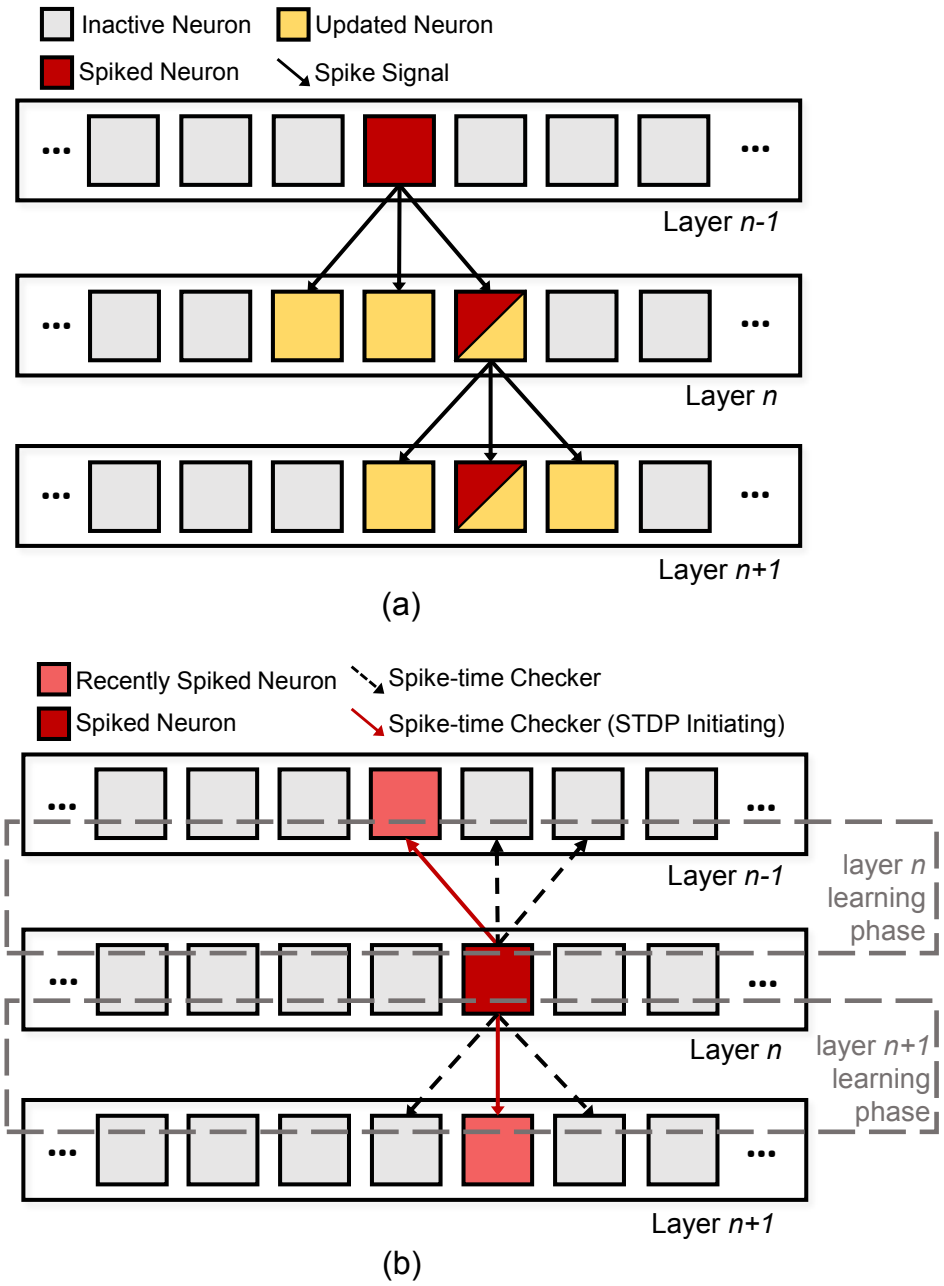


Figure 5.3: In an SNN with each neuron connecting to three neurons in the next layer, (a) shows the event-driven neuron update process, and (b) shows the event-driven STDP process.

5.2.3 Event-driven Learning

Implementing the STDP learning algorithm requires time difference between pre-synaptic and post-synaptic spikes. Discrete-time simulation of SNN can acquire such information with a timer set for spiked neurons. For the proposed event-driven learning, the precise timing of pre-synaptic and post-synaptic spikes is acquired from *last_spike_time* recorded for the pre-synaptic and post-synaptic neurons in their data structure. To support the layer-by-layer STDP learning process used in multi-layer SNN, which will be described later, we implement a fixed-polarity STDP process based on the layer from which spikes are originated.

Specifically, the event-driven learning process proceeds as this: during learning, neurons that generate spikes, referred to as source neurons, initiate a checker function for all pre-synaptic or post-synaptic neurons to the spiked neurons. The checker function computes the difference between the current time and the stored *last_spike_time* of the checked neurons as Δt. Since the network uses convolution connections, the number of neurons to check remains low. To further improve learning efficiency, we exploit the exponential decaying property of STDP magnitude, and apply a time window to filter both the LTP and LTD operations. In case of $\Delta t > T_{window\_LTP}$ for LTP or $\Delta t > T_{window\_LTD}$ for LTD, STDP will not be applied. Based on the spike signal travel direction, a fixed-polarity STDP process is applied. This process regulates that only LTP is considered if the source neuron is post-synaptic, and only LTD is considered if the source neuron is pre-synaptic. In other words, since the network learning process proceeds layer-by-layer, during the learning phase of a layer n, only spikes from neurons in layer n and $n-1$ are considered for STDP. The spiked neuron launches checker function for either its pre-synaptic neurons or post-synaptic neurons depending on the layer it resides, and can initiate only LTP or LTD operation, respectively.

To better explain the event-driven STDP process, an illustration is shown in Figure 5.3(b). In this network, one neuron (the source neuron) in layer n generates a spike at the current

time. If the learning process is in layer n learning phase, checker function accesses the three pre-synaptic neurons of the source neuron in the $n-1$ layer. By checking the values of *last_spike_time*, one neuron in layer $n-1$ is found to have recently spiked. This indicates a causal relationship between the two spiking events since pre-synaptic spike happens before post-synaptic spike. LTP is initiated for the synapse connecting the recently spiked neuron in layer $n-1$ with the source neuron. In this scenario, the post-synaptic neurons in layer $n+1$ are not checked. On the other hand, if learning process is in the layer $n+1$ learning phase, pre-synaptic neurons in layer $n-1$ are not considered. In this case, spike from the source neuron acts as a pre-synaptic spike to neurons in the next layer. Thus, three post-synaptic neurons in layer $n+1$ are checked for their *last_spike_time* and one neuron is found to have spiked recently. Since the spike timing of source neuron is later than the timing of the recently spiked neuron in layer $n+1$, LTD is applied to the synapse to decrease the strength of connection.

5.3 Experimental Details and Results

The event-driven processing method described in the previous section is generic to most feedforward SNN architectures using convolutional layers. In this section, we show an investigation of efficiency of the proposed method for convolutional SNN in general, then review the specific network architectures used in the experiments for unsupervised learning and inference of event camera data. We compare the accuracy of H-SNN with SPEED processing against baseline machine learning and other SNN models. We also demonstrate the computational performance of SPEED processing in comparison to baseline discrete-time event simulation of SNN.

5.3.1 Network Processing Efficiency

We study the efficiency of the proposed method for convolutional SNN sharing similar convolution configurations with networks used in prior works [13, 34], such as the one

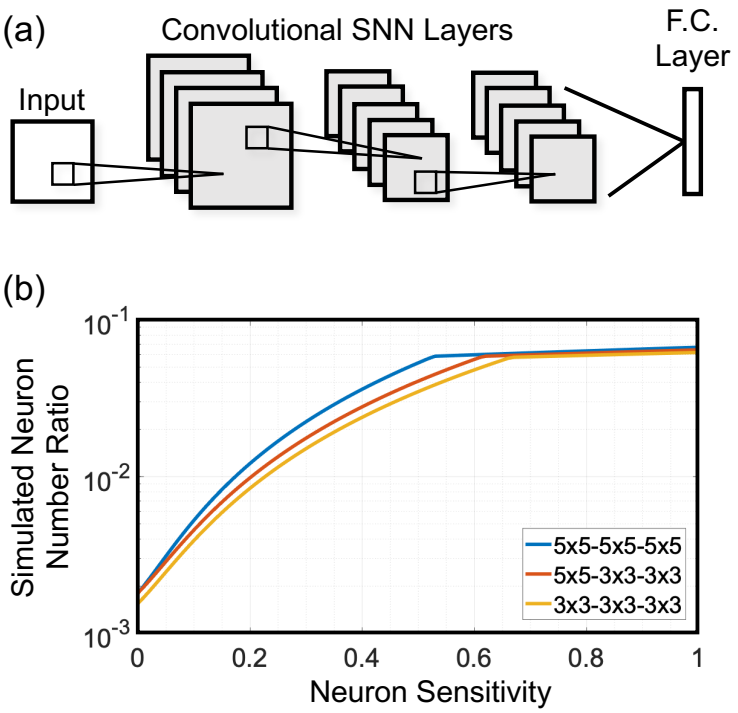


Figure 5.4: (a) A general convolutional SNN architecture; (b) ratio of the number of simulated neurons in event-driven networks to discrete-time simulated networks under different neuron sensitivity.

shown in Figure 5.4, which contains three convolutional layers of spiking neuron and one fully-connected layer for classification. The improvement in efficiency is evaluated by comparing the number of neurons processed per timestep with the regular discrete-time SNN simulation process. The number of processed neuron in an event-driven network depends on two aspects: the convolution configuration which determines the range of spike signal propagation, and the likelihood of spikes being generated from the processed neurons. To help the investigation, we define a metric called neuron sensitivity, which is the ratio between number of spiked neuron and number of total processed neuron, averaged over the simulation period. Neuron sensitivity has a range of 0 to 1, with 0 representing no processed neuron ever spikes and 1 representing all processed neurons always spike. Neuron sensitivity depends on neuron dynamics as well as the distribution of input dataset. When implementing an SNN for STDP learning, network hyperparameters are tuned to achieve the optimal learning behavior, and the resulting neuron sensitivity often settles to

a value well below 1. For example, the networks implemented in our learning tests are measured to have neuron activity on the order of magnitude of 10^{-2}.

In this experiment, efficiency of event-driven networks with different neuron sensitivity levels and three convolution configurations: one with 3 layers of 5x5 convolution kernel, one with 1 layer of 5x5 and 2 layers of 3x3 kernel, and one with 3 layers of 3x3 kernel, are studied. To investigate network activity under specific neuron sensitivity level, the networks consist of pseudo neurons that spike with probability equal to the current neuron sensitivity when receiving input. The test input contains a sequence of events with randomized locations, and the total number of processed neurons in all convolutional SNN layers is recorded. In order to evaluate the reduction of neuron processing, we consider the ratio of the number of processed neurons using the proposed method to the number of processed neurons for a regular, discrete-time simulation process. The result is shown in Figure 5.4(b). It can be observed that, under the rare case of neuron sensitivity near value of 1, for all network configurations, the processed neuron number is still below 10% of discrete-time simulation process. When neuron sensitivity is below 0.5, the event-driven networks achieve significant reduction of neuron update operations. This reduction is also reflected in the actual run-time measurement for learning/inference process shown in subsection 5.3.5.

5.3.2 SPEED Processing of H-SNN

For event camera output which contains both spatial and temporal information, a network with H-SNN architecture is suitable for processing such type of data. To implement H-SNN with SPEED, spiking neurons with different dynamics can be simulated by solving the equations discussed in subsection 5.2.2 using the corresponding neuron parameters. In terms of the event-driven learning and inference process of the entire H-SNN network, we consider one of the networks tested in the experiment, shown in Figure 5.5, as an example. This network contains three heterogeneous convolution layer, and one fully connected

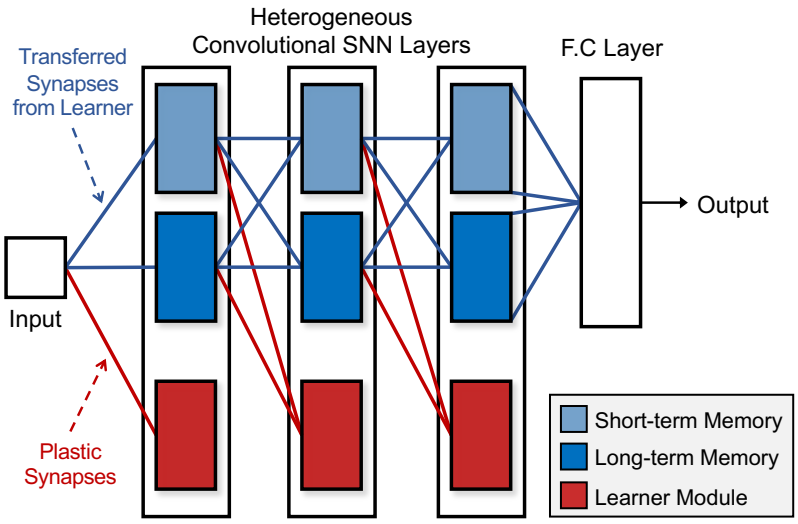


Figure 5.5: An example of H-SNN network implemented using SPEED.

layer. Input signal is sent to the convolution layers each consists of short-term memory, long-term memory and a learner module. In each layer, long-term memory and short-term memory has the same dimension and convolution settings as the learner module. Short-term and long-term neurons have different τ_m and a to achieve different membrane potential decay rate and different R_m for balanced input signal response.

Synapses connecting input and the first layer learner module, as well as those connecting memory module and learner module after the first layer, as marked in red in Figure 5.5, are learned with STDP. Synapses marked in blue have conductance transferred from learner module and are kept fixed. Learner module uses spiking neurons optimized for STDP to learn the input spatiotemporal dataset without supervision. The two memory modules are optimized for perception of patterns with different temporal duration. The last layer is fully connected to the memory modules in its previous convolution layer and provides classification results.

The learned and transferred synapses that are established between long-term memory, short-term memory and the learner module in each layer follow the same connection principle of regular convolutional layers, which can be processed by SPEED using the procedure as presented in subsection 5.2.2. The same applies to skip-layer connections. During STDP

Table 5.2: Comparison of model architectures and training rules

Network	Architecture	Training Rule
3D CNN	4 layers 3D CNN {[3x3x3,20],[3x3x3,32],[5x5x5,64],[5x5x5,64]}	SGD
AsyNet [30]	Sparse deep network (VGG13)	SGD
HATS [68]	Time surface histogram	SVM
SCRNN [75]	SNN with recurrent Conv2D {[5x5,32],[3x3,64],[3x3,128]}	BPTT
DECOLLE [16]	3 layers SNN Conv2D {[7x7,64],[7x7,64],[7x7,128]}	Surrogate Gradient
H-SNN	SNN with Convolutional layers and multiple neuron dynamics	STDP

learning of H-SNN, SPEED first processes the learner module in layer 1, which learns the entire training dataset with event-driven STDP. After the first layer learning is complete, synaptic conductance between input and layer 1 learner module is transferred to layer 1 memory modules in which the synaptic conductance is then kept fixed. This concludes layer 1 learning phase. During layer 2 learning, SPEED processes neurons in layer 1 and layer 2. Layer 1 short-term and long-term memory modules generate output spikes by perceiving the input, and event-driven STDP is used for layer 2 learner module to learn the entire training dataset. This layer-by-layer learning process is repeated for all convolutional layers. Finally the classifier layer is fine-tuned with stochastic gradient descent (SGD). The spike frequency matrix of the last layer memory module, sampled from a sliding windows over the observed event sequence with 50 ms width and 1 ms stride, is used as the training input and the original training sequence label is used as the training target.

5.3.3 Experimental Configuration of Learning Event-based Datasets

In the network learning experiments, the H-SNN network models as discussed in chapter 3 and in chapter 4 processed with discrete-time simulation and the proposed method are compared with different baselines as shown in Table 5.2. The configuration of {[5x5,32],[5x5,64], [5x5,128]} represents a network with one convolutional layer with 5x5 filter and 32 depth

Table 5.3: Comparison of H-SNN models tested in the experiments

Network	Number of Conv. Layer	Skip-layer Connection	Number of Neuron Dynamics
H-SNN Original	4	N/A	2
H-SNN Optimized	6	(3,5),(5,7)	4

followed by a layer of 5x5 filter and 64 depth followed by a layer of 5x5 filter and 128 depth. The tested H-SNN models have the same configurations as presented in prior chapters, and are as shown in Table 5.3.

5.3.4 Accuracy Results

The two datasets used in this test are N-Cars [68] and DVS Gesture [63]. N-Cars is a car recognition benchmark captured with an event camera mounted on a car driving over multiple sessions. Our experiment uses the 15422 training samples and 8607 test samples and each sample lasts 100 ms. DVS Gesture contains human gestures recorded with an event camera in different illumination conditions. There are 1342 recordings with various length and in our experiment 1000 ms is taken for each training sample, and the spatial resolution is 128x128. For both datasets, the two polarities of the events are separated to two input channels. All models use the full training dataset. The computation cost during inference for each model, in terms of average floating point operations required to classify a test sequence, is also compared. The results are shown in Table 5.4 and Table 5.5.

For the N-Cars dataset, baseline networks include a conventional 4-layer 3D CNN trained with stochastic gradient descent (SGD) using input frames converted from the event data, AsyNet [30] that uses sparse VGG13 trained with SGD, and HATS [68] that uses SVM to classify histograms of time surfaces. Results show that AsyNet achieves the highest accuracy among baselines, but at the same time uses the most operations. The event based operation and simple classifier design from HATS provides high efficiency and good accuracy. H-SNN Original reaches accuracy similar to HATS, while using much more op-

Table 5.4: N-Cars dataset: accuracy result and number of operations for inference

Model	Accuracy Result (%)	Number of Operations (FLOP)
3D CNN	87.7	17.3G
AsyNet [30]	94.4	87.0G
HATS [68]	90.2	115M
H-SNN Original	91.4	12.8G
H-SNN Optimized	93.4	24.7G
H-SNN Original (SPEED)	90.3	92.8M
H-SNN Optimized (SPEED)	92.7	179.0M

erations. The SPEED implementation of H-SNN Original achieves similar accuracy as the discrete-time simulated H-SNN Original. It also has accuracy comparable to HATS and 3D CNN, while the computation cost is lower than all baselines. The optimized version of H-SNN reaches performance similar to the highest among baselines. This network requires more operations than the original H-SNN during inference, but the amount is still less than that required by AsyNet. With SPEED implementation of H-SNN Optimized, the number of operations can be significantly reduced while the accuracy remains on similar level.

Table 5.5: DVS Gesture dataset: accuracy result and number of operations for inference

Model	Accuracy Result (%)	Number of Operations (FLOP)
3D CNN	91.4	119.0G
HATS [68]	95.2	13.6G
SCRNN [75]	96.6	278.3G
DECOLLE [16]	97.5	35.6G
H-SNN Original	96.2	82.7G
H-SNN Optimized	96.6	147.1G
H-SNN Original (SPEED)	94.7	2.1G
H-SNN Optimized (SPEED)	96.0	3.83G

For the DVS Gesture dataset, the baseline networks are: 3D CNN using SGD, HATS, SCRNN [75] that uses backpropagation through time (BPTT) to train a recurrent SNN, DECOLLE [16] which implements surrogate gradient learning. Among baseline networks,

DECOLLE achieves the highest accuracy while using a moderate number of operations. 3D CNN shows to be an inefficient design as it has the lowest accuracy and uses a significant amount of operations. Similar to the N-Cars dataset, HATS achieves good accuracy with low number of operations. The original H-SNN and H-SNN Optimized perform well for this dataset, but their discrete-time based simulation process uses a large amount of operations. With the proposed SPEED method, H-SNN Original (SPEED) shows advantage in its low operation count per classification while achieving similar accuracy levels as H-SNN Original. The Bayesian optimized network of H-SNN Optimized (SPEED) achieves performance similar to DECOLLE while using noticeably less number of operations.

5.3.5 Computational Performance

In the previous subsection we show that H-SNN processed with SPEED have comparable or better accuracy than baseline models, while being unsupervised and using less computations. In this subsection, we investigate in details the computation performance of SPEED, by comparing the memory consumption and processing speed of discrete-time simulation and SPEED, considering a simple H-SNN model with 3 convolutional layers that achieves 89.3% accuracy for N-Cars, and 94.3% accuracy for DVS Gesture.

Memory Requirement We derive memory requirement of all neuron and synapse variables based on number of neurons and synapses in the network and unit memory requirement of the two components. The result is 3.61 MB for discrete-time simulation and 4.48 MB for the proposed SPEED method. The overhead from using additional variables to support event-driven network operations, as described in subsection 5.2.2, causes around 24.3% more memory consumption. Overall, the memory overhead from using the proposed method is noticeable but not drastic.

Processing Speed For processing speed comparison, networks are implemented with three different hardware: single threaded processing conducted on Intel Core i5 low power CPU and AMD Ryzen high performance CPU, as well as parallel processing on NVIDIA

Table 5.6: Specifications of hardware used for simulation

	TDP	Core Clock	Memory Clock
Intel Core i5-4278U	28 Watt	3.1 GHz	1600 MHz
AMD Ryzen 5 5600X	65 Watt	4.6 GHz	3800 MHz
NVIDIA RTX 2080 Ti	260 Watt	1.5 GHz	1750 MHz

RTX 2080 Ti GPU. The hardware specifications are listed in Table 5.6 and measurements are shown in Table 5.7. In terms of single threaded processing, we observe over 100 times decrease of latency in H-SNN memory module from using the event-driven operations. A significant reduction for learner module is also observed. The resulting network throughput is around 6.0x of discrete-time simulation for learning, and 167x for inference on high performance CPU; on low power CPU, the improvement is 6.3x for learning and 170x for inference. In terms of parallel processing on GPU, we observe a lower but still considerable improvement in latency for both memory and learning modules. The throughput from using the event-driven operations is around 3.5x for learning and 8.2x for inference, compared to discrete-time simulation.

In general, using parallel processing that is event-driven is the optimal solution for fast learning and inference. It is also worth noting that by using the proposed method, inference speed on both low power and high performance CPU exceeds that on a GPU running the regular discrete-time simulation. In terms of real-time applications of the processing pipeline, maximum readout rate of event cameras can range from 1 MHz to 1200 MHz [23], but the actual event output rate heavily depends on the observed scenes. For example, the N-Cars dataset records an average of 40k events per second. For such scenarios, with discrete-time simulation, SNN can only processes around 10% of the generated events on GPU implementation. While the proposed method still cannot achieve exact real-time learning, it is able to achieve near-real-time inference with GPU. In comparison with conventional CNN, the 3D CNN baseline implemented in PyTorch processes 4.7k events per second when performing inference on the same event stream (batch size of 1) with

Table 5.7: Processing speed measurements

Single Threaded (4278U)	Discrete-time Simulation	Event-driven Simulation
Memory Module Latency	10.1 ms/event	59.2 μs/event
Learner Module Latency	15.9 ms/event	5.4 ms/event
Learning Throughput	26 events/s	164 events/s
Inference Throughput	32 events/s	5424 events/s
Single Threaded (5600X)		
Memory Module Latency	5.9 ms/event	35.0 μs/event
Learner Module Latency	9.1 ms/event	3.2 ms/event
Learning Throughput	46 events/s	274 events/s
Inference Throughput	55 events/s	9216 events/s
GPU Parallel (2080 Ti)		
Memory Module Latency	75.9 μs/event	9.1 μs/event
Learner Module Latency	147.4 μs/event	68.0 μs/event
Learning Throughput	3224 events/s	11351 events/s
Inference Throughput	4325 events/s	35320 events/s

GPU. This speed is similar to SNN with discrete-time simulation but slower than SNN with event-driven simulation.

5.4 Low-precision Networks

Since reduced memory consumption is an attractive property for ASIC or FPGA based SNN learning and inference implementations, low-precision SNN models [84, 85] have been actively studied in prior works, such as the recent development of binary SNN [86, 87]. Here, we investigate the influence of low-precision synapse conductance on the proposed event-driven SNN simulation method. As shown in Table 5.8, for network model with the same architecture as H-SNN Original, reduced precision networks provide considerable reduction in memory requirement. Due to the overall reduced memory consumption, the overhead of event-driven simulation for 4-bit and 2-bit precision increases from 30.5% to 31.9%. Under reduced precision, for STDP algorithm to learn patterns correctly, post-

Table 5.8: Memory requirement of network states with different synapse precision

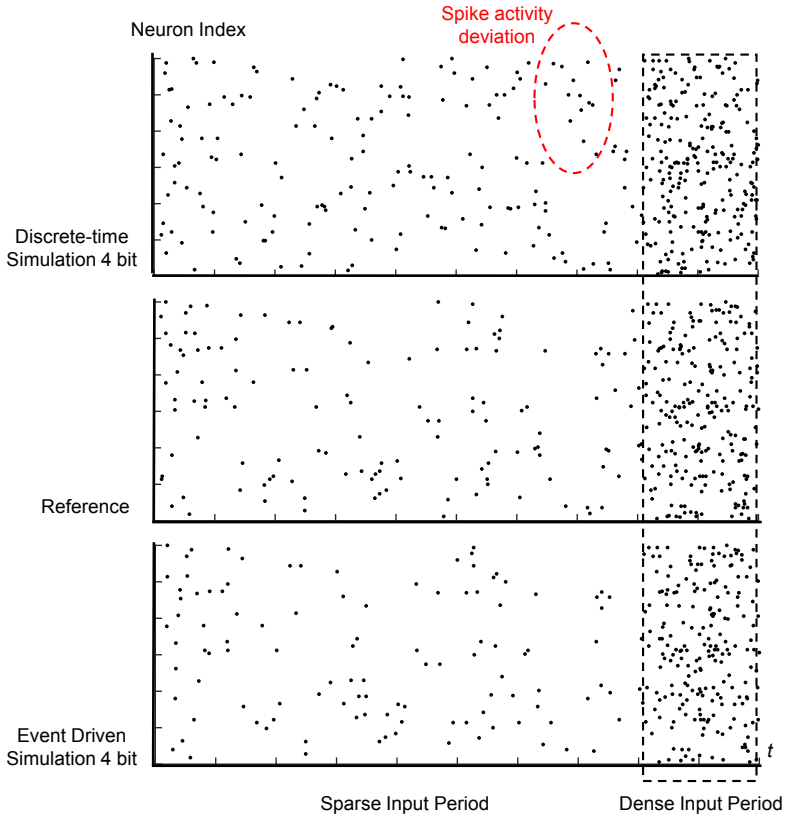
	Floating Point	4-bit	2-bit
Discrete-time Simulation	4.59 MB	2.90 MB	2.78 MB
Event-driven Simulation	5.47 MB	3.78 MB	3.66 MB
Memory Overhead	19.3%	30.5%	31.9%

synaptic activity needs to respond to the input spikes accurately: errors in post-synaptic timing leads to shifted STDP magnitude or even inverted polarity. Therefore, in order to understand the effect of reduced precision on network activities in the proposed event-driven SNN, we conduct two sets of experiments. The first one investigates low-precision network spiking activity, and the second tests low-precision network learning.

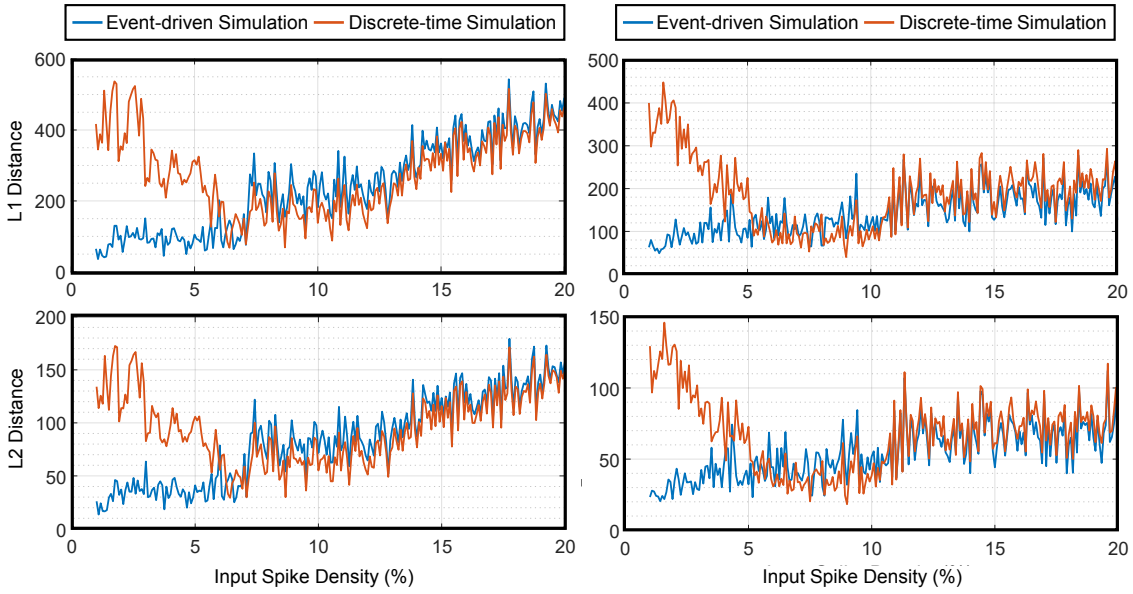
5.4.1 Low Precision Network Activity

First, visual inspection of network activities is performed by plotting spiking events of three networks. The first network, which is the reference design, has floating point precision (no quantization to the conductance matrix) and uses the regular discrete-time simulation; the second network has conductance matrix quantized to 4-bit precision and uses the regular discrete-time simulation; the third network also has conductance matrix quantized to 4-bit precision but uses the proposed event-driven simulation. Three networks receive the same input pattern that starts with sparse signal then switches to dense signal. As shown in Figure 5.6(a), during sparse input, event-driven simulation produces spike activities more similar to the reference design than discrete-time simulation. For discrete-time simulation, as error accumulates, the deviation from reference design is more evident particularly towards the end of sparse input phase, such as the marked area in the figure. On the other hand, when the input becomes dense, the two networks shows similar level of activity distortion from the reference design.

To quantitatively analyze the activity distortion observed in the visual inspection, we again use a network with floating point precision and discrete-time simulation as refer-



(a)



(b)

Figure 5.6: (a) Spike activities from three simulations; each point represents a spike from a specific neuron and time. (b) L_1 and L_2 distance between spike frequency arrays for 2-bit simulations (left) and 4-bit simulations (right).

ence, in comparison with two networks that have reduced-precision synapses simulated with discrete-time and event-driven simulation, respectively. The three networks have the same dimension, and are connected to input layer with the same conductance matrix. We test all three networks with input spikes with different density, and record the generated spike frequency array from all neurons. The distance between frequency array of the reference network and that from networks with reduced precision is measured to quantify the level of spike activity distortion. The distance metric is L_1 and L_2 distance. We conduct this experiment for two bitwidth: 2-bit and 4-bit.

The results are shown in Figure 5.6(b), where distance is plotted against input spike density, which represents the percentage of time the input spike signal is active. It can be observed that, at 2-bit precision as shown on the left, discrete-time simulation has increasing distance below 6% density, while distance of event-driven simulation remains low within this low input density range. Above 6%, the two simulation methods show similar distance, while both increasing with higher input density. At 4-bit precision, a similar trend can be observed for low density input: the event-driven method provides spike frequency array more similar to the reference. It is also worth noting that, at this precision, distance of the two methods do not increase significantly when density is above 10%. This result indicates that, for low-precision networks, event-driven simulation has less activity distortion than regular discrete-time simulation when the input signal is sparse, which is in alignment with the property of event-camera data.

5.4.2 Low Precision Learning

To investigate if the reduction of spike activity distortion improves network learning capability, in this experiment, we test whether event-driven SNN simulation achieves better accuracy than discrete-time simulation under low precision. For reduced-precision synapses, gap between conductance levels is increased and the magnitude of conductance change calculated from the original STDP equations (Equation 2.3 and Equation 2.4) is not com-

Table 5.9: Low-precision learning accuracy (DVS Gesture)

Model	Floating Point	4-bit	2-bit
H-SNN Original	96.2%	87.9%	80.8%
H-SNN Optimized	96.6%	90.1%	83.5%
H-SNN Original (SPEED)	94.7%	90.8%	86.7%
H-SNN Optimized (SPEED)	96.0%	91.9%	88.1%

patible. To implement STDP learning for reduced-precision networks, the following rule is used: the simulator determines STDP polarity and calculates spike time difference with the same procedure as shown in subsection 5.2.3; if the time difference is within STDP window manually tuned to $T_{window\_LTP} = 40$ or $T_{window\_LTD} = 60$, the synaptic conductance is increased/decreased by one step of the quantized conductance levels. Two network models are compared in this experiment: H-SNN Original and H-SNN Optimized, and the two networks are simulated with two methods: regular discrete-time simulation and the proposed event-driven simulation process of SPEED. 2-bit and 4-bit precision levels are compared to floating point in this experiment and the test dataset is DVS Gesture. Results are shown in Table 5.9.

The resulting accuracy values indicate that, for both network models simualted with discrete-time simulation, reducing precision from floating point to 4-bit or 2-bit has noticeable impact to network performance. Meanwhile, with event-driven simulation, the networks experience less performance degradation from reducing the precision, and are able to achieve better accuracy than discrete-time simulated networks. Such accuracy advantage can be observed at 4-bit precision and becomes more obvious at 2-bit precision: compared to discrete-time simulation, H-SNN Original (SPEED) receives around 6% accuracy improvement and H-SNN Optimized (SPEED) has around 4% accuracy improvement.

5.5 Summary

In this chapter, we demonstrate the effort to improve computation efficiency of H-SNN using event-driven processing method. We show that H-SNN can be implemented with the SPEED method to process event-based spatiotemporal data from neuromorphic vision sensors, and demonstrate it to be an efficient solution for both STDP learning and inference processes. The event-driven SNN simulation process used in SPEED introduces non-significant overhead to network memory usage and improves processing speed significantly by reducing the number of operations. Experimental results show that the lower computation cost can be achieved while maintaining similar accuracy of the H-SNN models. The reduced number of operations also introduce less spike-activity distortion to networks with low-precision synaptic conductance, leading to better learning performance than discrete-time simulated low-precision networks. SPEED therefore helps to achieve a fully event-based vision processing pipeline that can be integrated in resource-limited platforms such robot vision, IoT edge monitoring/computing devices, etc.

CHAPTER 6
CONCLUSION

This thesis presents a methodology to implement and optimize feedforward-only SNN for spatiotemporal data processing that is able to achieve performance similar to DNN with high computation efficiency. The proposed network architecture of H-SNN combines spiking neurons with heterogeneous dynamics within each network layer. With the design of learned synapses and transferred synapses, the heterogeneous neurons are connected in a way such that distinct memory pathways can be formed. The collection of distinct memory pathways enables the network to approximate mapping functions from input to output spike sequences. We demonstrate H-SNN designs that can be trained with the biologically inspired STDP unsupervised learning rule or the supervised BPTT training algorithm based on stochastic gradient descent. The H-SNN architecture is demonstrated for both frame-based and event-based data classification tasks, and an event-driven processing method for event-based data is developed to improve the computation efficiency of the proposed design.

In chapter 3, the architecture of H-SNN is presented. Neurons with heterogeneous dynamics and cross-over connections between long-term and short-term neurons are used to create distinct spike propagation paths, i.e. memory pathways, within the feedforward network. The design of separating the learner module and the memory module in each layer facilities STDP learning for the proposed architecture. With empirical studies, we demonstrate H-SNN's capability to learn from spatiotemporal datasets. We show that the proposed design with unsupervised STDP learning can achieve performance on similar level as DNN models while using less training labels.

In chapter 4, the theoretical basis is developed to provide support for the function approximation capability of feedforward SNN. Based on this, we prove that the use of multi-

ple neuron dynamics and skip-layer connections increases the upper bound of the number of distinct memory pathways in H-SNN, thus improves the network's capability to approximate input-output spike sequence mapping functions. An efficient Bayesian-based algorithmic approach to optimize network structure and neuron parameters is developed and used to optimize networks for different spatiotemporal learning tasks. Ablation studies demonstrate the effectiveness of implementing the proposed optimization process for H-SNN. Comparison with baseline networks shows that the proposed design can achieve higher performance than the state-of-the-art SNN baselines, while requiring less trainable parameters.

In chapter 5, an event-driven processing pipeline of event-camera data, named SPEED, is proposed for H-SNN. The event-driven STDP learning and inference methods significantly reduce computation cost and increase network throughput compared to regular discrete-time simulation of SNN. It is also demonstrated that the SPEED method has improved robustness for reduced-precision networks when the input is sparse. For hardware implementations of event-driven SNN, this network learning/inference method facilitates potential designs that have reduced requirement for computation resources, while maintaining similar level of learning performance.

In summary, theoretical and empirical research is performed on using neurons with heterogeneous dynamics in feedforward SNN for spatiotemporal processing. Based on this, the architecture of H-SNN is proposed and optimized. We show that H-SNN demonstrates learning performance comparable to DNN while being more label efficient. With the proposed processing method of SPEED implemented, the computation cost of using H-SNN for event-based data processing can be significantly reduced.

Future Works The completed work was primarily focused on computer vision related classification tasks and future works include designing H-SNN architecture for other computer vision tasks such as object tracking and segmentation, and for other applications

such as natural language processing. Beside expanding the application domains, another potential direction to explore both theoretically and empirically, is to investigate the generalization capability of H-SNN in comparison with DNN and other SNN models. The event-based pipeline of SPEED has been designed for STDP learning and future work in this aspect can include developing SPEED for BPTT training of H-SNN, which has shown promising results in the spatiotemporal learning experiments. Another potential future development is to further improve SPEED and the design of H-SNN aiming specifically for low-precision simulations.

REFERENCES

[1] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[2] W. Gerstner and W. M. Kistler, "Mathematical formulations of hebbian learning," *Biological cybernetics*, vol. 87, no. 5-6, pp. 404–415, 2002.

[3] C. C. Bell, V. Z. Han, Y. Sugawara, and K. Grant, "Synaptic plasticity in a cerebellum-like structure depends on temporal order," *Nature*, 1997.

[4] J. C. Magee and D. Johnston, "A synaptically controlled, associative signal for Hebbian plasticity in hippocampal neurons," *Science*, 1997.

[5] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[6] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience*, vol. 11, p. 682, 2017.

[7] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in neuroscience*, vol. 13, p. 95, 2019.

[8] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, ieee, 2015, pp. 1–8.

[9] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, vol. 10, p. 508, 2016.

[10] D. Huh and T. J. Sejnowski, "Gradient descent for spiking neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 1433–1443.

[11] W. Nicola and C. Clopath, "Supervised learning in spiking neural networks with force training," *Nature communications*, vol. 8, no. 1, pp. 1–15, 2017.

[12] G. Srinivasan and K. Roy, "Restocnet: Residual stochastic binary convolutional spiking neural network for memory-efficient neuromorphic computing," *Frontiers in Neuroscience*, vol. 13, p. 189, 2019.

[13] C. Lee, P. Panda, G. Srinivasan, and K. Roy, "Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning," *Frontiers in neuroscience*, vol. 12, p. 435, 2018.

[14] B. DePasquale, M. M. Churchland, and L. Abbott, "Using firing-rate dynamics to train recurrent networks of spiking model neurons," *arXiv preprint arXiv:1601.07620*, 2016.

[15] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems*, 2018, pp. 787–797.

[16] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *Frontiers in Neuroscience*, vol. 14, p. 424, 2020.

[17] I. Marković, F. Chaumette, and I. Petrović, "Moving object detection, tracking and following using an omnidirectional camera on a mobile robot," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 5630–5635.

[18] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 6753–6760.

[19] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.

[20] J. Donahue *et al.*, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2625–2634.

[21] M. Weiler, F. A. Hamprecht, and M. Storath, "Learning steerable filters for rotation equivariant cnns," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 849–858.

[22] G. Cheng, P. Zhou, and J. Han, "Learning rotation-invariant convolutional neural networks for object detection in vhr optical remote sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 12, pp. 7405–7415, 2016.

[23] G. Gallego *et al.*, "Event-based vision: A survey," *arXiv preprint arXiv:1904.08405*, 2019.

[24] S. Liu, B. Rueckauer, E. Ceolini, A. Huber, and T. Delbruck, "Event-driven sensing for efficient perception: Vision and audition algorithms," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 29–37, 2019.

[25] S. Afshar, A. P. Nicholson, A. van Schaik, and G. Cohen, "Event-based object detection and tracking for space situational awareness," *IEEE Sensors Journal*, vol. 20, no. 24, pp. 15 117–15 132, 2020.

[26] G. Chen, L. Hong, J. Dong, P. Liu, J. Conradt, and A. Knoll, "Eddd: Event-based drowsiness driving detection through facial motion analysis with neuromorphic vision sensor," *IEEE Sensors Journal*, vol. 20, no. 11, pp. 6170–6181, 2020.

[27] D. Gehrig, A. Loquercio, K. G. Derpanis, and D. Scaramuzza, "End-to-end learning of representations for asynchronous event-based data," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5633–5643.

[28] B. Ramesh, H. Yang, G. Orchard, N. A. Le Thi, S. Zhang, and C. Xiang, "Dart: Distribution aware retinal transform for event-based cameras," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 11, pp. 2767–2780, 2019.

[29] M. Cannici, M. Ciccone, A. Romanoni, and M. Matteucci, "Asynchronous convolutional networks for object detection in neuromorphic cameras," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.

[30] N. Messikommer, D. Gehrig, A. Loquercio, and D. Scaramuzza, "Event-based asynchronous sparse convolutional networks," in *European Conference on Computer Vision*, Springer, 2020, pp. 415–431.

[31] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, pp. 500–544, 1952.

[32] R. Moreno-Bote and J. Drugowitsch, "Causal Inference and Explaining Away in a Spiking Network," *Scientific Reports*, 2015.

[33] B. J. Lansdell and K. P. Kording, "Spiking allows neurons to estimate their causal effect," *bioRxiv*, p. 253 351, 2019.

[34] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "Stdp-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, 2018.

[35] 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.

[36] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 288–295, 2013.

[37] X. She, Y. Long, and S. Mukhopadhyay, "Improving robustness of reram-based spiking neural network accelerator with stochastic spike-timing-dependent-plasticity," in *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2019, pp. 1–8.

[38] ——, "Fast and low-precision learning in gpu-accelerated spiking neural network," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2019, pp. 450–455.

[39] Y. Amit and Y. Huang, "Precise capacity analysis in binary networks with multiple coding level inputs," *Neural Computation*, vol. 22, pp. 660–688, 2010.

[40] J. Brea, W. Senn, and J.-P. Pfister, "Matching recall and storage in sequence learning with spiking neural networks," *Journal of neuroscience*, vol. 33, no. 23, pp. 9565–9575, 2013.

[41] W. Maass, "Fast sigmoidal networks via spiking neurons," *Neural computation*, vol. 9, no. 2, pp. 279–304, 1997.

[42] N. Iannella and A. D. Back, "A spiking neural network architecture for nonlinear function approximation," *Neural Networks*, vol. 14, no. 6, pp. 933–939, 2001.

[43] H. Torikai, A. Funew, and T. Saito, "Digital spiking neuron and its learning for approximation of various spike-trains," *Neural Networks*, vol. 21, no. 2, pp. 140–149, 2008, Advances in Neural Networks Research: IJCNN '07.

[44] E. Z. Farsa, S. Nazari, and M. Gholami, "Function approximation by hardware spiking neural network," *Journal of Computational Electronics*, vol. 14, no. 3, pp. 707–716, 2015.

[45] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, vol. 10, p. 508, 2016.

[46] S. Kim, S. Park, B. Na, and S. Yoon, "Spiking-yolo: Spiking neural network for energy-efficient object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 11 270–11 277.

[47] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, "Direct training for spiking neural networks: Faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1311–1318.

[48] A. Safa *et al.*, "Improving the accuracy of spiking neural networks for radar gesture recognition through preprocessing," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[49] B. Han, G. Srinivasan, and K. Roy, "Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[50] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[52] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[53] R. Zhang, "Making convolutional networks shift-invariant again," in *International Conference on Machine Learning*, 2019, pp. 7324–7334.

[54] A. Azulay and Y. Weiss, "Why do deep convolutional networks generalize so poorly to small image transformations?," 2018.

[55] M. A. Khoei, A. Yousefzadeh, A. Pourtaherian, O. Moreira, and J. Tapson, "Sparnet: Sparse asynchronous neural network execution for energy efficient inference," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, IEEE, 2020, pp. 256–260.

[56] E. De Kloet and J. Reul, "Feedback action and tonic influence of corticosteroids on brain function: A concept arising from the heterogeneity of brain receptor systems," *Psychoneuroendocrinology*, vol. 12, no. 2, pp. 83–105, 1987.

[57] R. C. Gupta, "Brain regional heterogeneity and toxicological mechanisms of organophosphates and carbamates," *Toxicology mechanisms and methods*, vol. 14, no. 3, pp. 103–143, 2004.

[58] Y.-L. Tan, Y. Yuan, and L. Tian, "Microglial regional heterogeneity and its role in the brain," *Molecular psychiatry*, vol. 25, no. 2, pp. 351–367, 2020.

[59] X. She, S. Dash, D. Kim, and S. Mukhopadhyay, "A heterogeneous spiking neural network for unsupervised learning of spatiotemporal patterns," *Frontiers in Neuroscience*, vol. 14, p. 1406, 2021.

[60] O. Köpüklü, N. Kose, A. Gunduz, and G. Rigoll, "Resource efficient 3d convolutional neural networks," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, IEEE, 2019, pp. 1910–1919.

[61] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in neuroscience*, vol. 12, p. 331, 2018.

[62] P. Panda, S. A. Aketi, and K. Roy, "Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization," *Frontiers in Neuroscience*, vol. 14, 2020.

[63] A. Amir *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.

[64] G. Xia *et al.*, "DOTA: A large-scale dataset for object detection in aerial images," *CoRR*, vol. abs/1711.10398, 2017. arXiv: 1711.10398.

[65] J. R. Gardner, M. J. Kusner, Z. E. Xu, K. Q. Weinberger, and J. P. Cunningham, "Bayesian optimization with inequality constraints.," in *ICML*, vol. 2014, 2014, pp. 937–945.

[66] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in Neuroscience*, vol. 9, p. 437, 2015.

[67] D. Salaj, A. Subramoney, C. Kraišniković, G. Bellec, R. Legenstein, and W. Maass, "Spike-frequency adaptation provides a long short-term memory to networks of spiking neurons," *bioRxiv*, 2020. eprint: https://www.biorxiv.org/content/early/2020/05/12/2020.05.11.081513.full.pdf.

[68] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, "Hats: Histograms of averaged time surfaces for robust event-based object classification," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1731–1740.

[69] X. She and S. Mukhopadhyay, "Speed: Spiking neural network with event-driven unsupervised learning and near-real-time inference for event-based vision," *IEEE Sensors Journal*, vol. 21, no. 18, pp. 20 578–20 588, 2021.

[70] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014.

[71] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[72] T. Delbruck and P. Lichtsteiner, "Fast sensory motor control based on event-based hybrid neuromorphic-procedural system," in *2007 IEEE International Symposium on Circuits and Systems*, 2007, pp. 845–848.

[73] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "High speed and high dynamic range video with an event camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019.

[74] G. Chen *et al.*, "A novel visible light positioning system with event-based neuromorphic vision sensor," *IEEE Sensors Journal*, vol. 20, no. 17, pp. 10 211–10 219, 2020.

[75] Y. Xing, G. Di Caterina, and J. Soraghan, "A new spiking convolutional recurrent neural network (scrnn) with applications to event-based hand gesture recognition," *Frontiers in Neuroscience*, vol. 14, p. 1143, 2020.

[76] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[77] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. Veidenbaum, "Efficient simulation of large-scale spiking neural networks using cuda graphics processors," in *2009 International Joint Conference on Neural Networks*, 2009, pp. 2145–2152.

[78] W. W. Lytton and M. L. Hines, "Independent variable time-step integration of individual neurons for network simulations," *Neural computation*, vol. 17, no. 4, pp. 903–921, 2005.

[79] F. Naveros, J. A. Garrido, R. R. Carrillo, E. Ros, and N. R. Luque, "Event- and time-driven techniques using parallel cpu-gpu co-processing for spiking neural networks," *Frontiers in Neuroinformatics*, vol. 11, p. 7, 2017.

[80] D. Pecevski, D. Kappel, and Z. Jonke, "Nevesim: Event-driven neural simulation framework with a python interface," *Frontiers in neuroinformatics*, vol. 8, p. 70, 2014.

[81] R. Massa, A. Marchisio, M. Martina, and M. Shafique, "An efficient spiking neural network for recognizing gestures with a dvs camera on the loihi neuromorphic processor," *arXiv preprint arXiv:2006.09985*, 2020.

[82] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," *arXiv preprint arXiv:1810.08646*, 2018.

[83] F. Paredes-Vallés, K. Y. W. Scheper, and G. C. H. E. De Croon, "Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion perception," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[84] E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber, and S.-C. Liu, "Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms," *Frontiers in neuroscience*, vol. 9, p. 222, 2015.

[85] S. R. Kulkarni and B. Rajendran, "Spiking neural networks for handwritten digit recognition—supervised learning and network optimization," *Neural Networks*, vol. 103, pp. 118–127, 2018.

[86] H. Jang, N. Skatchkovsky, and O. Simeone, *Bisnn: Training spiking neural networks with binary weights via bayesian learning*, 2020. arXiv: 2012.08300 [cs.LG].

[87] S. Lu and A. Sengupta, "Exploring the connection between binary and spiking neural networks," *Frontiers in Neuroscience*, vol. 14, p. 535, 2020.