TRAINING DEEP SPIKING NEURAL NETWORK: ENABLING SPIKE-BASED LEARNING

by

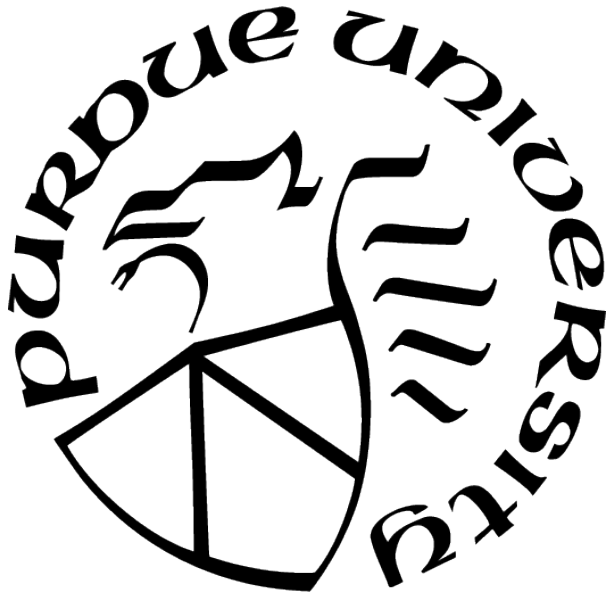
Chankyu Lee

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy



School of Electrical and Computer Engineering

West Lafayette, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Kaushik Roy, Chair

School of Electrical and Computer Engineering

Dr. Anand Raghunathan

School of Electrical and Computer Engineering

Dr. Byunghoo Jung

School of Electrical and Computer Engineering

Dr. Shreyas Sen

School of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis

Dedicated to my parents and brother. It would not have been possible without you.

ACKNOWLEDGMENTS

First of all, I would like to thank my Ph.D. advisor Prof. Kaushik Roy for his amazing guidance, patience and encouragement throughout my doctoral study at Purdue University. Over the past years, I have been extremely fortunate to embark on my doctoral study under him and go through the challenging journey. He has always provided a good role model for me — positive-minded, hard-working and full of intellectual curiosity. Accordingly, I was able to learn from him every day and progressively grow as a capable researcher in my field. Also, I would like to extend my gratitude toward the members of my thesis committee: Prof. Anand Raghunathan, Prof. Byunghoo Jung, and Prof. Shreyas Sen for providing invaluable advice and insights throughout my doctoral study.

Next, I would like to thank my Nokia Bell Labs internship supervisors Dr. Hungkei Chow and Mr. Joe Glaro for giving me an exciting internship opportunity and providing invaluable feedback.

My doctoral study would not have been possible without the support and help of the Nanoelectronics Research Laboratory (NRL) members at Purdue University. I have my deepest thanks to every members who have directly or indirectly interacted with me during my time at Purdue. Particularly, I would like to thank my NRL collaborators Dr. Gopalakrisnan Srinivasan (MediaTek), Dr. Priyadarshini Panda (Yale University), Dr. Syed Shakib Sarwar (Facebook), Siama Sharmin, Amogh Agrawal, Adarsh Kumar Kosta and Sayeed Shafayet Chowdhury for their constructive discussion and critical feedback. I would also like to thank the C-BRIC collaborators Dr. Alex Zihao Zhu (Waymo) and Prof. Kostas Daniilidis (University of Pennsylvania) for bringing the insights and helping me to initiate event-based vision research. In addition, I would like to take this opportunity to thank Dr. Aayush Ankit (Microsoft), Dr. Minsuk Koo (Incheon National University), Dr. Yong Shim (Chung-Ang University) and Dr. Yeongkyo Seo (Inha University) who made my life inside and outside of campus.

My time at Purdue was made enjoyable due to the Purdue Electrical Engineering Korean Association (PEEKA). I would like to thank the current and past members of PEEKA who

welcomed me on the first day at Purdue and supported me when I served as president in the 2017-2018 academic year.

Lastly and most importantly, I would like to express my eternal gratitude to my parents Byungyun Lee and Eunyoung Lee for their endless love and support. Also, I would like to thank my younger brother Chanwook Lee who initially encouraged me to start my doctoral study in the USA and became a companion in pursuing his doctoral degree.

Funding acknowledgements: I would like to acknowledge the funding sponsors of my doctoral research, which include the Center for Brain Inspired Computing (C-BRIC), one of the six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, the Semiconductor Research Corporation, the National Science Foundation, the Sandia National Laboratory, the Intel Corporation, the DoD Vannevar Bush Fellowship, the U.S. Army Research Laboratory and the U.K. Ministry of Defence.

TABLE OF CONTENTS

LIST OF TABLES 13

LIST OF FIGURES 15

ABSTRACT 20

1 INTRODUCTION 22

1.1 Biologically Inspired Computing: Spiking Neural Networks 22

1.2 Contributions 23

1.3 Thesis Outline 24

2 DEEP SPIKING CONVOLUTIONAL NEURAL NETWORK TRAINED WITH UNSUPERVISED SPIKE TIMING DEPENDENT PLASTICITY 26

2.1 Introduction 26

2.2 SNN Fundamental: Leaky-Integrate-and-Fire Neuron Model 28

2.3 Proposed SNN Architecture and Learning Methodology 29

2.3.1 Deep Spiking Convolutional Neural Network (SpiCNN) 29

2.3.2 Synaptic Plasticity 32

Unsupervised Convolutional STDP Learning Methodology 33

Supervised STDP Learning Methodology for Final Layer 34

2.4 Results 35

2.4.1 Simulation Framework 35

2.4.2 MNIST Digit Recognition 37

2.4.3 Caltech Image Recognition 41
2.5 Discussion and Comparison with Related Works 43
2.6 Conclusion 45
3 ENABLING SPIKE-BASED BACKPROPAGATION IN STATE-OF-THE-ART DEEP NEURAL NETWORK ARCHITECTURES 47
3.1 Introduction 47
3.2 Deep Convolutional Spiking Neural Network 48
3.2.1 Building Blocks 48
3.2.2 Deep Convolutional SNN architecture: VGG and Residual SNNs . . 51
3.3 Supervised Training of Deep Spiking Neural Network 52
3.3.1 Spike-based Gradient Descent Backpropagation Algorithm 52
Forward Propagation 52
Backward Propagation and Weight Update 53
3.3.2 Dropout in Spiking Nerual Network 62
3.4 Experimental Setup and Result 63
3.4.1 Experimental Setup 63
Benchmarking Datasets 64
Network Topologies 65
ANN-SNN Conversion Scheme 66
Spike Generation Scheme 67

Time-steps . . . 68
3.4.2 Results . . . 70
The Classification Performance . . . 70
Accuracy Improvement with Network Depth . . . 72
3.5 Discussion . . . 74
3.5.1 Comparison with Relevant Works . . . 74
3.5.2 Spike Activity Analysis . . . 75
Spike Activity per Layer . . . 75
#Spikes/Inference . . . 76
3.5.3 Inference Speedup . . . 79
3.5.4 Complexity Reduction . . . 80
3.5.5 Iso-spike Comparison for Optimal Condition . . . 83
3.6 Conclusion . . . 84
4 TRAINING DEEP SPIKING CONVOLUTIONAL NEURAL NETWORKS WITH STDP-BASED UNSUPERVISED PRE-TRAINING FOLLOWED BY SUPERVISED FINE-TUNING . . . 85
4.1 Introduction . . . 85
4.2 Multi-layer Convolutional Spiking Neural Network Topology . . . 86
4.3 Proposed Semi-Supervised Learning Methodology . . . 87
Unsupervised Pre-training using Spike-Timing-Dependent-Plasticity . . . 88
Supervised Fine-tuning using Spike-based Backpropagation . . . 90

4.4 Results . . . 94
4.5 Discussion . . . 101
4.6 Conclusion . . . 104
5 TOWARDS UNDERSTANDING THE EFFECT OF LEAK IN SPIKING NEURAL NETWORKS . . . 106
5.1 Introduction . . . 106
5.2 Spiking Neural Network Fundamentals . . . 107
5.2.1 Spiking Neuron Model . . . 107
5.2.2 Frequency Domain Analyses . . . 108
5.2.3 Gradient Descent Learning in SNNs . . . 110
5.3 Poisson Spike Generation under Noisy Environments . . . 112
5.4 Experiments . . . 113
5.4.1 Experimental Setup . . . 113
5.4.2 Robustness against Noisy Spike-inputs . . . 115
5.4.3 Spectrum Analysis . . . 116
5.4.4 Analyses of Generalization . . . 117
5.4.5 Input Activity Analysis . . . 118
5.5 Discussion and Conclusion . . . 120
5.6 Appendix: Detailed Formulation of Coherence Function . . . 122
5.6.1 LIF Model Equation . . . 122

5.6.2 Coherence Function 124

6 SPIKE-FLOWNET: EVENT-BASED OPTICAL FLOW ESTIMATION WITH ENERGY-EFFICIENT HYBRID NEURAL NETWORKS 127

6.1 Introduction 127

6.2 Related Work 129

6.3 Method 130

6.3.1 Spiking Input Event Representation 130

6.3.2 Self-Supervised Loss 132

6.3.3 Spike-FlowNet Architecture 133

6.3.4 Backpropagation Training in Spike-FlowNet 136

6.4 Experimental Results 137

6.4.1 Dataset and Training Details 137

6.4.2 Algorithm Evaluation Metric 138

6.4.3 Average End-point Error (AEE) Results 138

6.4.4 Ablation Study 140

Hybrid Network 140

Input Representation 141

Loss Function 141

6.4.5 Computational Efficiency 142

6.5 Conclusion 143

7 FUSION-FLOWNET: ENERGY-EFFICIENT OPTICAL FLOW ESTIMATION USING SENSOR FUSION AND DEEP FUSED SPIKING-ANALOG NETWORK ARCHITECTURES 145

7.1 Introduction 145

7.2 Related Works 146

7.3 Sensors and Input Representation 147

7.3.1 Frame-based Images 147

7.3.2 Stream of Events 148

7.3.3 Sensor-fusion 149

7.4 Neuron Models 149

7.4.1 LeakyReLU Model 150

7.4.2 Signed Integrate-and-Fire (S-IF) Model 150

7.5 Fusion-FlowNet Architecture 151

7.6 Unsupervised Training Method 153

7.6.1 Photometric Loss 153

7.6.2 Smoothness Loss 154

7.7 Backpropagation in Fusion-FlowNet 154

7.8 Experiments 155

7.8.1 Dataset and Training Details 155

7.8.2 Evaluation of Optical Flow 156

7.8.3 Results 157

7.8.4 Ablation studies 158
Architectural Variations 158
Neuron Model Choice 158
Sensor Fusion 159
7.8.5 Computational Efficiency 159
7.9 Conclusion 161
8 SUMMARY AND FUTURE WORK 162
REFERENCES 164
VITA 178

LIST OF TABLES

2.1 SpiCNN simulation parameters. 36

2.2 The learning parameter χ_{offset}. 39

2.3 Classification accuracy of 2C-1S-FC SpiCNN on the MNIST training and testing dataset. 43

2.4 Classification accuracy of SNNs for Caltech (Face/Motorbike). 44

2.5 Classification accuracy of SNNs for MNIST digit recognition. 44

3.1 Parameters used in the experiments 64

3.2 Benchmark Datasets 65

3.3 The deep convolutional spiking neural network architectures for MNIST, N-MNIST and SVHN datasets 66

3.4 The deep convolutional spiking neural network architectures for a CIFAR-10 dataset 66

3.5 Comparison of the SNNs classification accuracies on MNIST, N-MNIST and CIFAR-10 datasets. 71

3.6 Comparison of classification performance 72

3.7 #Spikes/Image inference and spike efficiency comparison between SNN and ANN-SNN converted networks for benchmark datasets trained on different network models. (For each network, the 1st row corresponds to iso-accuracy and the 2nd row corresponds to maximum-accuracy condition.) 77

3.8 Inference #time-steps and corresponding speedup comparison between SNN and ANN-SNN converted networks for benchmark datasets trained on different network models. (For each network, the 1st row corresponds to iso-accuracy and the 2nd row corresponds to maximum-accuracy condition.) 79

3.9 Iso-spike comparison for optimal condition. SNN time-steps corresponds to the latency to reach accuracy within 1% of maximum accuracy. ANN-SNN time-steps corresponds to the latency required for same number of spike/inference as SNN to occur. SNN and ANN-SNN accuracies are accuracies corresponding to respective latency. 82

4.1 Parameters used in the experiments 95

4.2 Learning rate and mean standard deviation of classification errors in shallow and deep multi-layer networks 96

4.3 Comparison of the SNNs classification accuracies on MNIST digit recognition task. 98

4.4 Mean standard deviation of classification errors that are initialized with different weight initialization schemes in deep multi-layer networks 100

4.5 Final testing and training NLL costs (averaged out over 130-150 epochs) in shallow and deep multi-layer networks 102

5.1 Comparison between the network models with different leak amounts. The first row corresponds to baseline accuracy. The second and the third rows correspond to the sum-squared errors averaged over 130–150 epochs for testing and training data, respectively. The fourth and the fifth rows correspond to average spiking activity and the total number of synaptic operations, respectively. 115

6.1 Average Endpoint Error (AEE) comparisons with Zhu et al. [136] and EV-FlowNet [133]. 138

6.2 Analysis for Spike-FlowNet in terms of the mean spike activity, the total and normalized number of SNN operations in an encoder-block, the encoder-block and overall computational energy benefits. 140

6.3 Average Endpoint Error (AEE) for ablation studies with different design choices 141

7.1 AEE_{event} comparison with previous works (lower is better). 156

7.2 Average endpoint error (AEE) results for ablation studies (lower is better) . . . 157

7.3 Comparison of number of parameters and computational energy cost for different architectures for dt=1 and dt=4 cases. 159

LIST OF FIGURES

2.1 The operation of a Leaky Integrate and Fire (LIF) neuron. 28

2.2 Architecture of the proposed deep SpiCNN consisting of the input layer, stacked convolutional layers for feature extraction, spatial-pooling layer for dimensionality reduction, and a final fully-connected layer for inference. . . . 29

2.3 Illustration of the operation of the convolutional, spatial-pooling, and fully-connected layers. The input feature maps are convolved with the weight kernels to produce the respective output feature maps. The convolutional feature maps are then pooled spatially using max-pooling technique to achieve dimensionality reduction with minimal loss of spike information. The neurons in the max-pooled feature maps are unrolled and fully-connected to the output layer, which is augmented with competitive lateral inhibition for improved recognition capability. 30

2.4 (a) Weight-dependent STDP rule used for capturing the temporal correlation between a pair of pre- and post-synaptic spikes. (b) Illustration of the non-linear dependence of synaptic weight update on the current weight given W_{max}=1, W_{min}=−1. 31

2.5 Few samples from the MNIST and Caltech datasets and the respective Laplacian of Gaussian (LoG) filtered binary and grayscale images. 36

2.6 (a) Features self-learned by kernels of different sizes making up a 1C-1S-FC SpiCNN, composed of a single convolutional layer followed by a spatial-pooling layer and a fully-connected layer, trained for MNIST digit recognition. (b) Classification accuracy of 1C-1S-FC SpiCNN against the number of convolutional kernels for different kernel sizes. (c) Classification accuracy of various deep SpiCNN configurations with 16 feature maps per layer for different kernel sizes. 37

2.7 Feature maps at the output of every layer of a 2C-1S-FC SpiCNN for different kernel sizes. 38

2.8 Normalized Spiking activity of feature maps at the output of every layer of a 2C-1S-FC SpiCNN for different kernel sizes for (a) MNIST handwritten digit and (b) Caltech (Face/Motorbike) datasets. 40

2.9 (a) Features self-learned by kernels of different sizes making up a 1C-1S-FC SpiCNN trained for Caltech image recognition. (b) Classification accuracy of 1C-1S-FC SpiCNN against the number of convolutional kernels for different kernel sizes. (c) Classification accuracy of various deep SpiCNN configurations with 16 feature maps per layer for different kernel sizes. 42

3.1 Illustration of the simplified operational example of (a) convolutional, (b) spatial-pooling layers (assuming 2-D input and 2-D weight kernel) over three time steps. At each time step, the input spikes are convolved with the weight kernel to generate the current influx, which is accumulated in the post-neuron's membrane potential, V_{mem}. Whenever the membrane potential exceeds the firing threshold (V_{th}), the post-neuron in the output feature map spikes and V_{mem} resets. Otherwise, V_{mem} is considered as residue in the next time step while leaking in the current time step. For spatial-pooling, the kernel weights are fixed, and there is no membrane potential leak. 49

3.2 The basic building blocks of the described convolutional SNN architectures. (a) Spiking VGG Block. (b) Spiking ResNet Block. 50

3.3 Illustration of the forward and backward propagation phase of the proposed spike-based BP algorithm in a multi-layer SNN comprised of LIF neurons. . 54

3.4 (a,b) The illustration of the spike generation function of (a) IF and (b) LIF neuron models, respectively. The x-axis represents the total summation of input currents over time, and y-axis indicates the membrane potential (black) and output (red). The IF neuron generates a post-spike when the input currents accumulated in membrane potential overcome the firing threshold (because of no leaky effect in the membrane potential). However, LIF neuron needs more input currents to cross the firing threshold (because of leaky effect in the membrane potential). Hence, the effective threshold of LIF neurons is considered to be larger compared to the case of IF neurons. (c) The illustration of the estimation of the ratio (β) between the total membrane potential (V_{mem}^{total}) of LIF and IF neurons. If the LIF and IF neuron received the same amount of total input current, the ratio of the total membrane potential of LIF and IF neuron would be estimated as 1:β where β is greater than 1. . . 56

3.5 Inference performance variation due to (a) #Training-Timesteps and (b) #Inference-Timesteps. $T\#$ in (a) indicates number of time-steps used for training. Fig. (a) shows that inference accuracy starts to saturate as #training-timesteps increase. In Fig. (b), the zoomed version on the inset shows that the SNN trained with the proposed scheme performs very well even with only 30 time-steps while the peak performance occurs around 100 time-steps. . . 68

3.6 Accuracy improvement with network depth for (a) SVHN and (b) CIFAR-10 dataset. In Fig. (a), inference accuracy improves with an increase in network depth. In Fig. (b), the non-residual networks saturate at a certain depth and start to degrade if network depth increases further. However, the residual blocks in deep residual ANNs allow the network to maintain peak classification accuracy (ResNet9 and ResNet11). 73

3.7 Layer-wise spike activity in direct-spike trained SNN and ANN-SNN converted network for CIFAR-10 dataset: (a) VGG9 (b) ResNet9 network. The spike activity is normalized with respect to the input layer spike activity, which is the same for both networks. The spike activity reduces significantly for both SNN and ANN-SNN converted network towards the later layers. We have used scheme 1 for ANN-SNN conversion. 76

3.8 The comparison of 'accuracy vs latency vs #spikes/inference' for ResNet11 architecture. In this figure, the solid lines are representing inference accuracy while the dashed lines are representing #spikes/inference. The slope of #spikes/inference curve of the proposed SNN is larger than ANN-SNN converted networks. However, since proposed SNN requires much less time-steps for inference, the number of spikes required for one image inference is significantly lower compared to ANN-SNN. The required #time-steps and corresponding #spikes/inference are shown using highlighted points connected by arrows. Log scale is used for x-axis for easier viewing of the accuracy changes for lower number of time-steps. 78

3.9 Inference computation complexity comparison between ANN, ANN-SNN conversion and SNN trained with spike-based backpropagation. ANN computational complexity is considered as a baseline for normalization. 81

4.1 Architecture of the multi-layer convolutional spiking neural network consisting of an input layer, alternating convolutional and spatial-pooling layers, and final fully-connected layers for inference. 86

4.2 Illustration of spike forward and backward propagation of a multi-layer SNN consisting of LIF neurons. In forward pass, the spiking neuron integrates the input current (net) generated by the weighted sum of the pre-neuronal spikes with the interconnecting synaptic weights and produces an output spike train. In backward pass, the derivatives of designated loss function with respect to each synaptic weight are calculated from chain-rule. 91

4.3 The classification accuracies (in log scale) on (a) shallow and (b) deep multi-layer convolutional spiking neural networks of pre-trained and supervised model starting from different states of randomly initialized synaptic weights. 96

4.4 The plots show the classification accuracies on (a) shallow and (b) deep multi-layer convolutional spiking neural network as the semi-supervised optimization runs. The x-axis is the number of iterations (in log scale) and y-axis is classification accuracies (in log scale) on testing data. 98

4.5 The weight kernels of (a) purely supervised and (b) pre-trained model in first convolutional layer. 99

4.6 The classification accuracies (in log scale) on the deep multi-layer convolutional spiking neural networks of pre-trained and supervised model starting from (a) Lee initialization (b) Glorot initialization. 100

4.7 The plots show the NLL cost on (a) the shallow and (b) deep multi-layer convolutional spiking neural network. The horizontal and vertical axis indicate the NLL costs (in log scale) on training and testing data, respectively. 103

5.1 An LIF neuron, (a) a schematic connection between three pre-neurons to one post-neuron, (b) temporal dynamics of membrane potential in the post-neuron, (c) equivalent circuit model of the LIF neuron. 107

5.2 Illustration of frequency response for IF and LIF neuron models. The horizontal and vertical axes represent the frequency components and coherence function, respectively. 109

5.3 Classification accuracy at each level of noise severity. The horizontal and vertical axes present the input noise severity and classification accuracy, respectively. (a,b,c,d) Results from noisy input generation scenario 1. (e,f,g,h) Results from noisy input generation scenario 2. 114

5.4 Histogram of the spectrum of spike trains per image for clean and noisy (Gaussian noise) inputs with corresponding distribution curves. 116

5.5 Histogram of average normalized critical frequency components of target output neuron for (a) $\tau_m = infinity$, (b) $\tau_m = 100$ and (c) $\tau_m = 30$. The sum-squared errors of test data with respect to the ones of training data on (d) CIFAR-10 and (e) SVHN benchmarks. The horizontal and vertical axes present the sum-squared error (in log scale) on train and test data, respectively. 118

5.6 Layer-wise Euclidean norm of the weighted sum of spike-inputs of multi-layered SNNs for (a) CIFAR-10 and (b) SVHN datasets. 119

6.1 Input event representation. (*Top*) Continuous raw events between two consecutive grayscale images from an event camera. (*Bottom*) Accumulated event frames between two consecutive grayscale images to form the former and the latter event groups, serving as inputs to the network. 131

6.2 Spike-FlowNet architecture. The four-channeled input images, comprised of ON/OFF polarity events for former and latter groups, are sequentially passed through the hybrid network. The SNN-block contains the encoder layers followed by output accumulators, while the ANN-block contains the residual and decoder layers. The loss is evaluated after forward propagating all consecutive input event frames (a total of N inputs, sequentially taken in time from the former and the latter event groups) within the time window. The black arrows denote the forward path, green arrows represent residual connections, and blue arrows indicate the flow predictions. 134

6.3 Error backpropagation in Spike-FlowNet. After the forward pass, the gradients are back-propagated through the ANN block using standard backpropagation whereas the backpropagated errors pass through the SNN layers using the approximate IF gradient method and BPTT technique. 137

6.4 Optical flow evaluation and comparison with EV-FlowNet. The samples are taken from (*top*) *outdoor_day*1 and (*bottom*) *indoor_day*1. The Masked Spike-FlowNet Flow is basically a sparse optical flow computed at pixels at which events occurred. It is computed by masking the predicted optical flow with the spike image. 139

7.1 Dynamics of Signed Integrate-and-Fire (S-IF) neuron model. Whenever the membrane potential crosses either positive- or negative-threshold, the neuron fires a signed spike and resets its membrane potential. 150

7.2 The detailed illustration of the Fusion-FlowNet$_{\text{Early}}$. The network contains the SNN- and ANN-based encoder-branches to extract features from event streams and grayscale images, respectively. The rest of networks, involving residual and decoder blocks, are composed of ANN layers. The colors represent the types of layers. 152

7.3 The illustrations of activation function and its derivative (*left*) LeakyReLU neuron (*right*) S-IF neuron. 154

7.4 Predicted optical flow compared with Spike-FlowNet and Full-fledged ANN. The samples are taken from (*top*) *outdoor_day*1 and (*bottom*) *indoor_flying*2. Best viewed in color. 157

7.5 The architectures of (*left*) Fusion-FlowNet$_{\text{Early}}$ and (*right*) Fusion-FlowNet$_{\text{Late}}$. 158

ABSTRACT

In this era of data deluge with real-time contents continuously generated by distributed sensors, intelligent neuromorphic systems are required to efficiently deal with the massive amount of data and computations in ubiquitous automobiles and portable edge devices. Spiking Neural Networks (SNNs), often regarded as third generation neural networks, can be highly power-efficient and have competitive capabilities to deal with numerous cognitive tasks. However, the typical shallow spiking network architectures have limited capacity for expressing complex representations while training a very deep spiking network has not been successful so far.

The first part of this thesis explores several pathways to effectively train deep SNNs using unsupervised, supervised and semi-supervised schemes, and neuron model analysis. First, we present a layer-wise unsupervised Spike-Timing-Dependent-Plasticity (STDP) for training deep convolutional SNNs. Second, we propose an approximate derivative method to overcome the discontinuous and non-differentiable nature of spike generation function and to enable training deep convolutional SNNs with input spike events using supervised spike-based backpropagation algorithm. Third, we develop a pre-training scheme using biologically plausible unsupervised learning, namely STDP, in order to better initialize the network parameters prior to supervised spike-based backpropagation. In addition, we present a comprehensive and comparative analysis between neuron models with and without leak to analyze the impacts of leak on noise robustness and spike sparsity in deep SNNs.

The later part of this thesis explores the combination between SNNs and event camera that provides highly temporal information in the form of spike streams. Event-based cameras display great potential for a variety of tasks such as high-speed motion detection and navigation in low-light environments where standard frame-based cameras suffer critically. However, conventional computer vision methods as well as deep Analog Neural Networks (ANNs) are not compatible in their native form with the asynchronous and discrete nature of event camera outputs. In this regard, SNNs serve as ideal paradigms to directly handle event camera outputs. However, deep SNNs suffer in terms of performance due to the spike vanishing phenomenon. To overcome these issues, we present Spike-FlowNet, a deep hybrid

neural network architecture integrating SNNs and ANNs for efficiently estimating optical flow from sparse event camera outputs without sacrificing the performance. Furthermore, we propose Fusion-FlowNet, a sensor/architecture fusion framework for accurately estimating dense optical flow. In essence, we leverage the complementary characteristics of event- and frame-based sensors as well as ANNs and SNNs.

1. INTRODUCTION

1.1 Biologically Inspired Computing: Spiking Neural Networks

Over the last few years, deep learning has made tremendous progress and has become a prevalent tool for coping with various cognitive tasks such as object detection, speech recognition and reasoning. Various deep learning techniques [1]–[3] enable the effective optimization of deep Analog Neural Networks (ANNs[1]) by constructing multiple levels of feature hierarchies and show remarkable results, which occasionally outperform human-level performance [4]–[6]. To that effect, deploying deep learning is becoming necessary not only on large-scale computers, but also on edge devices (*e.g.* phone, tablet, smartwatch, robot, etc.). However, the ever-growing complexity of the state-of-the-art deep neural networks together with the explosion in the amount of data to be processed, place significant energy demands on current computing platforms. For example, typical deep ANN models require the unprecedented amount of computing hardware resources that often necessitate huge computing power of cloud servers and significant amount of time to train.

Biologically inspired neuromorphic computing models are widely being explored in an effort to mimic the computational efficiency of the human brain in performing classification, recognition, and decision making among other tasks. Spiking Neural Networks (SNNs), which is often regarded as the third generation of neural networks [7], have garnered significant research interest because of their ability to closely emulate certain facets of computations performed by the human brain. The high bio-fidelity and intrinsic sparse event-driven processing capability render SNNs an ideal neuromorphic-computing paradigm for realizing energy-efficient hardware [8]–[10] with on-chip intelligence for classification and recognition applications. The recent works [11], [12] have shown that these properties make SNNs significantly more attractive for deeper networks in the case of hardware implementation. This is because the spike signals become significantly sparser as the layer goes deeper, such that the number of required computations significantly reduces.

[1]↑We refer to the conventional deep learning networks as ANNs, owing to their analog nature of inputs and computations. This nomenclature helps to distinguish them from Spiking Neural Networks (SNNs), which perform spike-based computations.

Till now, two-layered (shallow) fully-connected SNN architectures have been widely explored for classification and recognition tasks [13]–[15]. However, they necessitate the large number of trainable parameters to attain competitive classification accuracy, which constrains their scalability for solving complex tasks. In recent times, several developments on multi-layer SNNs, composed of an input layer followed by two or more hidden layers and an output layer, address the scalability issue [16]–[19]. These multi-layer neural networks essentially allow the systems to hierarchically classify the complex input patterns by building feature hierarchies. The early layer detects elementary representations of input patterns while the subsequent layers capture the higher-level concepts comprising elementary features. Nevertheless, the training of deeper SNNs remains an intricate and challenging problem.

1.2 Contributions

In this thesis, we address this scalability challenge, improve the accuracy, energy-efficiency and robustness, and implement SNNs on novel applications. The key contributions of this thesis are summarized as follows,

- We explore novel learning algorithms that allow deep SNNs to achieve competitive accuracy, energy-efficiency and robustness. In regard to this, we present unsupervised, supervised and semi-supervised spike-based learning algorithms to develop the SNN frameworks for image classifications. Further, we show a comprehensive and comparative analysis between neuron models with and without leak to analyze the impacts of leak on noise robustness and spike sparsity in deep SNNs.

- Event cameras, such as Dynamic Vision Sensors (DVS) [20], are new types of bio-inspired vision sensors that asynchronously detect logarithmic brightness changes rather than sampling intensities to form synchronous frame-based images as in traditional cameras. This working principle grants promising advantages, namely high temporal resolution, high dynamic range and low power consumption. However, conventional computer vision methods as well as deep ANNs are no longer compatible in their native form with the asynchronous and discrete nature of event camera outputs. In regard to this, we demonstrate a great promise of SNNs for directly handling event-camera outputs. Furthermore,

we show that SNNs can effectively exploit the inherent sparsity of event streams by performing efficient event-based computations, carrying out operations only at the arrival of the input events.

1.3 Thesis Outline

The initial four chapters of this thesis explore several pathways to effectively train deep SNNs using unsupervised, supervised and semi-supervised schemes, and analyze the different neuron models. In the last two chapters of this thesis, we explore the match between SNNs and event camera that provides highly temporal information in the form of spike streams. More specific overview of chapters are as follows,

In chapter 2, we train convolutional kernels layer-by-layer in an unsupervised manner using Spike Timing Dependent Plasticity (STDP) that enables them to self-learn characteristic features making up the input patterns. In order to further improve the feature learning efficiency, we propose using smaller 3×3 kernels trained using STDP-based synaptic weight updates performed over a mini-batch of input patterns.

In chapter 3, we address the challenges to directly train deep SNNs with supervised backpropagation algorithm using input spike events due to the discontinuous and non-differentiable nature of spike generation function. To overcome these challenges, we propose an approximate derivative to enable training deep convolutional SNNs with input spike events using spike-based backpropagation algorithm. Our experiments show the effectiveness of the proposed spike-based learning strategy on state-of-the-art deep networks (VGG and Residual architectures) by achieving the best classification accuracies in MNIST, SVHN and CIFAR-10 datasets compared to other SNNs trained with spike-based learning at the time of writing this thesis.

In chapter 4, we propose a pre-training scheme using biologically plausible unsupervised learning, namely STDP, in order to better initialize the parameters in multi-layer systems prior to supervised learning. We train the deep SNNs in two phases wherein, first, convolutional kernels are pre-trained in a layer-wise manner with unsupervised learning followed by fine-tuning the synaptic weights with supervised spike-based backpropagation. Our exper-

iments on digit recognition demonstrate that the STDP-based pre-training with gradient-based optimization provides improved robustness, faster ($\sim 2.5\times$) training time and better generalization compared with purely gradient-based training without pre-training.

In chapter 5, we investigate the questions regarding the justification of leak and the pros and cons of using leaky behavior. We present a comprehensive and comparative analysis between neuron models with and without leak. Our experimental results reveal that leaky neuron model provides improved robustness and better generalization compared to models with no leak. However, leak decreases the sparsity of computation contrary to the common notion. Through a frequency domain analysis, we demonstrate the effect of leak in eliminating the high-frequency components from the input, thus enabling SNNs to be more robust against noisy spike-inputs.

In chapter 6, we present Spike-FlowNet, a deep hybrid neural network architecture integrating SNNs and ANNs for efficiently estimating optical flow from sparse event camera outputs. In addition, we present an input representation that efficiently encodes the sequences of sparse outputs from event cameras over time to preserve the spatio-temporal nature of spike events. The network is end-to-end trained with self-supervised learning on multi-vehicle stereo event camera dataset. Spike-FlowNet outperforms its corresponding ANN-based method in terms of the optical flow prediction capability while providing significant computational efficiency.

In chapter 7, we propose Fusion-FlowNet, a sensor/architecture fusion framework for accurately estimating dense optical flow. Accordingly, we leverage the complementary characteristics of event- and frame-based sensors as well as ANNs and SNNs. Our method generalizes well across distinct environments (rapid motion and challenging lighting conditions) and demonstrates state-of-the-art optical flow estimation performances. Furthermore, our network offers substantial savings in terms of the number of network parameters, computational energy and memory cost.

2. DEEP SPIKING CONVOLUTIONAL NEURAL NETWORK TRAINED WITH UNSUPERVISED SPIKE TIMING DEPENDENT PLASTICITY

2.1 Introduction

Spike-Timing-Dependent-Plasticity (STDP), one of the prominent mechanisms of learning in mammalian brains [21], is typically used for the unsupervised training of Spiking Neural Neworks (SNNs). SNNs, that have been widely explored for unsupervised pattern recognition, consist of input neurons fully-connected by plastic synapses to a layer of excitatory (output) neurons [13]–[15]. However, the two-layered SNN topology necessitates significantly larger number of excitatory neurons to attain competitive classification accuracy. In order to build an intelligent machine with improved scalability and reduced trainable parameters, it is imperative to devise a hierarchical SNN topology capable of extracting high-level features embedded in an image pattern and sharing learned features across different classes of patterns.

To this effect, we propose a deep Spiking Convolutional Neural Network (SpiCNN) composed of an input layer followed by a hierarchy of stacked convolutional layers for input feature extraction, a spatial-pooling layer for dimensionality reduction, and a fully-connected layer for final classification. Our proposal is inspired by the deep learning networks that exhibit state-of-the-art classification accuracies across a wide range of pattern recognition tasks while occasionally surpassing human performance [5], [6]. We train the convolutional weight kernels interconnecting consecutive layers in a sequential manner using STDP for self-learning distinctive features contained in the input patterns. The learned information is embedded in the average spiking rate of the neurons constituting the convolutional feature maps. We use the Poisson-distributed spike encoding mechanism for converting the average spiking rate of the neurons forming the input and convolutional feature maps to spike trains for training successive layers of SpiCNN. The chosen spike encoding scheme necessitates non-linear spiking neuronal model for competitive feature learning. Hence, we use the biologically plausible Leaky-Integrate-and-Fire (LIF) model [22] for the spiking neurons that offers rich

non-linear behavior. However, recent efforts on training deep spiking networks using STDP explored temporal rank-order spike encoding scheme, where pixel intensity is represented by the relative order of incoming spikes in a network of integrate-and-fire neurons [17], [23]–[25]. We note that Poisson-distributed spike encoding is more robust to intrinsic neuronal noise in comparison with the temporal rank-order scheme [26]. Moreover, rank-order coding incurs hardware overhead to modulate the weighted input spikes based on the respective order of firing. Poisson-distributed encoding, on the other hand, precludes this overhead since the individual spikes are statistically independent and information is encoded in the average neuronal firing rate. Further, the advances in nanotechnology have resulted in the emergence of neuro-mimetic devices capable of inherently mimicking dynamics of biological neurons and synapses [27]–[30]. We believe that the bio-inspired algorithms implemented using such emerging device technologies could pave the way for getting closer to the energy efficiency of the human brain.

The application of STDP learning to a network of LIF neurons using Poisson-distributed spike encoding scheme has thus far been limited to shallow SNN topologies due to the challenge associated with propagating spikes across multiple levels of hierarchy that is critical for feature learning. In this chapter, we stack multiple convolutional layers and demonstrate effective feature learning by precisely modulating the network hyper-parameters including the LIF neuronal and STDP learning parameters. In an effort to minimize the sensitivity of SpiCNN to various hyper-parameters and improve the feature learning capability, we propose using smaller convolutional weight kernels (for instance, 3×3 as opposed to the commonly used 5×5 or 7×7 kernel sizes) and training them using STDP-based mini-batch weight updates. The reduction in the number of trainable parameters offered by smaller kernel sizes together with STDP-based mini-batch weight updates enables the convolutional kernels to learn generalized features characterizing different input patterns. Our analysis indicates that smaller kernels learn prominent features and distributed internal representations across different layers, leading to improved classification accuracy. We comprehensively validate the efficacy of SpiCNN and the associated training methodology across different network topologies and kernel configurations using handwritten digits from the MNIST dataset [31] and natural images from the Caltech dataset [32].

Next, we provide the LIF neuron model, the proposed deep SpiCNN architecture and STDP-based unsupervised learning methodology. We subsequently outline the simulation framework and present experimental evidence illustrating the feature learning capability and robustness of SpiCNN. Finally, we highlight the benefits and trade-offs offered by our approach against recent works on deep SNNs.

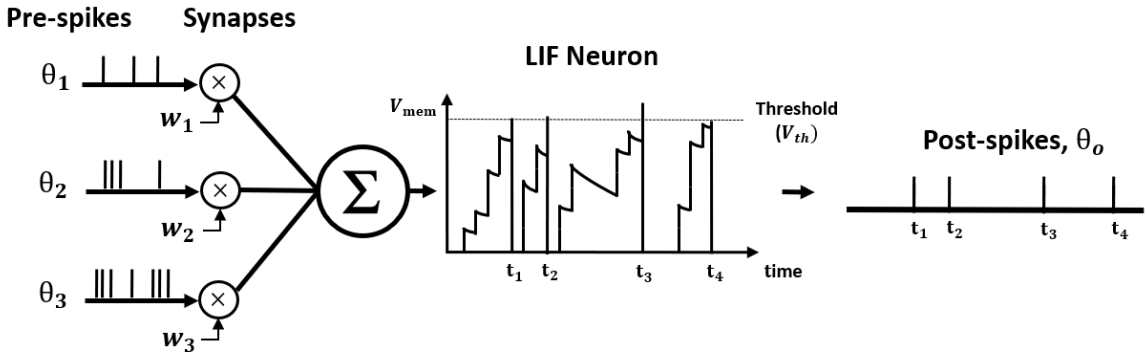


Figure 2.1. The operation of a Leaky Integrate and Fire (LIF) neuron.

2.2 SNN Fundamental: Leaky-Integrate-and-Fire Neuron Model

We first introduce the concept of Leaky-Integrate-and-Fire (LIF) neuron model [33] that is a fundamental and biologically plausible computational element for emulating the dynamics of biological neuronal functionalities. The sub-threshold dynamics of a LIF spiking neuron can be formulated as

$$\tau_m \frac{dV_{mem}}{dt} = -V_{mem} + I(t) \tag{2.1}$$

where V_{mem} is the post-neuronal membrane potential and τ_m is the time constant for membrane potential decay. The input current, $I(t)$, is defined as the weighted summation of pre-spikes at each time step as given below.

$$I(t) = \sum_{\mathrm{i}=1}^{n^l} (w_{\mathrm{i}} \sum_{k} \theta_{\mathrm{i}}(t - t_k)) \tag{2.2}$$

where n^l indicates the number of pre-synaptic weights, w_i is the synaptic weight connecting i^{th} pre-neuron to post-neuron. $\theta_\mathrm{i}(t-t_k)$ is a spike event from i^{th} pre-neuron at time t_k, which can be formulated as follows,

$$\theta(t-t_k) = \begin{cases} 1, & \text{if } t = t_k \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

where t_k is the time instant that k^{th} spike occurred. Fig.2.1 illustrates LIF neuronal dynamics. The impact of each pre-spike, $\theta_\mathrm{i}(t-t_k)$, is modulated by the corresponding synaptic weight (w_i) to generate a current influx to the post-neuron. Note, the units typically do not have bias term. The input current is integrated into the post-neuronal membrane potential (V_{mem}) that leaks exponentially over time with time constant (τ_m). When the membrane potential exceeds a threshold (V_{th}), the neuron generates a spike and resets its membrane potential to initial value (zero).

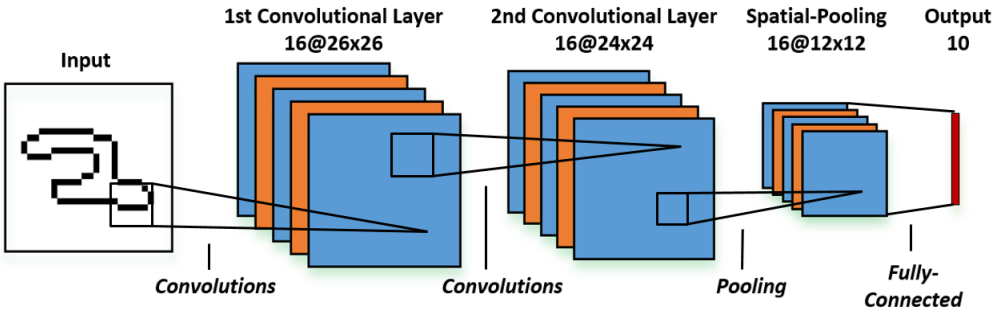


Figure 2.2. Architecture of the proposed deep SpiCNN consisting of the input layer, stacked convolutional layers for feature extraction, spatial-pooling layer for dimensionality reduction, and a final fully-connected layer for inference.

2.3 Proposed SNN Architecture and Learning Methodology

2.3.1 Deep Spiking Convolutional Neural Network (SpiCNN)

Our proposed SpiCNN (shown in Fig. 2.2) is composed of a hierarchy of stacked convolutional (C) layers followed by a spatial-pooling (S) layer and a final fully-connected (FC) layer.

The convolutional layers hierarchically extract characteristic features from the complex input image patterns. For instance, the first convolutional layer detects low-level features like edges and corners while the successive layers extract high-level features from the activation (feature) maps of the preceding layer. This is accomplished by training the shared weight kernels using the presented unsupervised convolutional STDP learning methodology. The learned information is embedded in the average spiking activity of the neurons forming the convolutional feature maps.

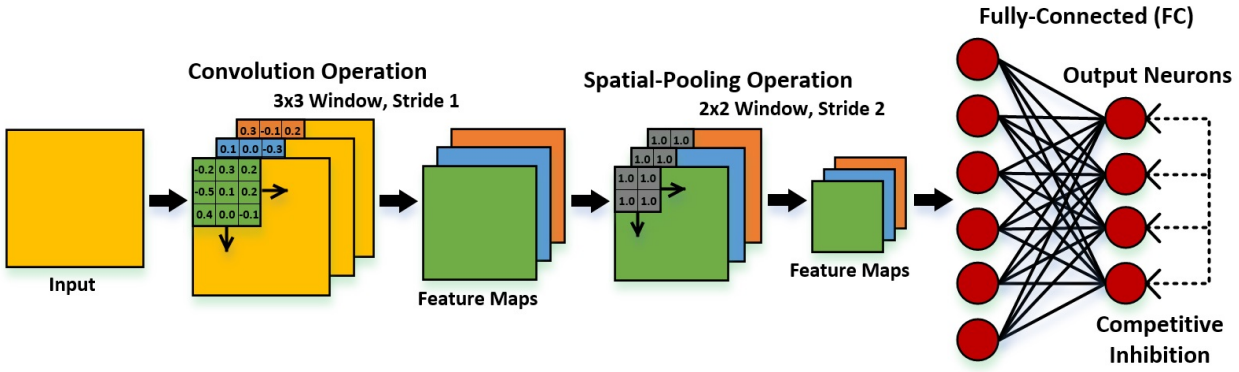


Figure 2.3. Illustration of the operation of the convolutional, spatial-pooling, and fully-connected layers. The input feature maps are convolved with the weight kernels to produce the respective output feature maps. The convolutional feature maps are then pooled spatially using max-pooling technique to achieve dimensionality reduction with minimal loss of spike information. The neurons in the max-pooled feature maps are unrolled and fully-connected to the output layer, which is augmented with competitive lateral inhibition for improved recognition capability.

Next, we have a spatial-pooling layer whose operation is detailed using a fixed 2×2 kernel with a stride of 2 pixels at a time. Spatial-pooling operation reduces the dimension of the convolutional feature maps while preserving the local correlation between the constituent pixels. In this chapter, we use average-pooling, which comprises each kernel weight of 1 and threshold of 0.25. During every stride of the kernel over a convolutional feature map, an output spike is fired by the corresponding neuron in the pooled feature map if any of the 4 input pixels spikes. This, in effect, reduces the dimension of the convolutional feature map by a factor of two with minimal loss of spike information. Note that the weight kernel used for spatial-pooling is not trainable and is fixed a priori. Spatial-pooling has the following two-

fold advantages. First, it enhances the computational efficiency by reducing the dimension of the convolutional feature maps. Second, it renders the network invariant to slight distortions and translation in the input patterns.

Finally, we have a fully-connected (output) layer containing as many neurons as the number of classes for a given recognition task. Each output neuron is fully-connected to the neurons constituting the feature maps in the preceding spatial-pooling layer, and is trained to infer the input patterns based on the extracted high-level representations. A test pattern is predicted to belong to the class represented by the output neuron with the highest spike-count during the presentation interval. In order to further the recognition capability of the fully-connected layer, we incorporate competitive lateral inhibition among the output neurons during inference as shown in Fig. 2.3. When an output neuron fires, membrane potentials of all other output neurons decrease with a constant inhibition factor. These inhibitory connections restrain the spiking activity of the neurons that have learned to infer input patterns sharing common features with the presented test pattern, thereby improving the recognition capability of SpiCNN.

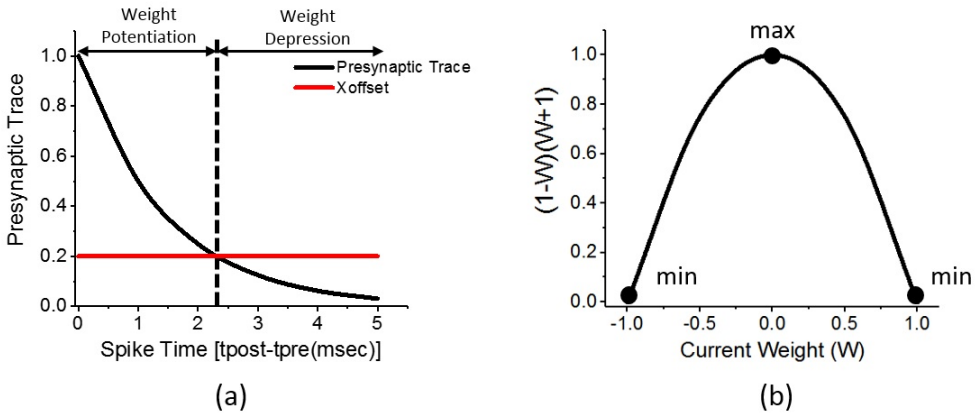


Figure 2.4. (a) Weight-dependent STDP rule used for capturing the temporal correlation between a pair of pre- and post-synaptic spikes. (b) Illustration of the non-linear dependence of synaptic weight update on the current weight given W_{max}=1, W_{min}=−1.

2.3.2 Synaptic Plasticity

The strength of the synapses interconnecting a pair of pre- and post-neurons is modulated using STDP, which postulates that the synaptic strength (weight) varies exponentially with the degree of timing correlation between the respective spike patterns. In this chapter, we use the weight-dependent positive-STDP rule that is illustrated in Fig. 2.4(a) and formulated below.

$$\Delta W = \eta(\mathrm{e}^{\frac{t_{pre}-t_{post}}{\tau}} - \chi_{offset})(W_{max} - W)(W - W_{min}) \tag{2.4}$$

where ΔW is the change in the synaptic weight, η is the learning rate governing the amount of weight update, t_{pre} and t_{post} are respectively the time instant of a pair of pre- and post-neuronal spikes, τ is the decay time constant, W is the current synaptic weight, and W_{max} (W_{min}) is the maximum (minimum) bound imposed on the synaptic weight. As depicted in Fig. 2.4(a), the presented positive-STDP learning rule uses only the positive timing window to measure the pre-post spike timing difference. The synaptic weights are potentiated/depressed by comparing the spike timing difference with a threshold (i.e. χ_{offset}). Synaptic potentiation is carried out for strong temporal correlation between a pair of pre- and post-spikes, i.e., if a pre-spike immediately causes the post-neuron to fire as determined by χ_{offset}. On the contrary, synaptic depression is carried out for larger spike-time differences. The weight updates are applied only at the time instants of post-synaptic spike and no weight change occurs at the time instants of pre-synaptic spike. The change in synaptic weight has a non-linear dependence on the current weight, constrained between W_{min} of -1 and W_{max} of $+1$, to achieve a gradual rise (decline) towards the maximum (minimum) bound. The non-linear factor, specified by the product of $(W_{max} - W)$ and $(W - W_{min})$ that simplifies to $(1 - W^2)$ and lies between 0 and 1, modulates the STDP-driven synaptic weight update given by $(\mathrm{e}^{\frac{t_{pre}-t_{post}}{\tau}} - \chi_{offset})$. As illustrated in Fig. 2.4(b), the non-linear factor ensures maximal STDP-driven update if the current weight is closer to zero and minimal STDP-driven update if the current weight is closer to the minimum or maximum bound.

Unsupervised Convolutional STDP Learning Methodology

We train the convolutional weights interconnecting successive pairs of layers of SpiCNN in a greedy layer-wise and unsupervised manner using STDP learning. At every time-step, the pre-neuronal spikes from the input feature maps are convolved with the respective weight kernels to generate a resultant current into the neurons constituting the output feature maps as depicted in Fig. 2.3. In the event of a post-neuronal spike, STDP-based updates are applied to the corresponding kernel weights. If several neurons making up an output feature map spike, as is typically observed, an average update is carried out on the shared kernel weights based on the respective pre- and post-neuronal spike times. In order to achieve efficient unsupervised learning, we propose uniform threshold adaptation across all neurons in a feature map as explained below. Whenever few neurons in a particular feature map fire, we increase the firing thresholds of all the neurons in the feature map by a constant value (ε) while decaying them exponentially over time. The proposed uniform threshold adaptation scheme effectively prevents few kernels from dominating learning and facilitates the weight kernels housed in the remaining feature maps to learn different features from other input patterns. In SpiCNN, we allow the kernels to have both positive and negative weights as illustrated in Fig. 2.3 for regulating the spiking activity of the neurons within a feature map. This precludes the need for explicit lateral inhibitory synaptic connections within a feature map that are typical in SNNs using only positive weights as demonstrated in [13], [17]. Even though negative weights are not biologically plausible, we incorporate them to achieve a reduction in the network complexity. These mechanisms collectively enable the weight kernel to self-learn unique input features. After one layer is trained, adjusted convolutional weights are frozen and firing thresholds of feature map are scaled by a constant factor (β) in order to increase spiking activities in the output feature map. We estimate the nonlinear transformation of input by feeding spikes from input through trained layers. Accordingly, following layer is trained by passing through regenerated spike events of nonlinear transformation of input pattern. This process is repeated until all the layers are trained.

The proposed greedy layer-wise unsupervised training methodology ensures that each layer receives sufficient input spikes to achieve efficient learning, thereby mitigating the issue of gradual decrease in spiking activity across layers that is inherent in deep SNNs [16]. However, extracting general characteristics of input patterns and sensitivity to hyper-parameters are a couple of key challenges that need to be addressed to achieve efficient learning. To this effect, we propose employing smaller 3×3 weight kernels instead of commonly used 5×5 or 7×7 kernels for deep SNNs. Our experimental analysis shows that smaller kernels trained using STDP-based mini-batch weight updates self-learn general input representations, which causes them to extract prominent characteristic features across successive convolutional layers. This leads to learn distributed internal representations compared to larger kernels across hidden convolutional layers of SpiCNN, which enhances the recognition capability of the final fully-connected layer. Furthermore, the decrease in the number of trainable parameters as a result of using smaller kernels together with performing mini-batch weight updates renders the feature learning efficiency less sensitive to the various network hyper-parameters. In mini-batch training, we compute the STDP-based weight updates individually for each input pattern in a randomly selected mini-batch of input patterns. We subsequently modify the kernels using weight updates averaged over the chosen mini-batch. Performing an average weight update (over a mini-batch) enables each kernel to extract features common to different classes of input patterns. Mini-batch training, in essence, causes general feature learning using fewer weight updates.

Supervised STDP Learning Methodology for Final Layer

Finally, we train the fully-connected layer in a supervised manner using the positive-STDP rule formulated in equation (2.1). For a given training pattern, the STDP-based weight updates are carried out on the output neuron that is pre-assigned to learn the particular input class. The firing thresholds of the remaining output neurons are momentarily raised to a high enough value that effectively prevents them from firing. The neuron that is supposed to learn the presented input pattern is guaranteed to spike at the beginning of the training period since it is initialized to a threshold of zero. Every time the neurons spike, we

increase their threshold by a small amount while exponentially decaying over time in order to maintain spiking activities during a course of the training period. Threshold adaptation ensures that the frequency of synaptic weight updates is high at the beginning of the training period and is gradually lowered, leading to efficient learning. When more than one output neuron fire in the final layer, the gradual threshold adaptations and competitive inhibitions take place together to regulate spiking activities and accentuate them between the output neurons.

2.4 Results

2.4.1 Simulation Framework

We evaluated the proposed SpiCNN using a MATLAB-based custom simulation framework. We pre-processed the original input image using the Laplacian of Gaussian (LoG) filter [34] to extract the contrasts among the image pixels and accentuate the edges. We truncated the LoG-filtered image pixels between 0 and 1, and subsequently converted them to Poisson-distributed spike trains whose firing rates are constrained between 0 and $400Hz$ for training the convolutional layers. We used a reduced maximum Poisson firing rate of $200Hz$ for training the final fully-connected layer. These input spike trains are kept active for a time period of $25msec$ during training and $300msec$ during inference assuming a simulation time-step of $1msec$. The synaptic weights are initialized randomly following a uniform distribution as formulated below.

$$W_{\mathrm{j,j+1}} \in U[-\sqrt{\frac{6}{n_{\mathrm{j}}+n_{\mathrm{j+1}}}}, \sqrt{\frac{6}{n_{\mathrm{j}}+n_{\mathrm{j+1}}}}] \tag{2.5}$$

where $W_{\mathrm{j,j+1}}$ is the synaptic weight matrix interconnecting layers j and j + 1, $U[-k,k]$ denotes a normal distribution in the interval between $-k$ and k, and n_{j} and $n_{\mathrm{j+1}}$ are the number of neurons in layers j and j + 1 respectively. Note that this weight initialization scheme is typically used in deep networks for breaking the symmetry between the units in each layer [35]. The neurons in the convolutional and fully-connected layers are modeled using Leaky-Integrate-and-Fire dynamics [22]. The STDP-based update on the synapse in-

terconnecting a pair of pre- and post-neurons is implemented by generating an exponentially decaying trace integrating the pre-spikes, and sampling it at the instant of a post-spike to update the synaptic weight. We trained the shared weight kernels using the presented unsupervised convolutional STDP learning methodology and validated its efficacy by evaluating the classification accuracy on an independent testing dataset. For a given test pattern, we accumulate the spike-count of the output neurons over a period of 300$msec$ and predict the test pattern to belong to the class represented by the output neuron with the highest spike count.

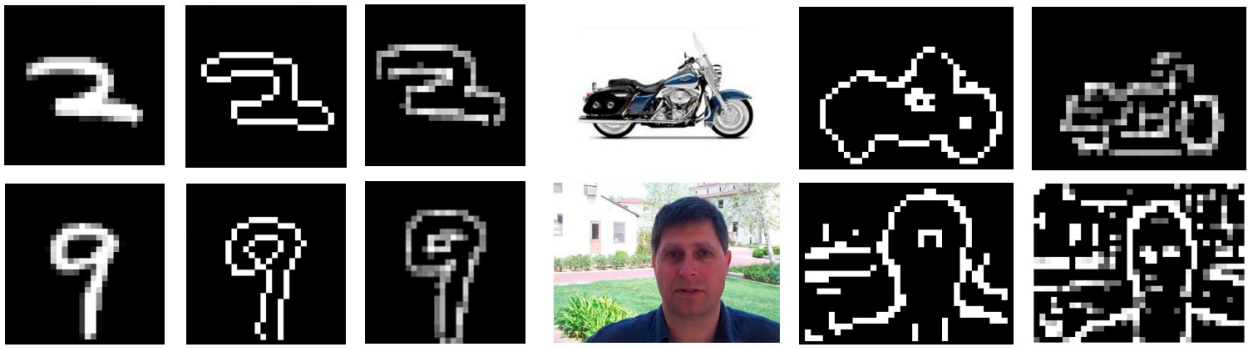


Figure 2.5. Few samples from the MNIST and Caltech datasets and the respective Laplacian of Gaussian (LoG) filtered binary and grayscale images.

Table 2.1. SpiCNN simulation parameters.

Parameters	**Values**
STDP Type	Nearest STDP
Synaptic Weight Range	[-1, 1]
Minimum Simulation Time-Step	1 $msec$
Decay Time Constant of Membrane Potential	100 $msec$
Decay Time Constant for Positive-STDP	1.5 $msec$
Decay Time Constant for Threshold	1000 $msec$
Training Duration	25 $msec$
Inference Duration	300 $msec$
Maximum Input Spiking Rate for Training	200 Hz – 400 Hz
Maximum Input Spiking Rate for Inference	200 Hz
Mini-batch Size	200 images
Convolutional Kernel Stride	1
Pooling Layer Stride	2
Neuronal Threshold Increase when Firing	1.5
Threshold Scaling Factor (β)	1.5
Lateral Inhibition Factor at Final Layer	0 – 5

2.4.2 MNIST Digit Recognition

We demonstrate the unsupervised feature extraction capability of deep SpiCNN by training it to infer handwritten MNIST digits [31]. The MNIST dataset consists of 60K training images and 10K testing images, each 28x28 in dimension and encoded in grayscale format. We LoG-filtered the original MNIST image and subsequently converted it to a binary image using a pre-defined pixel threshold. The LoG-filtered pixel intensities lower than the threshold including the negative values are truncated to 0 while those above the threshold are clipped to 1. Fig. 2.5 shows few samples from the input datasets and the respective LoG-filtered binary and grayscale images. It is important to note that the presented convolutional STDP learning methodology is capable of self-learning features both from the grayscale and binary images. Nevertheless, we begin our analysis by using binary images to determine the optimal network topology including the kernel size, number of kernels within a layer, and number of layers. We then train the optimal SpiCNN topology with both the grayscale and binary images, and present insights on how the choice of input encoding and spike encoding schemes impact the feature learning efficiency.

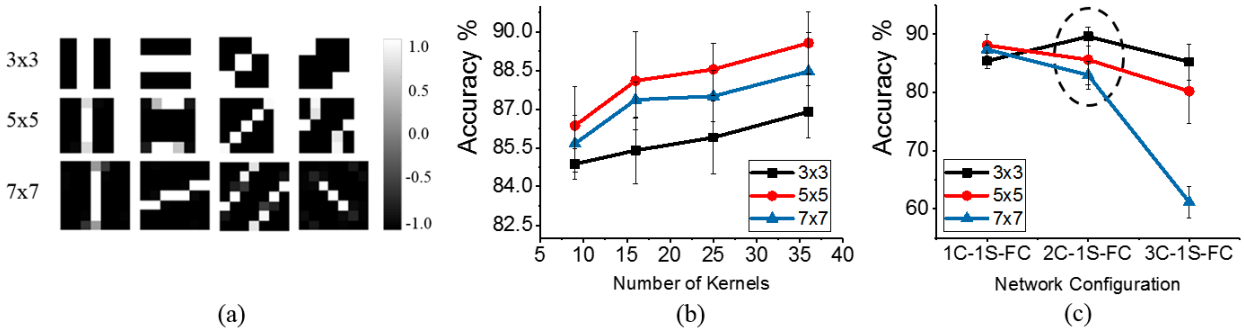


Figure 2.6. (a) Features self-learned by kernels of different sizes making up a 1C-1S-FC SpiCNN, composed of a single convolutional layer followed by a spatial-pooling layer and a fully-connected layer, trained for MNIST digit recognition. (b) Classification accuracy of 1C-1S-FC SpiCNN against the number of convolutional kernels for different kernel sizes. (c) Classification accuracy of various deep SpiCNN configurations with 16 feature maps per layer for different kernel sizes.

In our first experiment, we trained a shallow 1C-1S-FC SpiCNN, which is composed of a single convolutional layer (C) followed by a spatial-pooling (S) layer and a fully-connected

(FC) layer, across a range of kernel sizes and number of kernels making up the convolutional layer using the parameters listed in Table 2.1. Fig. 2.6(a) illustrates the general features including the horizontal, vertical, and diagonal edges acquired by the shared kernels, which is a testament to the efficacy of the presented convolutional learning methodology performing STDP-based mini-batch synaptic weight updates. We used a mini-batch size of 200 training images for enabling the convolutional kernels to self-learn shared features across different classes of input patterns. Our results (shown in Fig. 2.6(b)) indicate that the classification accuracy of the 1C-1S-FC SpiCNN, in general, increases with the number of kernels. Furthermore, we found that the larger 5×5 or 7×7 kernels outperform the 3×3 kernels by up to 2.0% in classification accuracy. This can be attributed to the ability of larger kernels to encode more features in such shallow topologies. The 1C-1S-FC SpiCNN with 36 5×5 kernels achieved a maximum classification accuracy of 89.5% on the MNIST testing dataset.

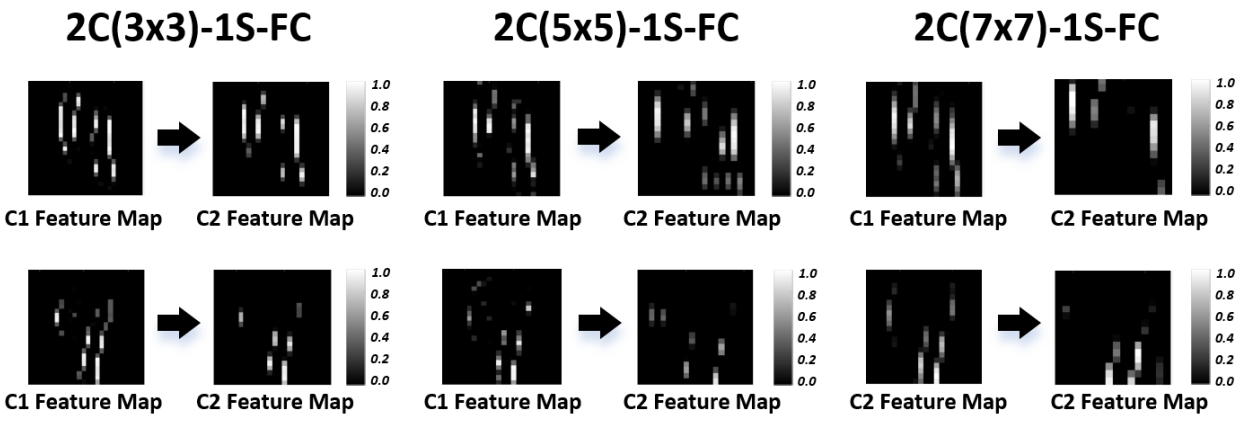


Figure 2.7. Feature maps at the output of every layer of a 2C-1S-FC SpiCNN for different kernel sizes.

In an attempt to further enhance classification accuracy, we stacked multiple convolutional layers to form a deep SpiCNN. We explored a couple of deep SpiCNN configurations, namely, 2C-1S-FC and 3C-1S-FC SpiCNN, consisting of two and three stacked convolutional layers respectively with 16 feature maps per layer. Our experimental analysis showed that deep SpiCNN with 3×3 kernels yielded improved classification accuracy over the one with 5×5 kernels while also performing significantly better than that with the larger 7×7 kernels as illustrated in Fig. 2.6(c). This is in stark contrast to the trend observed for shallow SpiC-

NNs (refer to Fig. 2.6(b)). The improved recognition capability of deep SpiCNN using 3×3 kernels stems from their ability to learn prominent features and distributed internal representations across successive convolutional layers and the ensuing increase in hidden spiking activity, as illustrated in Fig. 2.7 and Fig. 2.8. The smaller 3x3 kernels can learn generalized representations of input data as a result of having fewer trainable parameters, while larger kernels with a greater number of trainable parameters learn specific features that are less common to overall classes of input patterns. Given χ_{offset} is a threshold that is compared with the spike timing difference at the time instant of postsynaptic spikes, there is an optimal χ_{offset} value depending on kernel size to determine the sharpness of kernel shape and retain a certain amount of spiking activity. If χ_{offset} is set high, the convolutional kernel learns a sharper shape to incur a drastic decrease in spiking activity at the output feature map since they only detect a particular pattern. The inherent characteristic of the smaller kernel to learn generalized representations of input patterns allows the increase χ_{offset} to extract prominent features prevalent among different classes of input data. Conversely, a larger kernel should have lower χ_{offset} since it learns the specific features that produce a less discernable output feature maps which drastically decreases the spiking activity across successive layers. Given that a certain amount of spiking activity at the input feature maps of fully-connected layer is needed to reasonably infer the class of the input patterns, we adjusted χ_{offset} as illustrated in Table 2.2 in order to match the output spiking activity of successive convolutional layers by considering the kernel sizes and depth of network configurations.

Table 2.2. The learning parameter χ_{offset}.

Network Topology	**1C-1S-FC**			**2C-1S-FC**			**3C-1S-FC**		
Kernel Size	3×3	5×5	7×7	3×3	5×5	7×7	3×3	5×5	7×7
MNIST	0.27	0.26	0.23	0.23	0.17	0.15	0.225	0.16	0.14
Caltch (Face/Motorbike)	0.32	0.27	0.23	0.27	0.22	0.21	0.28	0.21	0.2

As depicted in Fig. 2.8, the output spiking activities after the successive convolutional layers are similar for all 2C-1S-FC SpiCNN configurations, but the spiking activities at the output of the first convolutional layer depends on the convolutional kernel sizes. The results show that 3x3 and 5x5 kernels produces higher spiking activities than the 7x7 kernels at the output of the first convolutional layer, which indicates the degree of resultant fea-

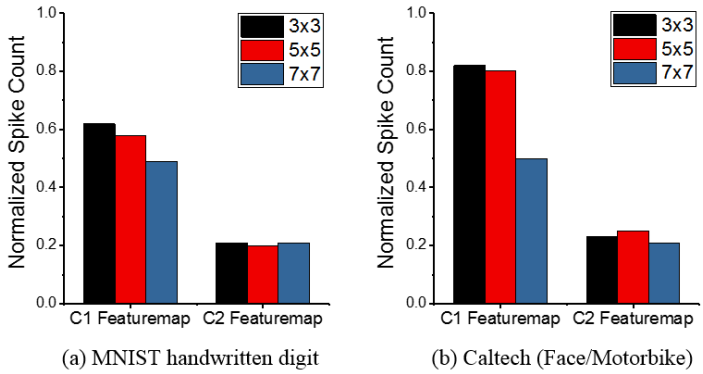


Figure 2.8. Normalized Spiking activity of feature maps at the output of every layer of a 2C-1S-FC SpiCNN for different kernel sizes for (a) MNIST handwritten digit and (b) Caltech (Face/Motorbike) datasets.

tures captured from the input patterns. In other words, the smaller kernel learns prominent features that are common to overall output classes and constructs distributed internal representations across successive layers as illustrated in Fig. 2.7. Note that researchers in [36] have validated the efficacy of smaller 3×3 kernels in the traditional deep learning networks. Hence, the 2C-1S-FC SpiCNN with 3×3 kernels across both the convolutional layers containing the same number of feature maps offered the highest accuracy of 91.1%. Last, we note that the classification accuracy degrades by stacking an additional convolutional layer (refer to 3C-1S-FC SpiCNN in Fig. 2.6(c)). This is common in deep networks, where the optimal network depth depends on the target application. For MNIST digit recognition, we experimentally determined 2C-1S-FC SpiCNN (with 3×3 kernels and 16 feature maps per layer) as the optimal SpiCNN topology. Our analysis, shown in Fig. 2.6, further reveals that 1C-1S-FC SpiCNNs have improved the robustness for randomly initialized weights as the number of kernel increases across 5 simulation runs. On the other hand, networks with more hidden layers are comparatively sensitive to weight initialization with a standard deviation of 1.3%, 2.4%, and 3.8% respectively in classification accuracy for SpiCNN composed of one, two and three stacked convolutional layers.

Finally, we trained the 2C-1S-FC SpiCNN using grayscale images instead of binary images used for the previous analysis. We note that a grayscale image intrinsically encodes more information compared to its binary counterpart, which can be translated to the spiking

domain effectively using the Poisson-distributed spike encoding scheme. However, our results indicated nearly 5% degradation in the classification accuracy using grayscale images relative to that achieved with binary images. We hypothesize that this could be an artifact of the input dataset, where the precise location of the edges in the MNIST data carries more significance than the absolute pixel intensities at the respective locations. As mentioned earlier, all the edge pixels carry equal significance in the binary image irrespective of the absolute intensities (contained in the grayscale image), which is effectively translated to the spiking domain by using the same Poisson firing rate for all the edge pixels. Thus, the attributes of the input dataset determine which image type between binary and grayscale yields better classification accuracy. In order to validate our hypothesis, we demonstrate the utility of rich grayscale inputs and Poisson-distributed spike encoding scheme using natural real-world images in the subsequent analysis.

2.4.3 Caltech Image Recognition

We used a subset of 1226 images from the Caltech dataset[32] spanning two different object categories, namely, Face and Motorbike, for further illustrating the applicability of the proposed SpiCNN and the unsupervised convolutional STDP learning methodology. We randomly chose 200 images from each object category for training and the remaining for testing SpiCNN. Each individual Caltech image, originally encoded in high dimensional RGB colorspace, was LoG-filtered to obtain a single channel grayscale image with edges accentuated. The resultant grayscale image was resized and converted to a binary image of reduced dimension (28x36). We similarly use the binary images for determining the optimal topology, and finally highlight the benefits of directly training with the grayscale images for such real-world images.

We initially trained a 1C-1S-FC SpiCNN for different kernel sizes and number of kernels making up the convolutional layer. We observed that the feature representations acquired by certain kernels (for instance, the horizontal, vertical, and diagonal features in Fig. 2.9(a)) are similar to those learned from the MNIST digits (refer to Fig. 2.7(a)). This highlights the general feature learning capability of the presented convolutional STDP learning methodol-

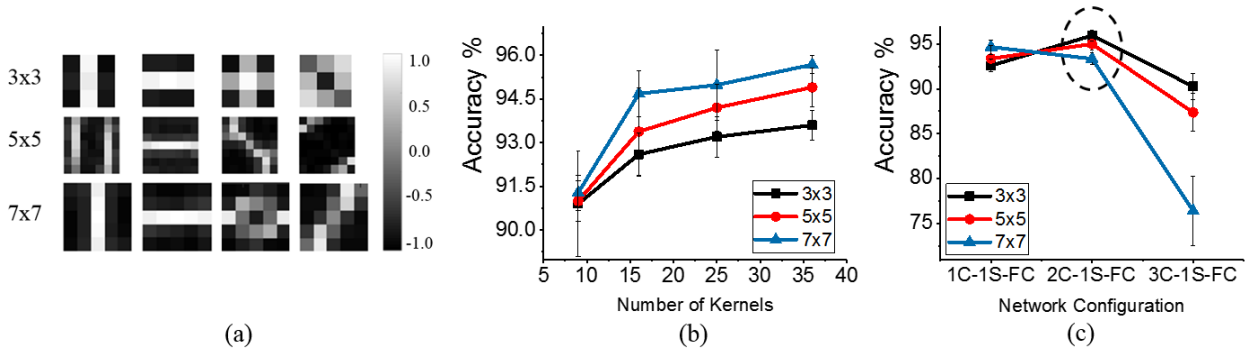


Figure 2.9. (a) Features self-learned by kernels of different sizes making up a 1C-1S-FC SpiCNN trained for Caltech image recognition. (b) Classification accuracy of 1C-1S-FC SpiCNN against the number of convolutional kernels for different kernel sizes. (c) Classification accuracy of various deep SpiCNN configurations with 16 feature maps per layer for different kernel sizes.

ogy. Our results (shown in Fig. 2.9(b)) indicate that the classification accuracy increases with the number of kernels with a kernel size of 7×7 yielding the highest accuracy of 95.6%. In an effort to achieve improved classification accuracy in a scalable manner, we explored the deep 2C-1S-FC and 3C-1S-FC SpiCNN configurations with 16 feature maps per layer. Our experimental analysis offered the following twin insights. First, the smaller 3×3 kernels performed better than the larger 5×5 and 7×7 kernels for deep SpiCNNs (as illustrated in Fig. 2.9(c)) owing to extract prominent features and construct distributed representation across successive hierarchical layers as shown in Fig. 2.8. Second, the 2C-1S-FC SpiCNN was found to be the optimal topology with a classification accuracy of 96.0% for the chosen Caltech image recognition task.

In our final experiment, we trained the optimal 2C-1S-FC SpiCNN directly using the grayscale images and achieved an improved accuracy of 97.6%. This is contrary to the trend we observed for MNIST dataset and essentially corroborates our hypothesis regarding the usefulness of the rich grayscale information for self-learning distinctive features from complex natural images. It is important to note that the chosen Poisson-distributed spike encoding scheme is capable of efficiently translating the grayscale information into spike trains, leading to improved feature learning by deep SpiCNN.

2.5 Discussion and Comparison with Related Works

Our proposed deep SpiCNN achieves competitive classification accuracy across different datasets using fewer trainable parameters in comparison with typical fully-connected (shallow) SNNs. For instance, SpiCNN composed of two convolutional layers with 16 feature maps per layer provided a classification accuracy of 91.1% for MNIST digit recognition. On the other hand, the fully-connected SNN presented in [13] required 1600 excitatory (output) neurons amounting to 50× more trainable parameters to attain an equivalent accuracy. Furthermore, the fully-connected SNN was trained with a unit batch size to enable every excitatory neuron to learn a complete representation of a unique input pattern. Conversely, we used mini-batch learning that achieves feature learning with fewer synaptic weight updates and facilitates every convolutional kernel to self-learn features shared across different classes of input patterns. Furthermore, mini-batch learning help the network avoid drastic changes because of the individual training data that is far from generality. To these effect, the mini-batch learning renders convolutional kernels to efficiently learn better features that contain the generalized characteristics. Hence, SpiCNN yields competitive accuracy with fewer synaptic weight updates.

Table 2.3. Classification accuracy of 2C-1S-FC SpiCNN on the MNIST training and testing dataset.

Kernel Size	**3×3**	**5×5**	**7×7**
Training Accuracy	90.5%	84.3%	80.6%
Testing Accuracy	91.1%	85.0%	82.4%

Deep SNNs varying in degrees of bio-fidelity have been proposed in the literature. In [37] and [38], a deep SNN is proposed which is trained offline using the backpropagation algorithm as an Analog Neural Network (ANN). The ANN-to-SNN conversion merely exploits the event-driven processing capability of SNNs for achieving energy efficiency during inference while trading off the on-chip learning capability. Furthermore, mapping the trained weights of an ANN to the corresponding SNN leads to a loss in the classification accuracy. Researchers in [16], [18], [19] directly trained deep SNNs through spike events using neuronal

spiking rate-based and membrane potential-based error backpropagation algorithm, respectively. Nevertheless, the computational complexity incurred during training hinders on-chip implementation. Our deep SpiCNN is trained using bio-inspired STDP-based unsupervised learning that harnesses the event-driven processing and on-chip learning capabilities. Furthermore, as depicted in Table 2.3, estimated training accuracies of 2C-1S-FC SpiCNN are not higher than testing accuracies on handwritten digit (MNIST) datasets. SpiCNNs trained by greedy layer-wise unsupervised STDP learning are less subjective to an overfitting phenomenon, which is commonly observed in supervised backpropagation algorithm.

Table 2.4. Classification accuracy of SNNs for Caltech (Face/Motorbike).

SNN topology	Architecture	Spike Encoding Scheme	Learning Rule	#Trainable Parameters	Accuracy
SDNN [17]	Convolutional	Rank-order encoding	STDP + SVM	25480	99.1%
Spiking CNN [25]	Convolutional	Rank-order encoding	STDP + RBF	20480	97.7%
Spiking CNN [39]	Convolutional	Rank-order encoding	Reinforcement STDP	23120	98.9%
SpiCNN (This work)	Convolutional	Poisson-distributed spike encoding	STDP	25488	97.6%

Table 2.5. Classification accuracy of SNNs for MNIST digit recognition.

SNN topology	Architecture	Spike Encoding Scheme	Learning Rule	#Trainable Parameters	Accuracy
Two-layer SNN [13]	Fully-connected	Poisson-distributed spike encoding	STDP	5017600	95%
SDNN [17]	Convolutional	Rank-order encoding	STDP + SVM	76500	98.4%
Spiking CNN [40]	Convolutional	Poisson-distributed spike encoding	Sparse Coding + STDP + SVM	590642	98.3%
SpiCNN (This work)	Convolutional	Poisson-distributed spike encoding	STDP	25488	91.1%

Recent works have explored the use of STDP for training multi-layer SNNs. In [25], an illustration of the applicability of STDP-based visual feature learning on SNNs with single convolutional layer is shown while reference [17] successfully trained SNNs with multiple convolutional layers. Both [25] and [17] used temporal rank-order spike encoding scheme in a network of integrate-and-fire neurons and trained convolutional layers for input feature extraction using STDP and ANN-based classifier of Support Vector Machine or Radial Basis Function, respectively, to evaluate the effectiveness of extracted features. Our work differs from that presented in [25] and [17] in the following respects. First, we use Poisson-distributed spike encoding scheme that requires non-linear Leaky-Integrate-and-Fire (LIF) dynamics for better regulating the neuronal spiking activity. The chosen spike encoding scheme is implementation friendly since the individual spike events are uncorrelated, which requires the LIF neurons to simply integrate the weighted sum of incoming spikes. How-

ever, the rank order spike encoding scheme necessitates hardware overhead to account for the relative order of spikes emitted by the individual neurons. This is illustrated in [23], which shows a feed-forward SNN with inhibitory inter-neurons that reduce the effectiveness of the input spikes feeding the excitatory neurons based on the respective order of firing. Additionally, temporal rank-order spike encoding scheme is sensitive to variability in spike timings caused by intrinsic neuronal noise typical in custom hardware realizations using analog-CMOS or emerging technologies [26]. Second, we demonstrate the ability of smaller 3×3 kernels in all the convolutional layers containing the same number of feature maps to learn generalized prominent features as a result of having fewer trainable parameters. Last, we perform mini-batch learning that leads to general feature learning using fewer synaptic weight updates. Table 2.4 shows that SpiCNN offers comparable classification accuracy for Caltech (Face/Motorbike) image recognition. However, the classification accuracy of SpiCNN is lower than that reported in [17] and [40] for MNIST digit recognition as shown in Table 2.5. However, related works [17], [40] used ANN-based readout for final classification upon training the convolutional layers with STDP while we train all the layers of SpiCNN including the final classification layer using greedy layer-wise STDP through direct spike events. It is important to note that SpiCNN offers competitive classification accuracy with fewer trainable parameters as illustrated in Table 2.5. Future works could explore enhanced learning mechanisms like reward-modulated STDP to further improve the performance of multi-layer SNNs without using external classifiers as demonstrated in [39], [41], [42].

2.6 Conclusion

The application of STDP-based unsupervised learning has been primarily limited to shallow fully-connected SNN topologies, which necessitates a large number of trainable parameters to achieve competitive classification accuracy. In this chapter, we propose SpiCNN composed of a hierarchy of stacked convolutional layers followed by a spatial-pooling layer and a fully-connected layer for self-learning input features using fewer trainable parameters. We hierarchically train the shared synaptic weight kernels interconnecting successive convolutional layers using STDP for unsupervised feature extraction. Furthermore, we

demonstrate improved feature learning using smaller 3×3 kernels trained with STDP-based mini-batch synaptic weight updates. We validate the efficacy of the presented unsupervised convolutional STDP learning methodology by training deep SpiCNN to effectively recognize handwritten MNIST digits and natural Caltech images. The reduction in the number of trainable parameters (for smaller 3×3 kernels) and the frequency of synaptic weight updates (as a result of mini-batch training) coupled with the use of robust Poisson-distributed spike encoding scheme (for layer-wise training) render SpiCNN amenable for energy-efficient neuromorphic hardware implementations.

3. ENABLING SPIKE-BASED BACKPROPAGATION IN STATE-OF-THE-ART DEEP NEURAL NETWORK ARCHITECTURES

3.1 Introduction

Spiking Neural Networks (SNNs) has recently emerged as a prominent neural computing paradigm. However, the typical shallow spiking network architectures have limited capacity for expressing complex representations, while training a very deep spiking network has not been successful so far. Diverse methods have been proposed to get around this issue such as converting off-line trained deep Analog Neural Networks (ANNs) to SNNs. However, ANN-SNN conversion scheme fails to capture the temporal dynamics of a spiking system. On the other hand, it is still a difficult problem to directly train deep SNNs using input spike events due to the discontinuous and non-differentiable nature of spike generation function. To overcome this problem, we propose an approximate derivative method that accounts for leaky behavior of LIF neuron. This method enables to train deep convolutional SNNs with input spike events using spike-based backpropagation algorithm. Our experiments show the effectiveness of the proposed spike-based learning strategy on state-of-the-art deep networks (VGG and Residual architectures) by achieving the best classification accuracies in MNIST, SVHN and CIFAR-10 datasets compared to other SNNs trained with spike-based learning.

The main contributions of this work are specified as follows.

- First, we develop a spike-based supervised gradient descent BP algorithm that exploits a conditionally differentiable approximated activation function of LIF neuron.
- In addition, we leverage the key idea of the successful deep ANN models such as LeNet5 [1], VGG [36] and ResNet [5] for efficiently constructing state-of-the-art deep SNN network architectures. We also adapt dropout [2] technique in order to better regularize deep SNN training.
- Next, we demonstrate the effectiveness of our methodology for visual recognition tasks on standard character and object datasets (MNIST, SVHN, CIFAR-10) and a neuro-

morphic dataset (N-MNIST). To the best of our knowledge, this work achieves the best classification accuracy in MNIST, SVHN and CIFAR-10 datasets through training deep SNNs.

- Lastly, we expand our efforts to quantify and analyze the advantages of spike-based BP algorithm compared to ANN-SNN conversion techniques in terms of inference time and energy consumption.

The rest of this chapter is organized as follows. First, we provide the background on the architectures of deep convolutional SNNs. Second, we detail the spike-based gradient descent backpropagation learning algorithm and describe the spiking version of dropout technique used for this work. Next, we describe the experiments and report the simulation results, which validate the efficacy of spike-based BP training for MNIST, SVHN, CIFAR-10 and N-MNIST datasets. Subsequently, we discuss the proposed algorithmin comparison to relevant works. Then, we analyze the spike activity, inference speedup and complexity reduction of direct-spike trained SNNs and ANN-SNN converted networks. Finally, we summarize and conclude the work.

3.2 Deep Convolutional Spiking Neural Network

3.2.1 Building Blocks

In this chapter, we develop a training methodology for convolutional SNN models that consist of an input layer followed by intermediate hidden layers and a final classification layer. In the input layer, the pixel images are encoded as Poisson-distributed spike trains where the probability of spike generation is proportional to the pixel intensity. The hidden layers consist of multiple convolutional (C) and spatial-pooling (P) layers, which are often arranged in an alternating manner. These convolutional (C) and spatial-pooling (P) layers represent the intermediate stages of feature extractor. The spikes from the feature extractor are combined to generate a one-dimensional vector input for the fully-connected (FC) layers to produce the final classification. The convolutional and fully-connected layers contain trainable parameters (*i.e.* synaptic weights) while the spatial-pooling layers are fixed *a*

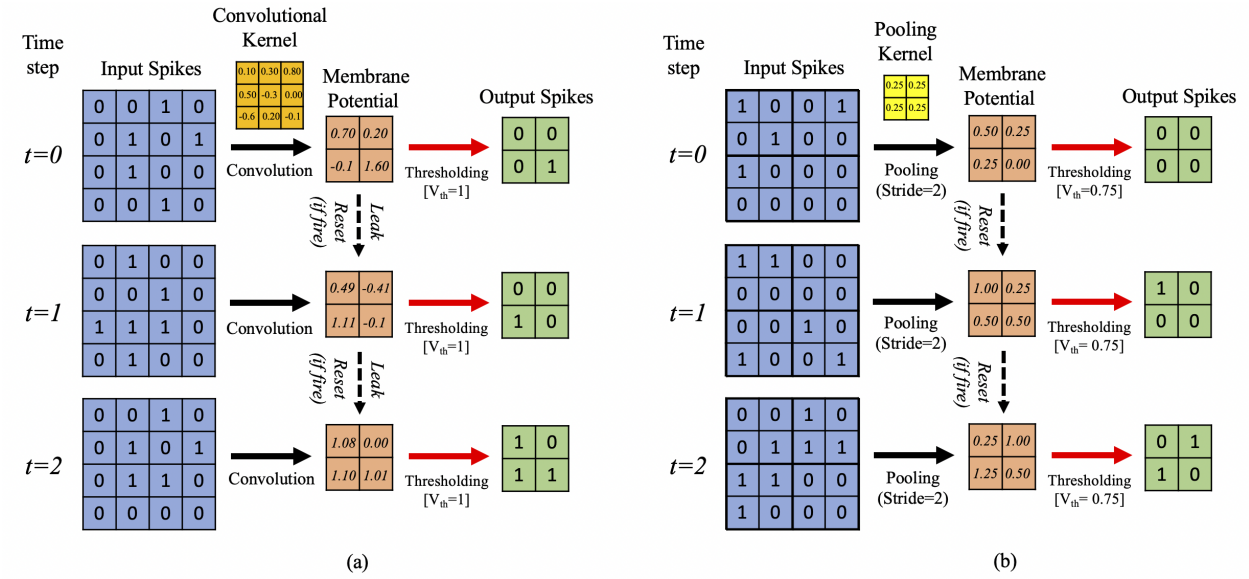


Figure 3.1. Illustration of the simplified operational example of (a) convolutional, (b) spatial-pooling layers (assuming 2-D input and 2-D weight kernel) over three time steps. At each time step, the input spikes are convolved with the weight kernel to generate the current influx, which is accumulated in the post-neuron's membrane potential, V_{mem}. Whenever the membrane potential exceeds the firing threshold (V_{th}), the post-neuron in the output feature map spikes and V_{mem} resets. Otherwise, V_{mem} is considered as residue in the next time step while leaking in the current time step. For spatial-pooling, the kernel weights are fixed, and there is no membrane potential leak.

priori. Through the training procedure, weight kernels in the convolutional layers can encode the feature representations of the input patterns at multiple hierarchical levels. Fig. 3.1(a) shows the simplified operational example of a convolutional layer consisting of LIF neurons over three time steps (assuming 2-D input and 2-D weight kernel). On each time step, each neuron convolves its input spikes with the weight kernel to compute its input current, which is integrated into its membrane potential, V_{mem}. If $V_{mem} > V_{th}$, the neuron spikes and its V_{mem} is set to 0. Otherwise, V_{mem} is considered as residue in the next time step while leaking in the current time step. Fig. 3.1(b) shows the simplified operation of a pooling layer, which reduces the dimensionality from the previous convolutional layer while retaining spatial (topological) information.

There are various choices for performing the spatial-pooling operation in the ANN domain. The The two major operations used for pooling are max and average. Both have been

used for SNNs, e.g., max-pooling [12] and average-pooling [37], [43]. We use average-pooling due to its simplicity. In the case of SNNs, an additional thresholding is used after averaging to generate output spikes. For instance, a fixed 2×2 kernel (each having a weight of 0.25) strides through a convolutional feature map without overlapping and fires an output spike at the corresponding location in the pooled feature map only if the sum of the weighted spikes of the 4 inputs within the kernel window exceeds a designated firing threshold (set to 0.75). Otherwise, the membrane potential remains as a residue in the next time step. Fig. 3.1(b) shows an example spatial-pooling operation over three time steps (assuming 2-D input and 2-D weight kernel). The average-pooling threshold need to be carefully set so that spike propagation is not disrupted due to the pooling. If the threshold is too low, there will be too many spikes, which can cause loss of spatial location of the feature that was extracted from the previous layer. If the threshold is too high, there will not be enough spike propagation to the deeper layers.

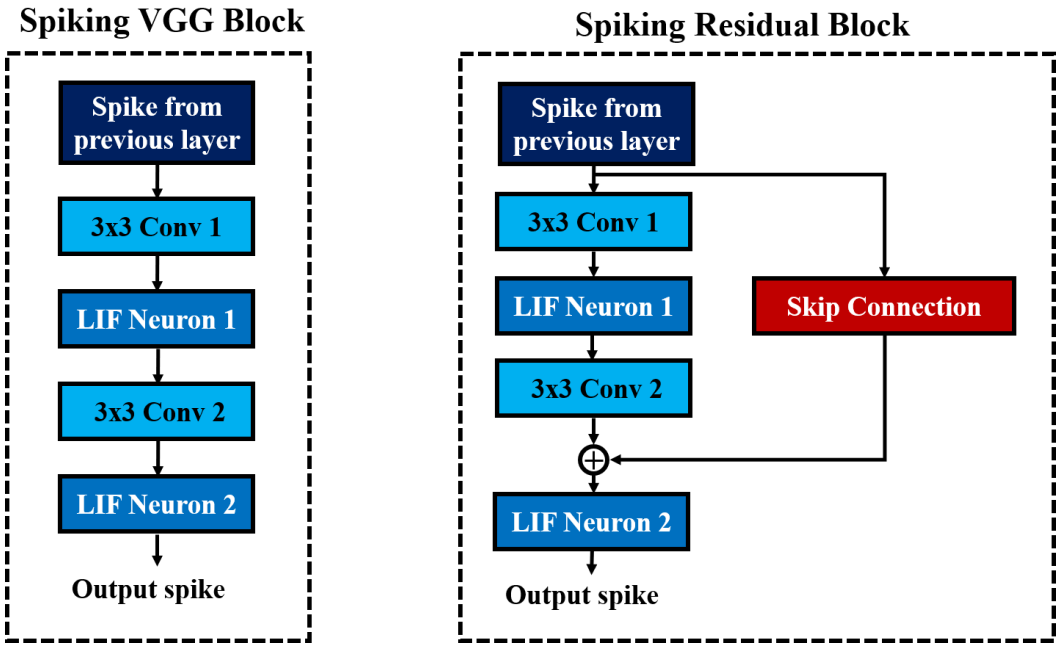


Figure 3.2. The basic building blocks of the described convolutional SNN architectures. (a) Spiking VGG Block. (b) Spiking ResNet Block.

3.2.2 Deep Convolutional SNN architecture: VGG and Residual SNNs

Deep networks are essential for recognizing intricate input patterns so that they can effectively learn hierarchical representations. To that effect, we investigate popular deep neural network architectures such as VGG [36] and ResNet [5] in order to build deep SNN architectures. VGG [36] was one of the first neural networks, which used the idea of using small (3×3) convolutional kernels uniformly throughout the network. Using small kernels enables effective stacking of convolutional layers while minimizing the number of parameters in deep networks. In this chapter, we build deep convolutional SNNs (containing more than 5 trainable layers) using 'Spiking VGG Block's, which contain stacks of convolutional layers using small (3×3) kernels. Fig. 3.2(a) shows a 'Spiking VGG block' containing two stacked convolutional layers, each followed by a LIF neuronal layer. The convolutional layer box contains the synaptic connectivity, and the LIF neuronal box contains the activation units. Next, ResNet [5] introduced the skip connections throughout the network that had considerable successes in enabling successful training of significantly deeper networks. In particular, ResNet addresses the degradation (of training accuracy) problem [5] that occurs while increasing the number of layers in the standard feedforward neural network. We employ the concept of skip connection to construct deep residual SNNs with 7-11 trainable layers. Fig. 3.2(b) shows a 'Spiking Residual Block' containing non-residual and residual paths. The non-residual path consists of two convolutional layers with an intermediate LIF neuronal layer. The residual path (skip connection) is composed of the identity mapping when the number of input and output feature maps are the same, and 1×1 convolutional kernels when the number of input and output feature maps differ. The outputs of both the non-residual and residual paths are integrated to the membrane potential in the last LIF neuronal activation layer (LIF Neuron 2 in Fig. 3.2(b)) to generate output spikes from the 'Spiking Residual Block'. Within the feature extractor, a 'Spiking VGG Block' or 'Spiking Residual Block' is often followed by an average-pooling layer. Note, in some 'Spiking Residual Blocks', the last convolutional and residual connections employ convolution with a stride of 2 to incorporate the functionality of the spatial-pooling layers. At the end of the feature

extractor, extracted features from the last average-pooling layer are fed to a fully-connected layer as a 1-D vector input for inference.

3.3 Supervised Training of Deep Spiking Neural Network

3.3.1 Spike-based Gradient Descent Backpropagation Algorithm

The spike-based BP algorithm in SNN is adapted from standard BP [44] in the ANN domain. In standard BP, the network parameters are iteratively updated in a direction to minimize the difference between the final outputs of the network and target labels. The standard BP algorithm achieves this goal by back-propagating the output error through the hidden layers using gradient descent. However, the major difference between ANNs and SNNs is the dynamics of neuronal output. An analog neuron (such as *sigmoid*, *tanh*, or $ReLU$) communicates via continuous values whereas a spiking neuron generates binary spike outputs over time. In SNNs, spatiotemporal spike trains are fed to the network as inputs. Accordingly, the outputs of spiking neuron are spike events, which are also discrete over time. Hence, the standard BP algorithm is incompatible with training SNNs, as it can not back-propagate the gradient through a non-differentiable spike generation function. In this chapter, we formulate an approximate derivative for LIF neuron activation, making gradient descent possible. We derive a spike-based BP algorithm that is capable of learning spatiotemporal patterns in spike-trains. The spike-based BP can be divided into three phases, forward propagation, backward propagation and weight update, which we describe in the following sections.

Forward Propagation

In forward propagation, spike trains representing input patterns are presented to the network for estimating the network outputs. To generate the spike inputs, the input pixel values are converted to Poisson-distributed spike trains and delivered to the network. The input spikes are multiplied with synaptic weights to produce an input current that accumulates in the membrane potential of post neurons as in equation (1.1-1.3). Whenever its membrane potential exceeds a neuronal firing threshold, the post-neuron generates an output spike and

resets. Otherwise, the membrane potential decays exponentially over time. The neurons of every layer (excluding output layer) carry out this process successively based on the weighted spikes received from the preceding layer. Over time, the total weighted summation of the pre-spike trains (i.e., net) is described as follows,

$$net_{\mathrm{j}}^{l}(t) = \sum_{\mathrm{i}=1}^{n^{l-1}} (w_{\mathrm{ij}}^{l-1} x_{\mathrm{i}}^{l-1}(t)),\ where\ x_{\mathrm{i}}^{l-1}(t) = \sum_{t}\sum_{k} \theta_{\mathrm{i}}^{l-1}(t - t_k) \tag{3.1}$$

where $net_{\mathrm{j}}^{l}(t)$ represents the total current influx integrated to the membrane potential of j^{th} post-neuron in layer l over the time t, n^{l-1} is the number of pre-neurons in layer l-1 and $x_{\mathrm{i}}^{l-1}(t)$ denotes the sum of spike train ($t_k \leq t$) from i^{th} pre-neuron over time t. The sum of post-spike trains ($t_k \leq t$) is represented by $a_{\mathrm{j}}^{l}(t)$ for the j^{th} post-neuron.

$$a_{\mathrm{j}}^{l}(t) = \sum_{t}\sum_{k} \theta_{\mathrm{j}}^{l}(t - t_k) \tag{3.2}$$

Clearly, the sum of post-spike train ($a^l(t)$) is equivalent to the sum of pre-spike train ($x^l(t)$) for the next layer. On the other hand, the neuronal firing threshold of the final classification layer is set to a very high value so that final output neurons do not spike. In the final layer, the weighted pre-spikes are accumulated in the membrane potential while decaying over time. At the last time step, the accumulated membrane potential is divided by the number of total time steps (T) in order to quantify the output distribution ($output$) as presented by equation (3.3).

$$output = \frac{V_{mem}^{L}(T)}{number\ of\ timesteps} \tag{3.3}$$

Backward Propagation and Weight Update

Next, we describe the backward propagation for the proposed spike-based backpropagation algorithm. After the forward propagation, the loss function is measured as a difference between target labels and outputs predicted by the network. Then, the gradients of the loss function are estimated at the final layer. The gradients are propagated backward all

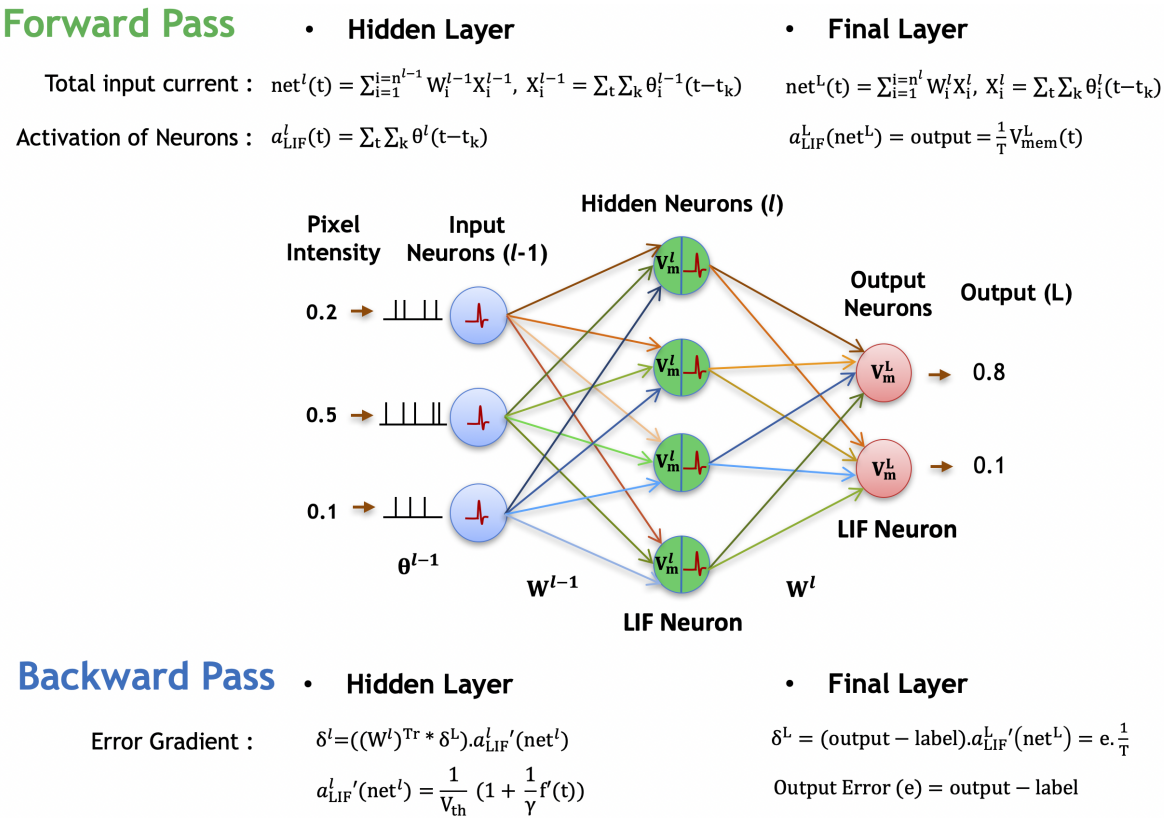


Figure 3.3. Illustration of the forward and backward propagation phase of the proposed spike-based BP algorithm in a multi-layer SNN comprised of LIF neurons.

the way down to the input layer through the hidden layers using recursive chain rule, as formulated in equation (3.4). The following equations (3.4 - 3.24) and Fig. 3.3 describe the detailed steps for obtaining the partial derivatives of (final) output error with respect to weight parameters.

The prediction error of each output neuron is evaluated by comparing the output distribution (*output*) with the desired target label (*label*) of the presented input spike trains, as shown in equation (3.5). The corresponding loss function (E in equation (3.6)) is defined as the sum of squared (final prediction) error over all output neurons. To calculate the $\frac{\partial E}{\partial a_{LIF}}$ and $\frac{\partial a_{LIF}}{\partial net}$ terms in equation (3.4), we need a defined activation function and a method to differentiate the activation function of a LIF neuron.

$$\frac{\partial E}{\partial w^l} = \frac{\partial E}{\partial a_{LIF}} \frac{\partial a_{LIF}}{\partial net} \frac{\partial net}{\partial w^l} \tag{3.4}$$

$$Final\ output\ error,\ \mathrm{e_j} = output_{\mathrm{j}} - label_{\mathrm{j}} \tag{3.5}$$

$$Loss\ function,\ E = \frac{1}{2}\sum_{\mathrm{j}=1}^{n^L} \mathrm{e_j}^2 \tag{3.6}$$

In SNN, the 'activation function' indicates the relationship between the weighted summation of pre-spike inputs and post-neuronal outputs over time. In forward propagation, we have different types of neuronal activation for the final layer and hidden layers. Hence, the estimation of neuronal activations and their derivatives are different for the final layer and hidden layers. For the final layer, the value of *output* in equation (3.3) is used as the neuronal activation (a_{LIF}) while considering the discontinuities at spike time instant as noise. Hence, $\frac{\partial E}{\partial output}$ is equal to the final output error, as calculated in equation (3.7).

$$\frac{\partial E}{\partial output} = \frac{\partial}{\partial output}\ \frac{1}{2}(output - label)^2 = output - label = \mathrm{e} \tag{3.7}$$

During back-propagating phase, we consider the leak statistics of membrane potential in the final layer neurons as noise. This allows us to approximate the accumulated membrane potential value for a given neuron as equivalent to the total input current (i.e. *net*) received by the neuron over the forward time duration (T) ($V^L_{mem,\mathrm{j}}(T) \approx \sum_{\mathrm{i}=1}^{n^{L-1}}(w_{\mathrm{ij}}x_{\mathrm{i}}(T)) = net^L_{\mathrm{j}}(T)$). Therefore, the derivative of post-neuronal activation with respect to *net* for final layer ($\frac{\partial output}{\partial net} \equiv \frac{\partial V^L_{mem}(T)/T}{\partial net} = \frac{\partial net^L(T)/T}{\partial net} = \frac{1}{T}$) is calculated as $\frac{1}{T}$ for the final layer.

For the hidden layers, we have post-spike trains as the neuronal outputs. The spike generation function is non-differentiable since it creates a discontinuity (because of step jump) at the time instance of firing. Hence, we introduce a pseudo derivative method for LIF neuronal activation ($a_{LIF}(net)$) for the hidden layers, for back-propagating the output error via the chain rule. The purpose of deriving $a_{LIF}(net)$ is to approximately estimate the $\frac{\partial a_{LIF}}{\partial net}$ term in equation (3.4) for the hidden layers only. To obtain this pseudo derivative of LIF neuronal activation with respect to total input current (i.e., *net*), we make the following approximations. We first estimate the derivative of an Integrate and Fire (IF) neuron's activation. Next, with the derivative of IF neuron's activation, we estimate a leak correctional term to compensate for the leaky effect of membrane potential in LIF activation. Finally, we obtain an approximate derivative for LIF neuronal activation as a combination of two

estimations (i.e., derivative for IF neuron and approximated leak compensation derivative). If a hidden neuron does not fire any spike, the derivative of corresponding neuronal activation is set to zero.

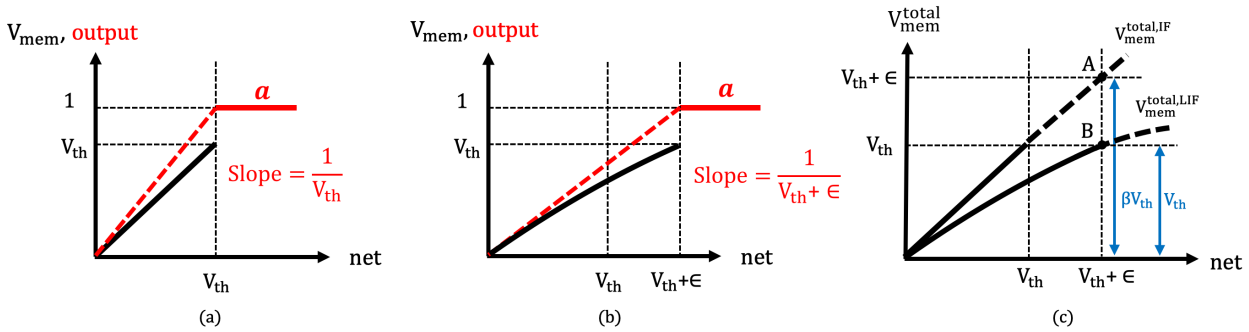


Figure 3.4. (a,b) The illustration of the spike generation function of (a) IF and (b) LIF neuron models, respectively. The x-axis represents the total summation of input currents over time, and y-axis indicates the membrane potential (black) and output (red). The IF neuron generates a post-spike when the input currents accumulated in membrane potential overcome the firing threshold (because of no leaky effect in the membrane potential). However, LIF neuron needs more input currents to cross the firing threshold (because of leaky effect in the membrane potential). Hence, the effective threshold of LIF neurons is considered to be larger compared to the case of IF neurons. (c) The illustration of the estimation of the ratio (β) between the total membrane potential (V_{mem}^{total}) of LIF and IF neurons. If the LIF and IF neuron received the same amount of total input current, the ratio of the total membrane potential of LIF and IF neuron would be estimated as 1:β where β is greater than 1.

The spike generation function of IF neuron is a hard threshold function that generates the output signal as either +1 or 0. The IF neuron fires a post-spike whenever the input currents accumulated in membrane potential exceed the firing threshold (note, in case of IF neuron, there is no leak in the membrane potential). Hence, the membrane potential of a post-neuron at time instant t can be written as,

$$V_{mem}(t) \approx \sum_{\mathrm{i}=1}^{n}(w_{\mathrm{i}}x_{\mathrm{i}}(t)) - V_{th}a_{IF}(t) \tag{3.8}$$

where n denotes the number of pre-neurons, $x_{\mathrm{i}}(t)$ is the sum of spike events from i^{th} pre-neuron over time t (defined in equation (3.1)) and $a_{IF}(t)$ represents the sum of post-spike

trains over time t (defined in equation (3.2)). In equation (3.8), $\sum_{\mathrm{i}=1}^{n}(w_{\mathrm{i}}x_{\mathrm{i}}(t))$ accounts for the integration behavior and $V_{th}a_{IF}(t)$ accounts for the fire/reset behavior of the membrane potential dynamics. If we assume V_{mem} as zero (using small signal approximation), the activation of IF neuron ($a_{IF}(t)$) can be formulated as the equation (3.9). Then, by differentiating it with respect to *net* (in equation (3.10)), the derivative of IF neuronal activation can be approximated as a linear function with slope of $\frac{1}{V_{th}}$ as the straight-through estimation [45].

$$a_{IF}(t) \approx \frac{1}{V_{th}} \sum_{\mathrm{i}=1}^{n}(w_{\mathrm{i}}x_{\mathrm{i}}(t)) = \frac{1}{V_{th}} net(t) \tag{3.9}$$

$$\frac{\partial a_{IF}}{\partial net} \approx \frac{1}{V_{th}} 1 = \frac{1}{V_{th}} \tag{3.10}$$

The spike generation function of both the IF and LIF neuron models are the same, namely the hard threshold function. However, the effective neuronal thresholds are considered to be different for the two cases, as shown in Fig. 3.4a,b. In the LIF neuron model, due to the leaky effect in the membrane potential, larger input current (as compared to IF neuron) needs to be accumulated in order to cross the neuronal threshold and generate a post-spike. Hence, the effective neuronal threshold becomes $V_{th} + \epsilon$ where ϵ is a positive value that reflects the leaky effect of membrane potential dynamics. Now, the derivative of LIF neuronal activation ($\frac{\partial a_{LIF}}{\partial net}$) can be approximated as a hard threshold function (similar to IF and equation (3.10)) and written as $\frac{1}{V_{th}+\epsilon}$. Clearly, the output of a LIF neuron depends on the firing threshold and leaky characteristics (embodied in ϵ) of the membrane potential whereas the output of an IF neuron depends only on the firing threshold. Next, we explain the detailed steps to estimate the ϵ and in turn calculate the derivative of LIF neuronal activation ($\frac{\partial a_{LIF}}{\partial net}$).

To compute ϵ, the ratio (β) between the total membrane potential ($V_{mem}^{total}(t)$) of IF and LIF neurons is estimated at the end of forward propagation time (T) as shown in Fig. 3.4c. Here, $V_{mem}^{total}(t)$ represents the hypothetical total membrane potential with accumulated input current without reset mechanism until time step (t). Suppose both the IF and LIF neurons received the same amount of total input current (i.e. $net(T)$), the total membrane potential of LIF neuron is expected to be lower than the total membrane potential of IF neuron ($V_{mem}^{total,LIF}(T) : V_{mem}^{total,IF}(T) = 1 : \beta$ where $\beta > 1$). Hence, by comparing the total membrane

potential values of IF and LIF neurons in Fig. 3.4c, the relation of ϵ and β can be obtained as follows,

$$V_{th} + \epsilon = \beta V_{th} \tag{3.11}$$

where $V_{th} + \epsilon$ represents the total membrane potential of IF neuron (point A in Fig. 3.4c) and V_{th} indicates the total membrane potential of LIF neuron (point B in Fig. 3.4c) when both neurons received the same amount of *net* inputs. Based on this assumption, we now estimate the ratio (β) by using the relation of the spike output evolution ($\frac{\partial a(t)}{\partial t}$) and the total membrane potential evolution ($\frac{\partial V_{mem}^{total}(t)}{\partial t}$) over time as described in equation (3.13-3.17). As mentioned previously, the total input current (i.e. $net(t)$) and total membrane potential ($V_{mem}^{total}(t)$) are estimated similar to that of IF neuron (because of no leaky effect) so that equation (3.12) can be derived from equation (3.9). By differentiating equation (3.12) with respect to time, we get the relation of the spike output evolution ($\frac{\partial a_{IF}(t)}{\partial t}$) and the membrane potential evolution ($\frac{\partial V_{mem}^{total,IF}(t)}{\partial t}$) over time for IF neuron as described in equation (3.13).

$$a_{IF}(t) \approx \frac{1}{V_{th}} net(t) \approx \frac{1}{V_{th}} V_{mem}^{total,IF}(t) \tag{3.12}$$

$$\frac{\partial a_{IF}(t)}{\partial t} \approx \frac{1}{V_{th}} \frac{\partial V_{mem}^{total,IF}(t)}{\partial t} \tag{3.13}$$

Hence, in IF neuron case, the evolution of membrane potential over time ($\frac{\partial V_{mem}^{total,IF}(t)}{\partial t}$) can be represented by the multiplication of firing threshold (V_{th}) and the spike output evolution ($\frac{\partial a_{IF}(t)}{\partial t}$) in equation (3.14). Note, the evolution of membrane potential over time ($\frac{\partial V_{mem}^{total,IF}(t)}{\partial t}$) indicates the integration component due to the average input current over time. We consider $a_{IF}(t)$ as homogeneous spike trains where spike firing rates are constant, so that the $\frac{\partial a_{IF}(t)}{\partial t}$ can be replaced with the post-neuronal firing rate ($rate(t)$). The homogeneous post-neuronal firing rate, $rate(t)$, can be represented by $\frac{a(t)}{t}$ where $a(t)$ is the number of post-spikes and t means the given forward time window. In LIF neuron case, however, the evolution of membrane potential ($\frac{\partial V_{mem}^{total,LIF}(t)}{\partial t}$) can be expressed as the combination of average input current (integration component) and leaky (exponential decay) effect as shown in equation (3.15). To measure the leaky effect in equation (3.15), we estimate the low-pass filtered

output spikes ($t_k \leq t$) that leak over time using the function $V_{th}f(t)$ (depicted in equation (3.16)), and differentiate it with respect to time at $t \rightarrow t_k^+$ (from the right-sided limit). The $V_{th}f(t)$, as a post-synaptic potential, contains the total membrane potential history over time. The time constant (τ_m) in equation (3.16) determines the decay rate of post-synaptic potential. Essentially, the main idea is to approximately estimate the leaky effect by comparing the total membrane potential and obtain the ratio (β) between both cases (i.e. IF and LIF neurons).

$$\frac{\partial V_{mem}^{total,IF}(t)}{\partial t} \approx V_{th}\frac{\partial a_{IF}(t)}{\partial t} \approx V_{th}rate(t) \tag{3.14}$$

$$\frac{\partial V_{mem}^{total,LIF}(t)}{\partial t} \approx V_{th}rate(t) + V_{th}\frac{\partial f(t)}{\partial t} \tag{3.15}$$

$$f(t) = \sum_k exp(-\frac{t-t_k}{\tau_m}) \tag{3.16}$$

$$\frac{\partial a_{IF}(t)}{\partial t} \approx \frac{1}{V_{th}}\frac{\partial V_{mem}^{total,IF}(t)}{\partial t} = \beta\frac{1}{V_{th}}\frac{\partial V_{mem}^{total,LIF}(t)}{\partial t} \tag{3.17}$$

By solving the equation (3.14-3.17), the inverse ratio ($\frac{1}{\beta}$) is derived as follows in equation (3.18),

$$\frac{1}{\beta} = 1 + \frac{1}{rate(t)}\frac{\partial f(t)}{\partial t} \tag{3.18}$$

where the first term (unity) indicates the effect of average input currents (that is observed from the approximate derivative of IF neuron activation, namely the straight-through estimation) and the second term ($\frac{1}{rate(t)}\frac{\partial f(t)}{\partial t}$) represents the leaky (exponential decay) effect of LIF neuron for the forward propagation time window. Then, by using the relations of ϵ and β in equation (3.11), the derivative of LIF neuronal activation can be obtained as $\frac{\partial a_{LIF}}{\partial net} = \frac{1}{V_{th}+\epsilon} = \frac{1}{\beta V_{th}}$. In this chapter, to avoid the vanishing gradient phenomena during the error back-propagation, the leaky effect term ($\frac{1}{rate(t)}\frac{\partial f(t)}{\partial t}$) is divided by the size of the forward propagation time window (T). Hence, the scaled time derivative of this function, $\frac{1}{\gamma}f(t)$, is used as the leak correctional term where γ denotes the number of output spike events for a particular neuron over the total forward propagation time. As a result, we obtain an approximate derivative for LIF neuronal activation (in hidden layers) as a combination of the straight-through estimation (i.e., approximate derivative of IF neuron activation) and the

leak correctional term that compensates leaky effect in the membrane potential as described in equation (3.19). Please note that, in this work, input and output spikes are not exponentially decaying, the leak only happens according to the mechanism of membrane potential. Moreover, $f(t)$ is not a part of the forward propagation phase, and rather it is only defined to approximately measure the leaky effect during the backward propagation phase by differentiating it with respect to time. The function $f(t)$ is a time-dependent function that simply integrates the output spikes ($t_k \leq t$) temporally, and the resultant sum is decayed over time. It is evident that $f(t)$ is continuous except where spikes occur and the activities jump up [16]. Therefore, $f(t)$ is differentiable at $t \to t_k^+$ (from the right-sided limit). Note that, to capture the leaky effect (exponential decay), it is necessary to compute the derivative of $f(t)$ at the points in between the spiking activities, not at the time instant of spiking.

$$\frac{\partial a_{LIF}}{\partial net} = \frac{1}{V_{th}+\epsilon} = \frac{1}{\beta V_{th}} \approx \frac{1}{V_{th}}(1+\frac{1}{\gamma}f(t)) = \frac{1}{V_{th}}(1+\frac{1}{\gamma}\sum_k -\frac{1}{\tau_m}\mathrm{e}^{-\frac{t-t_k}{\tau_m}}) \tag{3.19}$$

In summary, the approximations applied to implement a spike-based BP algorithm in SNN are as follows:

- During the back-propagating phase, we consider the leaks in the membrane potential of final layer neurons as noises so that the accumulated membrane potential is approximated as equivalent to the total input current ($V_{mem}^L \approx net$). Therefore, the derivative of post-neuronal activation with respect to net ($\frac{\partial output}{\partial net}$) is calculated as $\frac{1}{T}$ for the final layer.
- For hidden layers, we first approximate the activation of an IF neuron as a linear function (i.e., straight-through estimation). Hence, we are able to estimate its derivative of IF neuron's activation [45] with respect to total input current.
- To capture the leaky effect of a LIF neuron (in hidden layers), we estimate the scaled time derivative of the low-pass filtered output spikes that leak over time, using the function $f(t)$. This function is continuous except for the time points where spikes occur [16]. Hence, it is differentiable in the sections between the spiking activities.
- We obtain an approximate derivative for LIF neuronal activation (in hidden layers) as a combination of two derivatives. The first one is the straight-through estimation (i.e.,

approximate derivative of IF neuron activation). The second one is the leak correctional term that compensates the leaky effect in the membrane potential of LIF neurons. The combination of straight-through estimation and the leak correctional term is expected to be less than 1.

Based on these approximations, we can train SNNs with direct spike inputs using a spike-based BP algorithm.

At the final layer, the error gradient, δ^L, represents the gradient of the output loss with respect to total input current (i.e., net) received by the post-neurons. It can be calculated by multiplying the final output error (e) with the derivative of the corresponding post-neuronal activation ($\frac{\partial output}{\partial net^L}$) as shown in equation (3.20). At any hidden layer, the local error gradient, δ^l, is recursively estimated by multiplying the back-propagated gradient from the following layer ($(w^l)^{Tr} * \delta^{l+1}$) with derivative of the neuronal activation, $a_{LIF}(net^l)$, as presented in equation (3.21). Note that element-wise multiplication is indicated by '.' while matrix multiplication is represented by '*' in the respective equations.

$$\delta^L = \frac{\partial E}{\partial output}\frac{\partial output}{\partial net^L} = \mathrm{e}\frac{1}{T} = \frac{\mathrm{e}}{T} \tag{3.20}$$

$$\delta^l = ((w^l)^{Tr} * \delta^{l+1}).a_{LIF}(net^l) \tag{3.21}$$

The derivative of net with respect to weight is simply the total incoming spikes over time as derived in equation (3.22). The derivative of the output loss with respect to the weights interconnecting the layers l and $l+1$ ($\triangle w^l$ in equation (3.23)) is determined by multiplying the transposed error gradient at $l+1$ (δ^{l+1}) with the input spikes from layer l. Finally, the calculated partial derivatives of loss function are used to update the respective weights using a learning rate (η_{BP}) as illustrated in equation (3.24). As a result, iterative updating of the weights over mini-batches of input patterns leads the network state to a local minimum, thereby enabling the network to capture multiple-levels of internal representations of the data.

$$\frac{\partial net}{\partial w^l} = \frac{\partial}{\partial w^l}(w^l * x^l(t)) = x^l(t) \tag{3.22}$$

$$\triangle w^l = \frac{\partial E}{\partial w^l} = x^l(t) * (\delta^{l+1})^{Tr} \tag{3.23}$$

$$w^l_{updated} = w^l - \eta_{BP} \triangle w^l \tag{3.24}$$

Algorithm 1 Forward propagation with dropout at each iteration in SNN

Input : Poisson-distributed input spike train ($inputs$), Dropout ratio (p), Total number of time steps ($\#timesteps$), Membrane potential (V_{mem}), Time constant of membrane potential (τ_m), Firing threshold (V_{th})
Initialize $SNN^l.V_{mem} \leftarrow 0 \; \forall l = 2, ..., \#SNN.layer$
// Define the random subset of units (with a probability $1 - p$) at each iteration
for $l \leftarrow 1$ to $\#SNN.layer - 1$ **do**
    $mask^l \leftarrow generate\_random\_subset(probability = 1 - p)$
for $t \leftarrow 1$ to $\#timesteps$ **do**
    // Set input of first layer equal to spike train of a mini-batch data
    $SNN^1.spike[t] \leftarrow inputs[t]$;
    for $l \leftarrow 2$ to $\#SNN.layer$ **do**
        // Integrate weighted sum of input spikes to membrane potential
        $SNN^l.V_{mem}[t] \leftarrow SNN^l.V_{mem}[t\text{-}1] + SNN^{l-1}forward(SNN^{l-1}.spike[t]). * (mask^{l-1}/(1\text{-}p))$;
        // If V_{mem} is greater than V_{th}, post-neuron generate a spike
        if $SNN^l.V_{mem}[t] > SNN^l.V_{th}$ **then**
            // Membrane potential resets if the corresponding neuron fires a spike
            $SNN^l.spike[t] \leftarrow 1$
            $SNN^l.V_{mem}[t] \leftarrow 0$
        else
            // Else, membrane potential decays over time
            $SNN^l.spike[t] \leftarrow 0$
            $SNN^l.V_{mem}[t] \leftarrow e^{-\frac{1}{\tau_m}} * SNN^l.V_{mem}[t]$

3.3.2 Dropout in Spiking Nerual Network

Dropout [2] is one of the popular regularization techniques while training deep ANNs. This technique randomly disconnects certain units with a given probability (p) to avoid units being overfitted and co-adapted too much to given training data. There are prior works [46]–[48] that investigated the biological insights on how synaptic stochasticity can provide dropout-like functional benefits in SNNs. In this chapter, we employ the concept of dropout technique in order to regularize deep SNNs effectively. Note, dropout technique

is only applied during training and is not used when evaluating the performance of the network during inference. There is a subtle difference in the way dropout is applied in SNNs compared to ANNs. In ANNs, each epoch of training has several iterations of mini-batches. In each iteration, randomly selected units (with dropout ratio of p) are disconnected from the network while weighting by its posterior probability ($\frac{1}{1-p}$). However, in SNNs, each iteration has more than one forward propagation depending on the time length of the spike train. We back-propagate the output error and modify the network parameters only at the last time step. For dropout to be effective in our training method, it has to be ensured that the set of connected units within an iteration of mini-batch data is not changed, such that the neural network is constituted by the same random subset of units during each forward propagation within a single iteration. On the other hand, if the units are randomly connected at each time-step, the effect of dropout will be averaged out over the entire forward propagation time within an iteration. Then, the dropout effect would fade-out once the output error is propagated backward and the parameters are updated at the last time step. Therefore, we need to keep the set of randomly connected units for the entire time window within an iteration. In the experiment, we use the SNN version of dropout technique with the probability (p) of omitting units equal to 0.2-0.25. Note that the activations are much sparser in SNN forward propagations compared to ANNs, hence the optimal p for SNNs needs to be less than a typical ANN dropout ratio (p=0.5). The details of SNN forward propagation with dropout are specified in Algorithm 1.

3.4 Experimental Setup and Result

3.4.1 Experimental Setup

The primary goal of our experiments is to demonstrate the effectiveness of the proposed spike-based BP training methodology in a variety of deep network architectures. We first describe our experimental setup and baselines. For the experiments, we developed a custom simulation framework using the Pytorch deep learning package [49] for evaluating our proposed SNN training algorithm. Our deep convolutional SNNs are populated with biologically plausible LIF neurons (with neuronal firing thresholds of unity) in which a pair of pre- and

post- neurons are interconnected by plastic synapses. At the beginning, the synaptic weights are initialized with Gaussian random distribution of zero-mean and standard deviation of $\sqrt{\frac{\kappa}{n^l}}$ (n^l: number of fan-in synapses) as introduced in [50]. Note, the initialization constant κ differs by the type of network architecture. For instance, we have used $\kappa = 2$ for non-residual network and $\kappa = 1$ for residual network. For training, the synaptic weights are trained with a mini-batch spike-based BP algorithm in an end-to-end manner, as explained in section 3.3.1. For static datasets, we train our network models for 150 epochs using mini-batch stochastic gradient descent BP that reduces its learning rate at 70^{th}, 100^{th} and 125^{th} training epochs. For the neuromorphic dataset, we use Adam [51] learning method and reduce its learning rate at 40^{th}, 80^{th} and 120^{th} training epochs. Please, refer to Table 3.1 for more implementation details. The datasets and network topologies used for benchmarking, the input spike generation scheme for event-based operation and determination of the number of time-steps required for training and inference are described in the following sub-sections.

Table 3.1. Parameters used in the experiments

Parameter	**Value**
Time Constant of Membrane Potential (τ_m)	100 time-steps
BP Training Time Duration	50-100 time-steps
Inference Time Duration	Same as training
Mini-batch Size	16-32
Spatial-pooling Non-overlapping Region/Stride	2×2, 2
Neuronal Firing Threshold	1 (hidden layer), ∞ (final layer)
Weight Initialization Constant (κ)	2 (non-residual network), 1 (residual network)
Learning rate (η_{BP})	0.002 - 0.003
Dropout Ratio (p)	0.2 - 0.25

Benchmarking Datasets

We demonstrate the efficacy of our proposed training methodology for deep convolutional SNNs on three standard vision datasets and one neuromorphic vision dataset, namely the MNIST [1], SVHN [52], CIFAR-10 [53] and N-MNIST [54]. The MNIST dataset is composed of gray-scale (one-dimensional) images of handwritten digits whose sizes are 28 by 28. The SVHN and CIFAR-10 datasets are composed of color (three-dimensional) images whose sizes

are 32 by 32. The N-MNIST dataset is a neuromorphic (spiking) dataset that is converted from static MNIST dataset using Dynamic Vision Sensor (DVS) [55]. The N-MNIST dataset contains two-dimensional images that include ON and OFF event stream data whose sizes are 34 by 34. The ON (OFF) event represents the increase (decrease) in pixel bright changes. The details of the benchmark datasets are listed in Table 3.2. For evaluation, we report the top-1 classification accuracy by classifying the test samples (training samples and test samples are mutually exclusive).

Table 3.2. Benchmark Datasets

Dataset	**Image**	**#Training Samples**	**#Testing Samples**	**#Category**
MNIST	28×28, gray	60,000	10,000	10
SVHN	32×32, color	73,000	26,000	10
CIFAR-10	32×32, color	50,000	10,000	10
N-MNIST	$34 \times 34 \times 2$, ON and OFF spikes	60,000	10,000	10

Network Topologies

We use various SNN architectures depending on the complexity of the benchmark datasets. For MNIST and N-MNIST datasets, we used a network consisting of two sets of alternating convolutional and spatial-pooling layers followed by two fully-connected layers. This network architecture is derived from LeNet5 model [1]. Note that Table 3.3 summarizes the layer type, kernel size, the number of output feature maps and stride of SNN model for MNIST dataset. The kernel size shown in the table is for 3-D convolution where the 1^{st} dimension is for number of input feature-maps and 2^{nd}-3^{rd} dimensions are for convolutional kernels. For SVHN and CIFAR-10 datasets, we used deeper network models consisting of 7 to 11 trainable layers including convolutional, spatial-pooling and fully-connected layers. In particular, these networks consisting of beyond 5 trainable layers are constructed using small (3×3) convolutional kernels. We term the deep convolutional SNN architecture that includes 3×3 convolutional kernel [36] without residual connections as 'VGG SNN' and with skip (residual) connections [5] as 'Residual SNN'. In Residual SNNs, some convolutional layers convolve kernel with the stride of 2 in both x and y directions, to incorporate the functionality of spatial-pooling layers. Please, refer to Table 3.3 and Table 3.4 that summarize the details

of deep convolutional SNN architectures. In the results section, we will discuss the benefit of deep SNNs in terms of classification performance as well as inference speedup and energy efficiency.

Table 3.3. The deep convolutional spiking neural network architectures for MNIST, N-MNIST and SVHN datasets

4 layer network				VGG7				ResNet7			
Layer type	**Kernel size**	**#o/p feature-maps**	**Stride**	**Layer type**	**Kernel size**	**#o/p feature-maps**	**Stride**	**Layer type**	**Kernel size**	**#o/p feature-maps**	**Stride**
Convolution	1×5×5	20	1	Convolution	3×3×3	64	1	Convolution	3×3×3	64	1
Average-pooling	2×2		2	Convolution	64×3×3	64	2	Average-pooling	2×2		2
				Average-pooling	2×2		2				
Convolution	20×5×5	50	1	Convolution	64×3×3	128	1	Convolution	64×3×3	128	1
Average-pooling	2×2		2	Convolution	128×3×3	128	2	Convolution	128×3×3	128	2
				Convolution	128×3×3	128	2	Skip convolution	64×1×1	128	2
				Average-pooling	2×2		2				
								Convolution	128×3×3	256	1
								Convolution	256×3×3	256	2
								Skip convolution	128×1×1	256	2
Fully-connected		200		Fully-connected		1024		Fully-connected		1024	
Output		10		Output		10		Output		10	

Table 3.4. The deep convolutional spiking neural network architectures for a CIFAR-10 dataset

VGG9				ResNet9				ResNet11			
Layer type	**Kernel size**	**#o/p feature-maps**	**Stride**	**Layer type**	**Kernel size**	**#o/p feature-maps**	**Stride**	**Layer type**	**Kernel size**	**#o/p feature-maps**	**Stride**
Convolution	3×3×3	64	1	Convolution	3×3×3	64	1	Convolution	3×3×3	64	1
Convolution	64×3×3	64	1	Average-pooling	2×2		2	Average-pooling	2×2		2
Average-pooling	2×2		2								
Convolution	64×3×3	128	1	Convolution	64×3×3	128	1	Convolution	64×3×3	128	1
Convolution	128×3×3	128	1	Convolution	128×3×3	128	1	Convolution	128×3×3	128	1
Average-pooling	2×2		2	Skip convolution	64×1×1	128	1	Skip convolution	64×1×1	128	1
Convolution	128×3×3	256	1	Convolution	128×3×3	256	1	Convolution	128×3×3	256	1
Convolution	256×3×3	256	1	Convolution	256×3×3	256	2	Convolution	256×3×3	256	2
Convolution	256×3×3	256	1	Skip connection	128×1×1	256	2	Skip convolution	128×1×1	256	2
Average-pooling	2×2		2								
				Convolution	256×3×3	512	1	Convolution	256×3×3	512	1
				Convolution	512×3×3	512	2	Convolution	512×3×3	512	1
				Skip convolution	256×1×1	512	2	Skip convolution	512×1×1	512	1
								Convolution	512×3×3	512	1
								Convolution	512×3×3	512	2
								Skip convolution	512×1×1	512	2
Fully-connected		1024		Fully-connected		1024		Fully-connected		1024	
Output		10		Output		10		Output		10	

ANN-SNN Conversion Scheme

As mentioned previously, off-the-shelf trained ANNs can be successfully converted to SNNs by replacing ANN neurons (ReLU) with Integrate and Fire (IF) spiking neurons and adjusting the neuronal thresholds with respect to synaptic weights. In the literature, several methods have been proposed [11], [12], [37], [38], [56] for balancing appropriate ratios between neuronal thresholds and synaptic weights of spiking neuron in the case of ANN-SNN conversion. In this chapter, we compare various aspects of our direct-spike trained models with two prior ANN-SNN conversion works [11], [38], which proposed near-lossless ANN-SNN conversion schemes for deep network architectures. The first scheme [11] balanced the

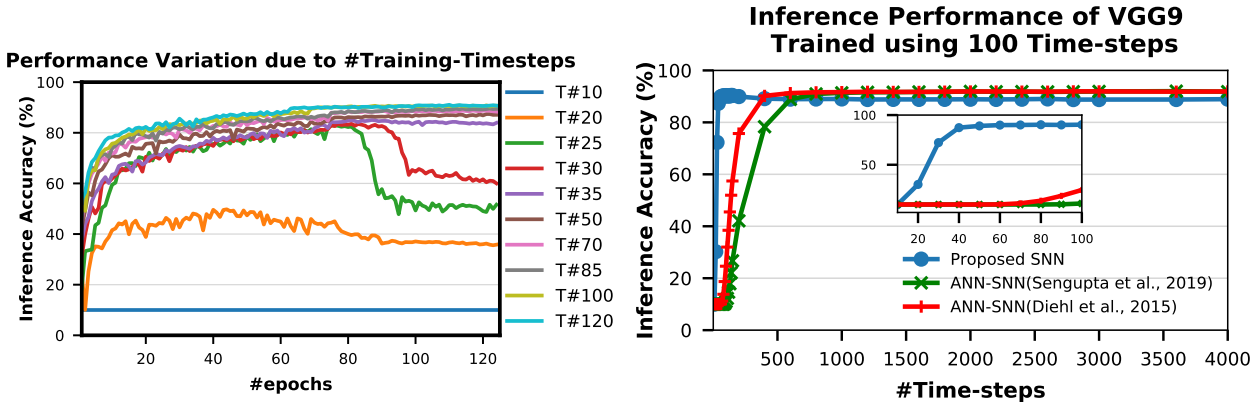
neuronal firing thresholds with respect to corresponding synaptic weights layer-by-layer depending on the actual spiking activities of each layer using a subset of training samples. The second scheme [38] balanced the neuronal firing thresholds with the consideration of ReLU activations in the corresponding ANN layer. Basically, we compare our direct-spike trained model with converted SNNs on the same network architecture in terms of accuracy, inference speed and energy-efficiency. Please note that there are a couple of differences on the network architecture between the conversion networks [11], [38] and our scheme. First, the conversion networks always use average-pooling to reduce the size of previous convolutional output feature-map, whereas our models interchangeably use average pooling or convolve kernels with a stride of 2 in the convolutional layer. Next, the conversion networks only consider identity skip connections for residual SNNs. However, we implement skip connections using either identity mapping or 1×1 convolutional kernel.

Spike Generation Scheme

For the static vision datasets (MNIST, SVHN and CIFAR-10), each input pixel intensity is converted to a stream of Poisson-distributed spike events that have equivalent firing rates. Specifically, at each time step, the pixel intensity is compared with a uniformly distributed random number (in the range between 0 and 1). If pixel intensity is greater than the random number at the corresponding time step, a spike is generated. This rate-based spike encoding is used to feed the input spikes to the network for a given period of time during both training and inference. For color image datasets, we use the pre-processing technique of horizontal flip before generating input spikes. These input pixels are normalized to represent zero mean and unit standard deviation. Thereafter, we scale the pixel intensities to bound them in the range [-1,1] to represent the whole spectrum of input pixel representations. The normalized pixel intensities are converted to Poisson-distributed spike events such that the generated input signals are bipolar spikes. For the neuromorphic version of the dataset (N-MNIST), we use the original (unfiltered and uncentered) version of spike streams to directly train and test the network in the time domain.

Time-steps

As mentioned in section 3.4.1, we generate a stochastic Poisson-distributed spike train for each input pixel intensity for event-based operation. The duration of the spike train is very important for SNNs. We measure the length of the spike train (spike time window) in time-steps. For example, a 100 time-step spike train will have approximately 50 random spikes if the corresponding pixel intensity is half in a range of [0,1]. If the number of time-steps (spike time window) is too less, then the SNN will not receive enough information for training or inference. On the other hand, a large number of time-steps will destroy the stochastic property of SNNs and get rid of noise and imprecision at the cost of high latency and power consumption. Hence, the network will not have much energy efficiency over ANN implementations. For these reasons, we experimented with the different number of time-steps to empirically obtain the optimal number of time-steps required for both training and inference. The experimental process and results are explained in the following subsections.



(a) (b)

Figure 3.5. Inference performance variation due to (a) #Training-Timesteps and (b) #Inference-Timesteps. *T#* in (a) indicates number of time-steps used for training. Fig. (a) shows that inference accuracy starts to saturate as #training-timesteps increase. In Fig. (b), the zoomed version on the inset shows that the SNN trained with the proposed scheme performs very well even with only 30 time-steps while the peak performance occurs around 100 time-steps.

Optimal #time-steps for Training

A spike event can only represent 0 or 1 in each time step, therefore usually its bit precision is considered 1. However, the spike train provides temporal data, which is an additional source of information. Therefore, the spike train length (number of time-steps) in SNN can be considered as its actual precision of neuronal activation. To obtain the optimal #time-steps required for our proposed training method, we trained VGG9 networks on CIFAR-10 dataset using different time-steps ranging from 10 to 120 (shown in Fig. 3.5(a)). We found that for only 10 time-steps, the network is unable to learn anything as there is not enough information (input precision too low) for the network to be able to learn. This phenomenon is explained by the lack of spikes in the final output. With the initial weights, the accumulated sum of the LIF neuron is not enough to generate output spikes in the later layers. Hence, none of the input spikes propagates to the final output neurons and the output distributions remain 0. Therefore, the computed gradients are always 0 and the network is not updated. For 35-50 time-steps, the network learns well and converges to a reasonable point. From 70 time-steps, the network accuracy starts to saturate. At about 100 time-steps, the network training improvement completely saturates. This is consistent with the bit precision of the inputs. It has been shown in [57] that 8 bit inputs and activations are sufficient to achieve optimal network performance for standard image recognition tasks. Ideally, we need 128 time-steps to represent 8 bit inputs using bipolar spikes. However, 100 time-steps proved to be sufficient as more time-steps provide marginal improvement. We observe a similar trend in VGG7, ResNet7, ResNet9 and ResNet11 SNNs as well while training for SVHN and CIFAR-10 datasets. Therefore, we considered 100 time-steps as the optimal #time-steps for training in our proposed methodology. Moreover, for MNIST dataset, we used 50 time-steps since the required bit precision is only 4 bits [57].

Optimal #time-steps for Inference

To obtain the optimal #time-steps required for inferring an image utilizing a network trained with our proposed method, we conducted similar experiments as described in section 3.4.1. We first trained a VGG9 network for CIFAR-10 dataset using 100 time-steps (optimal

according to experiments in section 3.4.1). Then, we tested the network performances with different time-steps ranging from 10 to 4000 (shown in Fig. 3.5(b)). We observed that the network performs very well even with only 30 time-steps while the peak performance occurs around 100 time-steps. For more than 100 time-steps, the accuracy degrades slightly from the peak. This behavior is very different from ANN-SNN converted networks where the accuracy keeps on improving as #time-steps is increased (shown in Fig. 3.5(b)). This can be attributed to the fact that our proposed spike-based training method incorporates the temporal information well into the network training procedure so that the trained network is tailored to perform best at a specific spike time window for the inference. On the other hand, the ANN-SNN conversion schemes are unable to incorporate the temporal information of the input in the trained network. Hence, the ANN-SNN conversion schemes require much higher #time-steps (compared to SNN trained using the proposed method) for the inference in order to resemble input-output mappings similar to ANNs.

3.4.2 Results

In this section, we analyze the classification performance and efficiency achieved by the proposed spike-based training methodology for deep convolutional SNNs compared to the performance of the transformed SNN using ANN-SNN conversion scheme.

The Classification Performance

Most of the classification performances available in the literature for SNNs are for MNIST and CIFAR-10 datasets. The popular methods for SNN training are 'Spike Time Dependent Plasticity (STDP)' based unsupervised learning [13], [15], [58], [59] and 'spike-based back-propagation' based supervised learning [16], [60]–[63]. There are a few works [17], [64]–[66] which tried to combine the two approaches to get the best of both worlds. However, these training methods were able to neither train deep SNNs nor achieve good inference performance compared to ANN implementations. Hence, ANN-SNN conversion schemes have been explored by researchers [11], [12], [37], [38], [56]. Till date, ANN-SNN conversion schemes achieved the best inference performance for CIFAR-10 dataset using deep networks [11], [12].

Classification performances of all these works are listed in Table 3.5 along with ours. To the best of our knowledge, we achieved the best inference accuracy for MNIST using LeNet structured network compared to our spike based training approaches. We also achieved accuracy performance comparable with ANN-SNN converted network [11], [43] for CIFAR-10 dataset while beating all other spike-based training methods.

Table 3.5. Comparison of the SNNs classification accuracies on MNIST, N-MNIST and CIFAR-10 datasets.

Model	**Learning Method**	**Accuracy (MNIST)**	**Accuracy (N-MNIST)**	**Accuracy (CIFAR-10)**
Hunsberger et al.[56]	Offline learning, conversion	98.37%	–	82.95%
Esser et al.[67]	Offline learning, conversion	–	–	89.32%
Diehl et al.[38]	Offline learning, conversion	99.10%	–	–
Rueckauer et al.[12]	Offline learning, conversion	99.44%	–	88.82%
Sengupta et al.[11]	Offline learning, conversion	–	–	91.55%
Kheradpisheh et al.[17]	Layerwise STDP + offline SVM classifier	98.40%	–	–
Panda et al.[19]	Spike-based autoencoder	99.08%	–	70.16%
Lee et al.[16]	Spike-based BP	99.31%	98.74%	–
Wu et al.[61]	Spike-based BP	99.42%	98.78%	50.70%
Lee et al.[66]	STDP-based pretraining + spike-based BP	99.28%	–	–
Jin et al.[60]	Spike-based BP	99.49%	98.88%	–
Wu et al.[68]	Spike-based BP	–	99.53%	90.53%
This work	Spike-based BP	99.59%	99.09%	90.95%

For a more extensive comparison, we compare the inference performances of trained networks using our proposed methodology with the ANNs and ANN-SNN conversion scheme for same network configuration (depth and structure) side by side in Table 3.6. We also compare with the previous best SNN training results found in the literature that may or may not have the same network depth and structure as ours. The ANN-SNN conversion scheme is a modified and improved version of [11]. We are using this modified scheme since it achieves better conversion performance than [11] as explained in section 3.4.1. Note that all reported classification accuracies are the average of the maximum inference accuracies for 3 independent runs with different seeds.

After initializing the weights, we train the SNNs using a spike-based BP algorithm in an end-to-end manner with Poisson-distributed spike train inputs. Our evaluation of MNIST dataset yields a classification accuracy of 99.59%, which is the best compared to any other SNN training scheme and also identical to other ANN-SNN conversion schemes. We achieve

∼96% inference accuracy on SVHN dataset for both trained non-residual and residual SNN. Inference performance for SNNs trained on SVHN dataset has not been reported previously in the literature. We implemented three different networks, as shown in Table 3.4, for classifying CIFAR-10 dataset using a proposed spike-based BP algorithm. For the VGG9 network, the ANN-SNN conversion schemes provides a near lossless converted network compared to baseline ANN implementation while our proposed training method yields a classification accuracy of 90.45%. For ResNet9 network, the ANN-SNN conversion schemes provide inference accuracy within 0.5-1% of baseline ANN implementation. However, our proposed spike-based training method achieves inference accuracy that is within ∼1.5% of baseline ANN implementation. In the case of ResNet11, we observe that the inference accuracy improvement is marginal compared to ResNet9 for baseline ANN implementation and ANN-SNN conversion schemes. However, our proposed SNN training shows improvement of ∼0.5% for ResNet11 compared to ResNet9. Overall, our proposed training method achieves comparable inference accuracies for both ResNet and VGG networks compared to baseline ANN implementation and ANN-SNN conversion schemes.

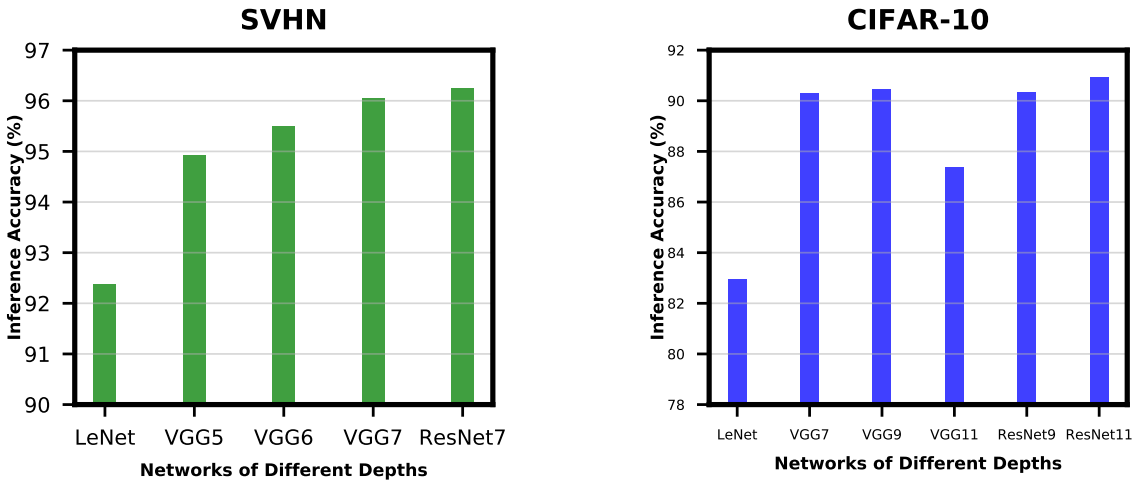
Table 3.6. Comparison of classification performance

Inference Accuracy						
Dataset	**Model**	**ANN**	**ANN-SNN (Diehl et al. [43])**	**ANN-SNN (Sengupta et al. [11])**	**SNN (Previous Best)**	**SNN (This Work)**
MNIST	LeNet	99.57%	99.55%	99.59%	99.49%[60]	99.59%
N-MNIST	LeNet	–	–	–	99.53%[68]	99.09%
SVHN	VGG7	96.36%	96.33%	96.30%	–	96.06%
	ResNet7	96.43%	96.33%	96.40%	–	96.21%
CIFAR-10	VGG9	91.98%	91.89%	92.01%	90.53%[68]	90.45%
	ResNet9	91.85%	90.78%	91.59%		90.35%
	ResNet11	91.87%	90.98%	91.65%		90.95%

Accuracy Improvement with Network Depth

In order to analyze the effect of network depth for SNNs, we experimented with networks of different depths while training for SVHN and CIFAR-10 datasets. For SVHN dataset, we started with a shallow network derived from LeNet5 model [1] with 2 convolutional and 2 fully-connected layers. This network was able to achieve inference accuracy of only 92.38%. Then, we increased the network depth by adding 1 convolutional layer before the

2 fully-connected layers and we termed this network as VGG5. VGG5 network was able to achieve significant improvement over its predecessor. Similarly, we tried VGG6 followed by VGG7, and the improvement started to become very small. We have also trained ResNet7 to understand how residual networks perform compared to non-residual networks of similar depth. The results of these experiments are shown in Fig. 3.6(a). We carried out similar experiments for CIFAR-10 dataset as well. The results show a similar trend as described in Fig. 3.6(b). These results ensure that network depth improves the learning capacity of direct-spike trained SNNs similar to ANNs. The non-residual networks saturate at a certain depth and start to degrade if network depth is further increased (VGG11 in Fig. 3.6(b)) due to the degradation problem mentioned in [5]. In such a scenario, the residual connections in deep residual ANNs allow the network to maintain peak classification accuracy utilizing the skip connections [5], as seen in Fig. 3.6(b) (ResNet9 and ResNet11).



(a) (b)

Figure 3.6. Accuracy improvement with network depth for (a) SVHN and (b) CIFAR-10 dataset. In Fig. (a), inference accuracy improves with an increase in network depth. In Fig. (b), the non-residual networks saturate at a certain depth and start to degrade if network depth increases further. However, the residual blocks in deep residual ANNs allow the network to maintain peak classification accuracy (ResNet9 and ResNet11).

3.5 Discussion

3.5.1 Comparison with Relevant Works

In this section, we compare our proposed supervised learning algorithm with other recent spike-based BP algorithms. The spike-based learning rules primarily focus on directly training and testing SNNs with spike-trains, and no conversion is necessary for applying in real-world spiking scenario. In recent years, there are an increasing number of supervised gradient descent method in spike-based learning. The [19] developed a spike-based auto-encoder mechanism to train deep convolutional SNNs. They dealt with membrane potential as a differentiable signal and showed recognition capabilities in standard vision tasks (MNIST and CIFAR-10 datasets). Meanwhile, [16] followed the approach using differentiable membrane potential to explore a spike-based BP algorithm in an end-to-end manner. In addition, [16] presented the error normalization scheme to prevent exploding gradient phenomena while training deep SNNs. Researchers in [60] proposed hybrid macro/micro level backpropagation (HM2-BP). HM2-BP is developed to capture the temporal effect of the individual spike (in micro-level) and rate-encoded error (at macro-level). The reference [69] employed exponential function for the approximate derivative of neuronal function and developed a credit assignment scheme to calculate the temporal dependencies of error throughout the layers. [70] has trained recurrent spiking networks by replacing the threshold with a gate function and employing BPTT technique [71]. While BPTT technique has been a popular method to train recurrent analog and spiking recurrent networks, [72] points out the storing and retrieving past variables and differentiation them through time in biological neurons seems to be impossible. Recently, e-prop [73] presented an approximation method to bypass neuronal state savings for enhancing the computational efficiency of BPTT. In temporal spike encoding domain, [63] proposed an interesting temporal spike-based BP algorithm by treating the spike-time as the differential activation of neuron. Temporal encoding based SNN has the potential to process the tasks with the small number of spikes. All of these works demonstrated spike-based learning in simple network architectures and has a large gap in classification accuracy compared to deep ANNs. More recently, [68] presented a neuron normalization technique (called NeuNorm) that calculates the average input firing rates to

adjust neuron selectivity. NeuNorm enables spike-based training within a relatively short time-window while achieving competitive performances. In addition, they presented an input encoding scheme that receives both spike and non-spike signals for preserving the precision of input data.

There are several points that distinguish this work from others. First, we use a pseudo derivative method that accounts for leaky effect in membrane potential of LIF neurons. We approximately estimate the leaky effect by comparing total membrane potential value and obtain the ratio between IF and LIF neurons. During the back-propagating phase, the pseudo derivative of LIF neuronal function is estimated by combining the straight through estimation and leak correctional term as described in equation (3.19). Next, we construct our networks by leveraging frequently used architectures such as VGG [36] and ResNet [5]. To the best of our knowledge, this is the first work that demonstrates spike-based supervised BP learning for SNNs containing more than 10 trainable layers. Our deep SNNs obtain the superior classification accuracies in MNIST, SVHN and CIFAR-10 datasets in comparison to the other networks trained with the spike-based algorithm. In addition, as opposed to complex error or neuron normalization method adopted by [16] and [68], respectively, we demonstrate that deep SNNs can be naturally trained by only accounting for spiking activities of the network. As a result, our work paves the effective way for training deep SNNs with a spike-based BP algorithm.

3.5.2 Spike Activity Analysis

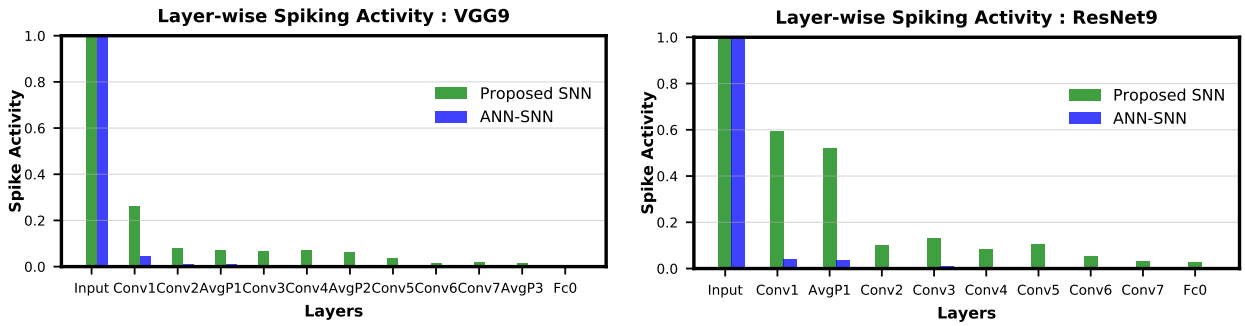
The most important advantage of event-based operation of neural networks is that the events are very sparse in nature. To verify this claim, we analyzed the spiking activities of the direct-spike trained SNNs and ANN-SNN converted networks in the following subsections.

Spike Activity per Layer

The layer-wise spike activities of both SNN trained using our proposed methodology, and ANN-SNN converted network (using scheme 1) for VGG9 and ResNet9 are shown in Fig. 3.7(a) and Fig. 3.7(b), respectively. In the case of ResNet9, only the first average

pooling layer's output spike activity is shown in the figure as for the direct-spike trained SNN, the other spatial-poolings are done by stride 2 convolutions. In Fig. 3.7, it can be seen that the input layer has the highest spike activity that is significantly higher than any other layer. The spike activity reduces significantly as the network depth increases.

We can observe from Fig. 3.7(a) and Fig. 3.7(b) that the average spike activity in a direct-spike trained SNN is much higher than ANN-SNN converted network. The ANN-SNN converted network uses a higher threshold compared to 1 (in case of direct-spike trained SNN) since the conversion scheme applies layer-wise neuronal threshold modulation. This higher threshold reduces spike activity in ANN-SNN converted networks. However, in both cases, the spike activity decreases with increasing network depth.



(a) (b)

Figure 3.7. Layer-wise spike activity in direct-spike trained SNN and ANN-SNN converted network for CIFAR-10 dataset: (a) VGG9 (b) ResNet9 network. The spike activity is normalized with respect to the input layer spike activity, which is the same for both networks. The spike activity reduces significantly for both SNN and ANN-SNN converted network towards the later layers. We have used scheme 1 for ANN-SNN conversion.

#Spikes/Inference

From Fig. 3.7, it is evident that average spike activity in ANN-SNN converted networks is much less than in the direct-spike trained SNN. However, for inference, the network has to be evaluated over many time-steps. Therefore, to quantify the actual spike activity for an inference operation, we measured the average number of spikes required for inferring one image. For this purpose, we counted the number of spikes generated (including input spikes)

Table 3.7. #Spikes/Image inference and spike efficiency comparison between SNN and ANN-SNN converted networks for benchmark datasets trained on different network models. (For each network, the 1st row corresponds to iso-accuracy and the 2nd row corresponds to maximum-accuracy condition.)

Dataset	Model	Spike/Image			Spike Efficiency Compared to	
		SNN	ANN-SNN [43]	ANN-SNN [11]	ANN-SNN [43]	ANN-SNN [11]
MNIST	LeNet	5.52E+04	3.4E+04	2.9E+04	0.62x	0.53x
				7.3E+04		1.32x
SVHN	VGG7	5.56E+06	3.7E+06	1.0E+07	0.67x	1.84x
			1.9E+07	1.7E+07	3.40x	2.99x
	ResNet7	4.66E+06	3.9E+06	3.1E+06	0.85x	0.67x
			2.4E+07	2.0E+07	5.19x	4.30x
CIFAR-10	VGG9	1.24E+06	1.6E+06	2.2E+06	1.32x	1.80x
			8.3E+06	9.6E+06	6.68x	7.78x
	ResNet9	4.32E+06	2.7E+06	1.5E+06	0.63x	0.35x
			1.0E+07	7.8E+06	2.39x	1.80x
	ResNet11	1.53E+06	9.7E+06	1.8E+06	6.33x	1.17x
				9.2E+06		5.99x

for classifying the test set of a particular dataset for a specific number of time-steps and averaged the count for generating the quantity *'#spikes per image inference'*. We have used two different time-steps for ANN-SNN converted networks; one for iso-accuracy comparison and the other for maximum accuracy comparison with the direct-spike trained SNNs. Iso-accuracy inference requires less #time-steps than maximum accuracy inference, hence has a lower number of spikes per image inference. For few networks, the ANN-SNN conversion scheme always provides accuracy less than or equal to the direct-spike trained SNN. Hence, we only compare spikes per image inference in maximum accuracy condition for those ANN-SNN converted networks while comparing with direct-spike trained SNNs. For the analysis, we quantify the spike-efficiency (amount reduction in #spikes) from the #spikes/image inference. The results are listed in Table 3.7, where the 1st row corresponds to iso-accuracy and the 2nd row corresponds to maximum-accuracy condition for each network. As shown in Table 3.7, the direct-spike trained SNNs are more efficient in terms of #spikes/inference compared to the ANN-SNN converted networks for the maximum accuracy condition. For an iso-accuracy condition, only deep SNNs (such as VGG9 and ResNet11) are more efficient in terms of #spikes/inference compared to the ANN-SNN converted networks.

Fig. 3.8 shows the relationship between inference accuracy, latency and #spikes/inference for ResNet11 networks trained on CIFAR-10 dataset. We can observe that #spikes/in-

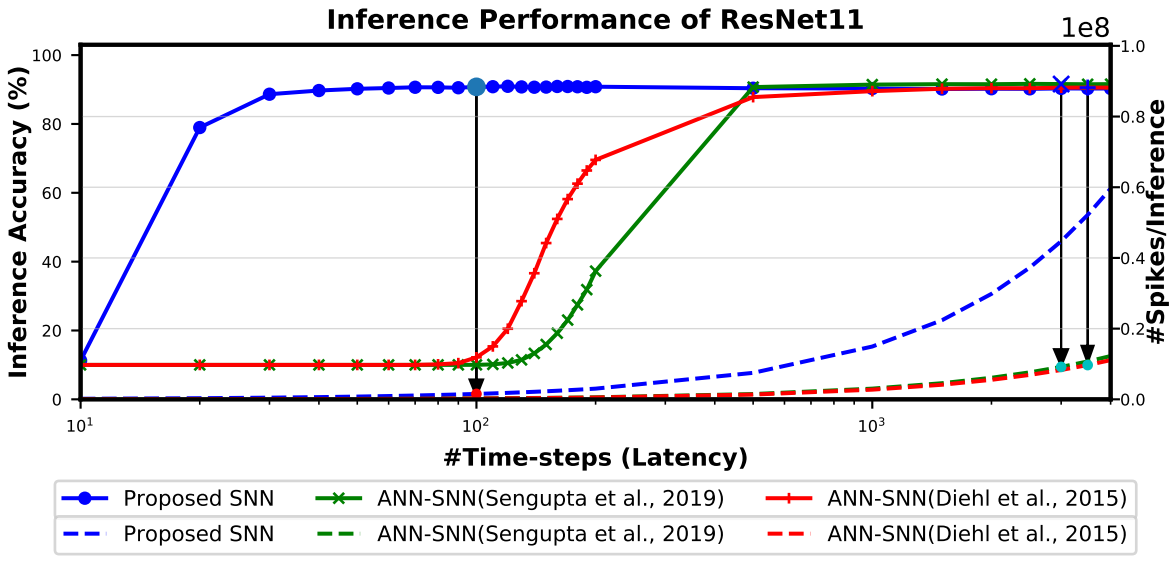


Figure 3.8. The comparison of 'accuracy vs latency vs #spikes/inference' for ResNet11 architecture. In this figure, the solid lines are representing inference accuracy while the dashed lines are representing #spikes/inference. The slope of #spikes/inference curve of the proposed SNN is larger than ANN-SNN converted networks. However, since proposed SNN requires much less time-steps for inference, the number of spikes required for one image inference is significantly lower compared to ANN-SNN. The required #time-steps and corresponding #spikes/inference are shown using highlighted points connected by arrows. Log scale is used for x-axis for easier viewing of the accuracy changes for lower number of time-steps.

ference is higher for direct-spike trained SNN compared to ANN-SNN converted networks at any particular latency. However, SNN trained with spike-based BP requires only 100 time-steps for maximum inference accuracy, whereas ANN-SNN converted networks require 3000-3500 time-steps to reach maximum inference accuracy. Hence, under maximum accuracy condition, direct-spike trained ResNet11 requires much fewer #spikes/inference compared to ANN-SNN converted networks, while achieving comparable accuracy. Even under iso-accuracy condition, the direct-spike trained ResNet11 requires fewer #spikes/inference compared to the ANN-SNN converted networks (Table 3.7).

3.5.3 Inference Speedup

The time required for inference is linearly proportional to the #time-steps (Fig. 3.8). Hence, we can also quantify the inference speedup for direct-spike trained SNNs compared to ANN-SNN converted networks from the #time-steps required for inference, as shown in Table 3.8. For example, for VGG9 network, the proposed training method can achieve 8× (5×) speedup for iso-accuracy and up to 36× (25×) speedup for maximum accuracy in inference compared to respective ANN-SNN converted networks (i.e., scheme 1 [11] and scheme 2 [38]). Similarly, for ResNet networks, the proposed training method can achieve 6× speedup for iso-accuracy and up to 35× speedup for maximum accuracy condition in inference. It is interesting to note that direct-spike trained SNN is always more efficient in terms of time-steps compared to the equivalent ANN-SNN conversion network, but not in terms of the number of spikes, in some cases. It will require a detailed investigation to determine if ANN-SNN methods used higher firing rates, whether they would be able to classify quickly as well, while incurring a lower number of spike/inference.

Table 3.8. Inference #time-steps and corresponding speedup comparison between SNN and ANN-SNN converted networks for benchmark datasets trained on different network models. (For each network, the 1st row corresponds to iso-accuracy and the 2nd row corresponds to maximum-accuracy condition.)

Dataset	**Model**	**Timesteps**			**SNN Inference Speedup Compared to**	
		SNN	**ANN-SNN** [43]	**ANN-SNN** [11]	**ANN-SNN** [43]	**ANN-SNN** [11]
MNIST	LeNet	50	180	200	3.6x	4x
				500		10x
SVHN	VGG7	100	500	1600	5x	16x
			2500	2600	25x	26x
	ResNet7	100	500	400	5x	4x
			3000	2500	30x	25x
CIFAR-10	VGG9	100	500	800	5x	8x
			2500	3600	25x	36x
	ResNet9	100	800	600	8x	6x
			3000	3000	30x	30x
	ResNet11	100	3500	600	35x	6x
				3000		30x

3.5.4 Complexity Reduction

Deep ANNs struggle to meet the demand of extraordinary computational requirements. SNNs can mitigate this effort by enabling efficient event-based computations. To compare the computational complexity of these two cases, we first need to understand the operation principle of both. An ANN operation for inferring the category of a particular input requires a single feed-forward pass per image. For the same task, the network must be evaluated over a number of time-steps in the spiking domain. If regular hardware is used for both ANN and SNN, then it is evident that SNN will have computation complexity in the order of hundreds or thousands more compared to an ANN. However, there are specialized hardware that account for the event-based neural operation and 'computes only when required' for inference. SNNs can potentially exploit such alternative mechanisms of network operation and carry out an inference operation in the spiking domain much more efficiently than an ANN. Also, for deep SNNs, we have observed the increase in sparsity as the network depth increases. Hence, the benefits from event-based neuromorphic hardware are expected to increase as the network depth increases.

An estimate of the actual energy consumption of SNNs and comparison with ANNs is outside the scope of this work. However, we can gain some insight by quantifying the computational energy consumption for a synaptic operation and comparing the number of synaptic operations being performed in the ANN versus the SNN trained with our proposed algorithm and ANN-SNN converted network. We can estimate the number of synaptic operations per layer of a neural network from the structure for the convolutional and linear layers. In an ANN, a multiply-accumulate (MAC) computation is performed per synaptic operation. While a specialized SNN hardware would perform simply an accumulate computation (AC) per synaptic operation only if an incoming spike is received. Hence, the total number of AC operations in a SNN can be estimated by the layer-wise product and summation of the average neural spike count for a particular layer and the corresponding number of synaptic connections. We also have to multiply the #time-steps with the #AC operations to get total #AC operation for one inference. For example, assume that there are L layers each with N_l neurons, S_l synaptic connections and a_l average spiking activity where l is the layer number.

Then, the total number of synaptic operations in a layer is $N_l \times S_l \times a_l$. The $N_l \times S_l$ is equal to the ANN (#MAC) operations of a particular layer. Therefore, the total number of synaptic operations in a layer of an SNN becomes $\#MAC_l \times a_l$. The total number of AC operations required for an image inference is the sum of synaptic operations in all layers during the inference time-window. Hence, $\#AC/inference=(\sum_l(\#MAC_l \times a_l)) \times \#timesteps$. This formula is used for estimating both ANN-SNN AC operations and SNN AC operations per image inference. On the other hand, the number of ANN (MAC) operation per inference becomes simply, $\#MAC/inference=\sum_1^L(\#MAC_l)$. Based on this concept, we estimated the total number of MAC operations for ANN, and the total number of AC operations for direct-spike trained SNN and ANN-SNN converted network, for VGG9, ResNet9 and ResNet11. The ratio of ANN-SNN converted networks' (scheme1-scheme2) AC operations to direct-spike trained SNN AC operations to ANN MAC operations is (28.18-25.60):3.61:1 for VGG9 while the ratio is (11.67-18.42):5.06:1 for the ResNet9 and (9.6-10.16):2.09:1 for ResNet11 (for maximum accuracy condition).

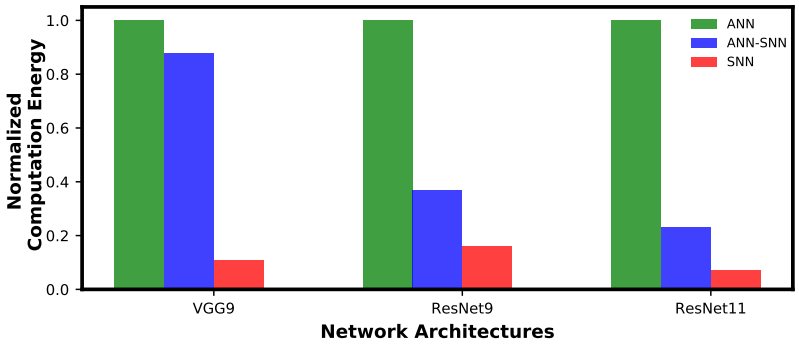


Figure 3.9. Inference computation complexity comparison between ANN, ANN-SNN conversion and SNN trained with spike-based backpropagation. ANN computational complexity is considered as a baseline for normalization.

However, a MAC operation usually consumes an order of magnitude more energy than an AC operation. For instance, according to [74], a 32-bit floating point MAC operation consumes 4.6pJ and a 32-bit floating point AC operation consumes 0.9pJ in 45nm technology node. Hence, one synaptic operation in an ANN is equivalent to ∼5.1 synaptic operations in a SNN. Moreover, 32-bit floating point computation can be replaced by fixed point com-

putation using integer MAC and AC units without losing accuracy since the conversion is reported to be almost loss-less [75]. A 32-bit integer MAC consumes roughly 3.2pJ while a 32-bit AC operation consumes only 0.1pJ in 45nm process technology. Considering this fact, our calculations demonstrate that the SNNs trained using the proposed method will be 7.81×-7.10× and 8.87× more computationally energy-efficient for inference compared to the ANN-SNN converted networks and an ANN, respectively, for the VGG9 network architecture. We also gain 4.6×-4.87× (2.31×-3.64×) and 15.32× (6.32×) energy-efficiency, for the ResNet11 (ResNet9) network, compared to the ANN-SNN converted networks and an ANN, respectively. The Fig. 3.9 shows the reduction in computation complexity for ANN-SNN conversions and SNN trained with the proposed methodology compared to ANNs.

It is worth noting here that as the sparsity of the spike signals increases with an increase in network depth in SNNs. Hence, the energy-efficiency is expected to increase almost exponentially in both ANN-SNN conversion network [11] and SNN trained with proposed methodology compared to an ANN implementation. The depth of network is the key factor for achieving a significant increase in the energy efficiency for SNNs in neuromorphic hardware. However, the computational efficiency does not perfectly align with the overall efficiency since the dominant energy consumption can be the memory traffic on von-Neumann computing hardware. The dataflows in asynchronous SNNs are less predictable and more complicated. Hence, a detailed study is required to estimate the overall efficiency of SNNs accurately.

Table 3.9. Iso-spike comparison for optimal condition. SNN time-steps corresponds to the latency to reach accuracy within 1% of maximum accuracy. ANN-SNN time-steps corresponds to the latency required for same number of spike/inference as SNN to occur. SNN and ANN-SNN accuracies are accuracies corresponding to respective latency.

Dataset	**Model**	**Time-steps**			**Accuracy (%)**		
		SNN	**ANN-SNN** [43]	**ANN-SNN** [11]	**SNN**	**ANN-SNN** [43]	**ANN-SNN** [11]
MNIST	LeNet	20	62	75	99.36	99.19	88.62
SVHN	VGG7	30	235	235	95.00	95.34	88.13
	ResNet7	30	200	200	95.06	95.63	95.48
CIFAR-10	VGG9	50	228	260	89.33	69.53	61.08
	ResNet9	50	390	490	89.52	89.51	90.06
	ResNet11	50	307	280	90.24	82.75	73.82

3.5.5 Iso-spike Comparison for Optimal Condition

In section 3.4.3, we observe that SNNs trained with proposed method achieve significant speed-up in both max-accuracy and iso-accuracy condition. However, in section 3.4.2.2, we found that the proposed method is in some cases (in an iso-accuracy condition) not more efficient than ANN-SNN conversions in terms of #spike/inference. The reason behind it is that an iso-accuracy condition may not be optimal for the SNNs trained with proposed method. In an iso-accuracy case, we have used max-accuracy latency (50 time-steps for MNIST and 100 time-steps for other networks) for direct-spike trained SNN, whereas most of the conversion networks used much less latency than the max-accuracy condition. In view of this, there is a need to determine the circumstances where our proposed method performs as well as or better than the SNN-ANN conversion methods on spike count, time steps, and accuracy. Consequently, in this section we analyze another interesting comparison.

In this analysis, we compare our proposed method and ANN-SNN conversion methods [11], [43] under the optimal condition at equal number of spikes. We define the 'optimal #time-steps' for SNNs trained with our proposed method as the #time-steps required to reach within ∼1% of peak accuracy (when the accuracy starts to saturate). Based on this definition, we observed that the optimal #time-steps for MNIST, SVHN, CIFAR10 networks are 20, 30, and 50, respectively. For this comparison, we recorded the achieved accuracy and #spike/inference of the SNNs trained with our proposed method for the corresponding optimal #time-steps. Then, we ran ANN-SNN networks for a length of time such that they use the similar number of spikes. In this iso-spike condition, we recorded the accuracy of the ANN-SNN networks (for both conversion methods) and the number of time-steps they require. The results are summarized in Table 3.9.

For comparatively shallower networks such as LeNet, VGG7 (VGG type) and ResNet7, ResNet9 (Residual type), the ANN-SNN conversion networks achieve as good as or slightly better accuracy at iso-spike condition compared to the SNNs trained with our proposed method. However, these ANN-SNN conversion networks require 3x-10x higher latency for inference. On the other hand, for deeper networks such as VGG9 and ResNet11, the ANN-SNN conversion networks achieve significantly lower accuracy compared to SNNs trained

with our proposed method even with much higher latency. This trend indicates that for deeper networks, SNNs trained with our proposed method will be more energy-efficient than the conversion networks under an iso-spike condition.

3.6 Conclusion

In this chapter, we propose a spike-based backpropagation training methodology for popular deep neural network architectures. This methodology enables deep SNNs to achieve competitive classification accuracies on standard image recognition tasks. Our experiments show the effectiveness of the proposed learning strategy on deeper SNNs (7-11 layer VGG and ResNet network architectures) by achieving the best classification accuracies in MNIST, SVHN and CIFAR-10 datasets among other networks trained with spike-based learning till date. The performance gap in terms of quality between ANN and SNN is substantially reduced by the application of our proposed methodology. Moreover, significant computational energy savings are expected when deep SNNs (trained with the proposed method) are employed on suitable neuromorphic hardware for the inference.

4. TRAINING DEEP SPIKING CONVOLUTIONAL NEURAL NETWORKS WITH STDP-BASED UNSUPERVISED PRE-TRAINING FOLLOWED BY SUPERVISED FINE-TUNING

4.1 Introduction

Given the deep hierarchical SNN models, it is still unclear which learning algorithm (i.e. unsupervised or supervised) is suitable for training the systems. Both the STDP and spike-based BP learning have been demonstrated to capture hierarchical features in SNNs [16]–[19], [25], [76], [77], but the insufficient classification performance of standalone STDP-trained networks, overfitting issues and unstable convergence behaviors of BP algorithm are a couple of obstacles toward efficient learning.

In this chapter, we propose a pre-training scheme using biologically plausible unsupervised learning, namely Spike-Timing-Dependent-Plasticity (STDP), in order to better initialize the parameters in multi-layer systems prior to supervised optimization. The multi-layer convolutional neural networks comprising of the convolutional and pooling layers followed by successive fully-connected layers are populated with bio-plausible leaky integrate-and-fire spiking neurons [33] to deal with sparse Poisson-distributed spike trains that encodes the pixel intensity in its firing rate. The proposed pre-training scheme trains the convolutional kernels using STDP algorithm in a layer-wise manner that enables them to self-learn features from input spike patterns. The pre-training enforces inductive bias to network parameters including the synaptic weights and neuronal thresholds, which provides a better starting point compared to random initialization. After finishing the pre-training, gradient descent BP algorithm fine-tunes the synaptic weights across all the layers leading toward the optimum local minima. The proposed strategy of using both the unsupervised and supervised learning algorithm can be referred to as 'semi-supervised learning'. We believe that biologically plausible unsupervised learning and state-of-the-art supervised deep learning algorithms can pave ways to jointly optimize the hierarchical SNNs for achieving efficient and competitive performance at the level of human brain.

The rest of the chapter is organized as follows. First, we explain the architecture of deep convolutional SNNs. Next, we describe the proposed semi-supervised training methodology, which includes the STDP-based unsupervised pre-training and BP-based supervised fine-tuning algorithms. Subsequently, we present the simulation results, which validate the efficacy of the semi-supervised training methodology for MNIST handwritten digit recognition. Then, we discuss the contributions of the proposed method and investigate how the pre-training helps the gradient-based optimization procedure.

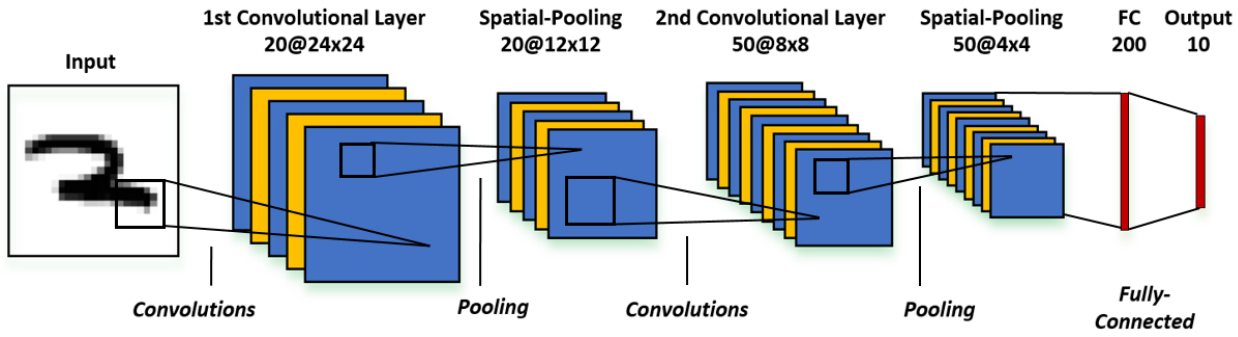


Figure 4.1. Architecture of the multi-layer convolutional spiking neural network consisting of an input layer, alternating convolutional and spatial-pooling layers, and final fully-connected layers for inference.

4.2 Multi-layer Convolutional Spiking Neural Network Topology

The recognition of high-dimensional input patterns necessitates multi-layer network topologies that can effectively learn hierarchical representations from input stimuli. In this chapter, we use a convolutional neural network model that consists of an input layer followed by intermediate hidden layers and the final output layer as illustrated in Fig.4.1. The input layer encodes the images as Poisson-distributed spike trains where the probability of spike generation is proportional to the pixel intensity. The hidden layers composed of alternating convolutional (C) and spatial-pooling (P) layers represent the intermediate stages of feature hierarchies. The spikes from the hidden layers are combined sequentially for final classification by the fully-connected (FC) layers. The convolutional and fully-connected layers consist of trainable parameters while the spatial-pooling layers are fixed a priori. The

weight kernels constituting the convolutional layers encode the feature representations at multiple hierarchical levels. The adapted convolutional kernels can appropriately detect the spatially correlated local features in the input patterns as a result of convolution, which inherently renders the network invariant to translation (shift) in the object location. Next, the spatial-pooling layer downscales the dimension of the feature maps produced by the previous convolutional layer while retaining the spatial correlation between neighborhood pixels in every feature map. For instance, a fixed 2×2 kernel (each having a weight of 0.25) strides through a convolutional feature map without overlapping and fires an output spike at the corresponding location in the pooled feature map if the summed spikes of the 4 input pixels within the window exceeds a threshold of 0.8. The pooling operation offers the following key benefits. First, it provides small amount of additional network invariance to input transformations while reducing the dimension of the convolutional feature maps. Furthermore, the pooling operation, by the virtue of downscaling the feature maps, enlarges the effective size of convolutional kernels in the subsequent layer. This helps successive convolutional layers to efficiently learn hierarchical representations from low to high levels of abstractions. The number of pooled feature maps is equal to the number of convolutional feature maps. The feature maps of the final pooling layer are unrolled into a $1-D$ vector that is fully-connected to the output layer which produces inference decisions. The fully-connected layer acts as a classifier to effectively incorporate the composition of features resulting from the alternating convolutional and pooling layers into the final output classes.

4.3 Proposed Semi-Supervised Learning Methodology

The proposed semi-supervised learning methodology is comprised of unsupervised pre-training followed by supervised fine-tuning using a spike-based gradient descent BP algorithm in a global fashion. The concept of unsupervised pre-training was introduced in [78] to efficiently train analog deep belief nets, a generative model comprising several stacked restricted Boltzmann machines, using a fast greedy layer-wise training algorithm. In [79]–[81], the authors employed unsupervised learning mechanisms such as contrastive divergence and de-noising auto-encoder to hierarchically pre-train successive layers of deep belief nets. In

spiking domain, [17], [19], [64], [65], [76], [82] have explored semi-supervised learning to train deep SNNs, with layer-wise unsupervised learning using spike-based auto-encoder/sparse-coding/STDP-based methods followed by supervised learning at the final classification layer. However, we use STDP-based unsupervised pre-training to discover useful characteristics and underlying structures of data to appropriately condition and initialize the synaptic weights and neuronal firing thresholds for a given pattern recognition task. After pre-training the network, we use the spike-based gradient descent BP algorithm to fine-tune the synaptic weights end-to-end in a manner that minimizes discrepancy between the actual outputs and target labels. We now describe the individual STDP-based unsupervised and BP-based supervised learning mechanisms.

Unsupervised Pre-training using Spike-Timing-Dependent-Plasticity

Spike-Timing-Dependent-Plasticity (STDP) is a biologically plausible unsupervised learning mechanism that self-learns synaptic weights based on the degree of temporal correlations between the pre- and post-synaptic spike events. As shown in Fig.2.4(a), the pre-synaptic trace resets to 1 when pre-synaptic spike arrives and exponentially decays over time. Hence, the pre-synaptic trace encodes the timing correlation between pre- and post-neuronal spikes in the positive timing window. The strength (weight) of synapse is potentiated if a pre-synaptic spike triggers the post-neuron within a period of time that is determined by a threshold, namely χ_{offset}. The synaptic weight is depressed for larger spike timing differences. The STDP weight updates are applied to the synapses only at the time instances of post-synaptic firing. Specifically, we use the weight-dependent positive-STDP rule whose characteristic is formulated as follows.

$$\Delta w = \eta_{STDP}(\mathrm{e}^{\frac{t_{pre}-t_{post}}{\tau_{pre}}} - \chi_{offset})(w_{max} - w)(w - w_{min}) \tag{4.1}$$

where Δw is the change in the synaptic weight, η_{STDP} is the learning rate, t_{pre} - t_{post} is the timing difference between pre- and post-synaptic spikes, τ_{pre} is the time constant controlling the length of the STDP timing window, and w_{max} (w_{min}) is the maximum (minimum) bound on the synaptic weight. The amount of weight change has a nonlinear dependence on the

current weight (w), which is specified by the product of (w_{max}-w) and (w-w_{min}). Smaller the absolute value of the current weight, larger is the ensuing weight change and vice versa as illustrated in Fig. 2.4(b). Such nonlinear weight-dependent updates ensure a gradual increase (decrease) of the synaptic weight towards the maximum (minimum) bound, thereby improving the efficiency of synaptic feature learning. Note that the synaptic weights are locally updated in an unsupervised way based on the spiking behaviors of pre-/post-neurons at adjacent layers.

In convolutional SNNs, the weight kernels locally inter-connecting the successive layers stride over the pre-neuronal maps to construct the output feature maps at every time step. In the event of a post-spike, the time difference between corresponding pre- and post-neuronal spikes is used to conduct individual STDP update on the convolutional weights. In case of multiple post-neuronal spikes in an output feature map, averaged STDP updates are applied to the kernel weights. Accordingly, the STDP learning enables the weight kernels to self-learn useful features from the complex input patterns. In addition to performing STDP updates on the weight kernels, we modulate the firing threshold of the units in the corresponding feature map to enable kernels (among the feature maps in a convolutional layer) to learn different representations of input patterns. In the event of a post-neuronal spike in a convolutional feature map, we uniformly increase the firing threshold of all the post-units constituting the feature map. In the period of non-firing, the firing threshold of the feature map exponentially decays over time. Such threshold adaptation, referred as homeostasis [83], balances the firing threshold with respect to the strength of kernel weights and effectively prevents convolutional kernels in a feature map from dominating the learning. In addition, the negative synaptic weights preclude the need for lateral inhibitory synaptic connections among feature maps in a layer (by regulating spiking activities of units within feature map) that is otherwise essential for competitive feature learning. In previous studies, STDP learning has been demonstrated to self-learn convolutional kernels layer-by-layer for training multi-layer convolutional SNNs [17], [76]. In this chapter, we exploit the unsupervised feature learning capabilities of STDP learning for appropriately initializing the convolutional weights and corresponding neuronal firing thresholds in multi-layer systems. We greedily pre-train each convolutional layer one at a time using the unsupervised STDP learning and uniform threshold adaptation scheme.

We begin by training the first convolutional layer that enables the weight kernels to discover low-level characteristic features from input patterns in an unsupervised manner. At every time step, the convolutional kernels slide over the input maps to detect the characteristic features and construct output feature maps. The unit in output feature maps fires when the convolutional kernel captures the characteristic features, and the weight kernel is updated with STDP and the threshold adaptation mechanism. After the first convolutional layer is trained, the adjusted weight kernels and neuronal firing thresholds are frozen to feed the input again for estimating the average firing rate of units in the output feature maps. The generated feature maps of first convolutional layer (the nonlinear transformations of inputs) are spatially pooled and passed to the next convolutional layer to extract the higher-level representations in hierarchical models. This process is repeated until all convolutional layers are pre-trained. Note that we do not modify the synaptic weights of the fully-connected layer (or the classifier) during the pre-training procedure. Therefore, the unsupervised pre-training mechanism, in essence, initially finds out unique features and underlying structures of input patterns for the task at hand prior to supervised fine-tuning.

Supervised Fine-tuning using Spike-based Backpropagation

In this section, we first discuss the standard supervised backpropagation (BP) learning that is a widely used first-order gradient descent algorithm for ANN [44], and subsequently detail its spike-based adaptation used in this chapter. The standard BP algorithm involves forward propagation and error back-propagation. During the forward propagation, an input pattern and its output (target) label are respectively presented to measure the corresponding loss function, which is a function of discrepancy between target labels and predicted outputs from the current network parameters. The error backpropagation is thereafter used to compute the gradients of the loss function with respect to each synaptic weight for determining their contributions to the final output loss. The synaptic weights are modified based on the individual gradient in the direction to minimize the output loss. The above steps are iteratively applied over mini-batches of input patterns to obtain the optimal network parameters, which facilitate the training loss to converge to a local minima. In this chapter, the standard

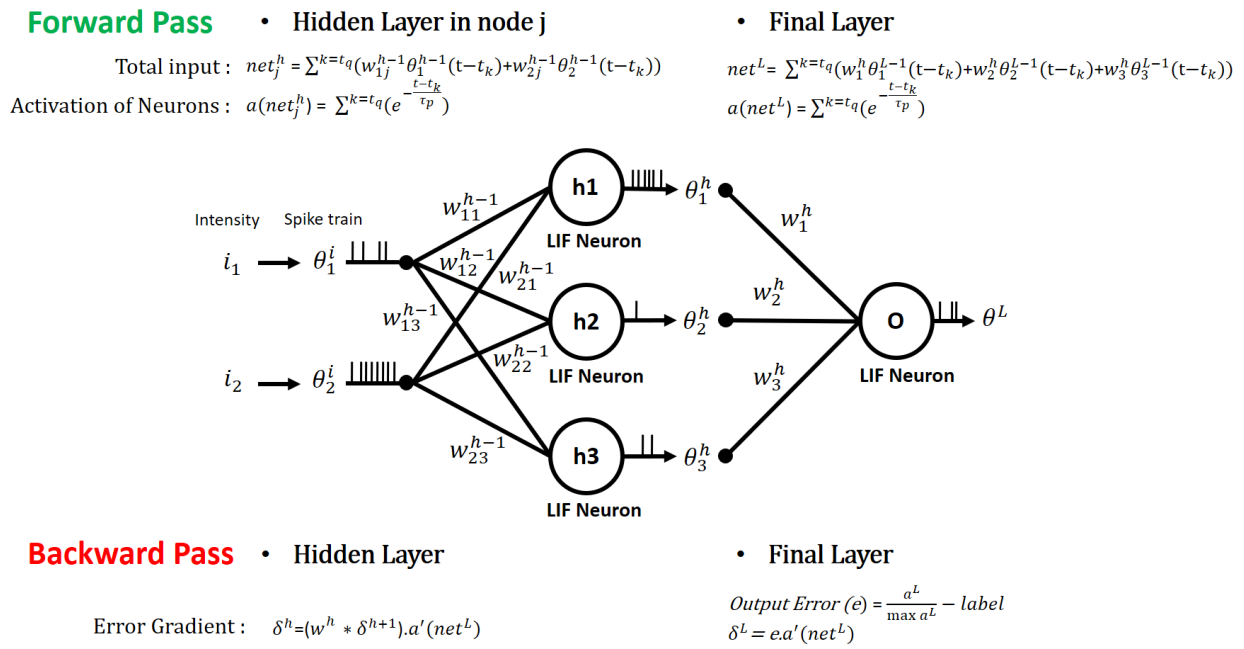


Figure 4.2. Illustration of spike forward and backward propagation of a multi-layer SNN consisting of LIF neurons. In forward pass, the spiking neuron integrates the input current (net) generated by the weighted sum of the pre-neuronal spikes with the interconnecting synaptic weights and produces an output spike train. In backward pass, the derivatives of designated loss function with respect to each synaptic weight are calculated from chain-rule.

BP technique is adapted for SNNs by taking into account the event-driven characteristics for optimizing the weights directly using the spike input signals. It is important to note that the primary difference between ANNs and SNNs lies in the dynamics of the output produced by the respective neuron models. The spiking neurons communicate over time by means of spike pulses that are discrete and non-differentiable signals. This is in stark contrast with the differentiable continuous (scalar) values from the analog neurons such as *sigmoid*, *tanh* and *ReLU* functions [4], [84]. In spike-based BP algorithm, we low-pass filtered the post-spike trains to obtain a pseudo derivative by creating differentiable activation function (explained below). This allows the final output loss to be propagated backward to hidden layers for updating the associated synaptic weights suitably.

During the forward propagation, the input pixel values are converted to Poisson-distributed spike trains and directly fed to the SNN. The sum of Dirac-delta pulses (denoted by x_i for

the i_{th} input neuron) are weighted by inter-connecting synaptic weights (w_{ij}^{l}) to be integrated to post-neurons as illustrated in Fig. 4.2 and formulated as equation (4.2).

$$net_{j}^{l+1}(t) = \sum_{i=1}^{n^l} w_{ij}^{l} * x_{i}(t), where\ x_{i}(t) = \sum_{k=1}^{t} \theta_{i}(t - t_k) \tag{4.2}$$

where net_{j}^{l+1} is the resultant current received by the j_{th} post-neuron at $(l+1)_{th}$ layer, n^l denotes the number of neurons in l_{th} layer, t_k represents the time instant at which pre-neuron spikes. The post-neurons fire output spikes when the respective membrane potentials exceed a definite neuronal firing threshold, after which the potentials are reset and the spikes are broadcast to the subsequent layer. This process is successively carried out by the post-neurons in every layer based on the incoming spikes received from the preceding layer to produce spike trains over time as shown in Fig. 4.2. The differentiable activation of the spiking neuron, which defines the highly non-linear relationship between the weighted pre-neuronal spikes and post-spike firing rate, is generated by low-pass filtering the individual post-spike train as formulated below.

$$Activation\ of\ neuron,\ a_{j}(t) = \sum_{k=1}^{t} exp(-\frac{t - t_k}{\tau_p}) \tag{4.3}$$

$$Final\ output\ error,\ e_{j} = \frac{a_{j}^{L}}{max(a^L)} - label_{j} \tag{4.4}$$

$$Loss\ function,\ E = \frac{1}{2} \sum_{j=1}^{n^L} e_{j}^{2} \tag{4.5}$$

The activation, a_j, of an LIF neuron is computed by integrating the unit's spikes (at time instants t_k in equation (4.3)) and decaying the resultant sum in the time period between successive spikes. The time constant (τ_p), which determines the rate of decay of the neuronal activation, accounts for the non-linear membrane potential decay and reset mechanisms that influence the spiking dynamics of the post-neuron. The activation of the output neurons in the fully-connected layer (classifier) is normalized to obtain a probability distribution over all final class predictions for a given input pattern. The final error (e_j) for each output neuron is evaluated by comparing the normalized output activation with the target label ($label_j$)

of the presented input as shown in equation (4.4). The corresponding loss function (E in equation (4.5)) is defined as the mean square of the final error over all the output neurons.

Next, we detail the gradient descent backpropagation algorithm that is used to minimize the output loss in SNNs. We first estimate the gradients of the output loss with maximum likelihood at the final output layer and back-propagate the gradients all the way down through the network using recursive chain rule [44]. The gradient with respect to the weights of hidden layers are obtained as described by the following equations.

$$\delta^L = \mathrm{e}.a(net^L) \tag{4.6}$$

$$a(net^L) = a(t) + 1 = \sum_{k=1}^{t}(-\frac{1}{\tau_p}\mathrm{e}^{-\frac{t-t_k}{\tau_p}}) + 1 \tag{4.7}$$

$$\delta^h = ((w^h)^T * \delta^{h+1}).a(net^h) \tag{4.8}$$

The quantity, δ^L, henceforth referred as the 'error gradient', represents the gradient of the output loss with respect to the net input current received by the post-neurons in the final output layer. It can readily be computed (as shown in equation (4.6)) by multiplying the final output error (e in equation (4.4)) with the derivative of the corresponding post-neuronal activation ($a(net^L)$) in equation (4.7)). Note that '.' denotes element-wise multiplication while '*' indicates matrix multiplication in the equations (4.2 – 4.9). The neuronal activation (as described in equation (4.3)) is non-differentiable with respect to input current because of discrete time series output signals. To overcome this, we obtain a pseudo-derivative of post-neuronal activation by adding a unity value to the time derivative of the corresponding activation as formulated in equation (4.7). The time derivative of neuronal activation reflects highly non-linear (leaky) characteristics of LIF neuron model and adding a unity value facilitates ignoring the discontinuity (step jump) that arises at each spike time. The error gradient, δ^h, at any hidden layer is recursively estimated by back-propagating the error gradient from the successive layer ($(w^h)^T * \delta^{h+1}$) and multiplying it with the derivative of neuronal activation ($a(net^h)$) as formulated in equation (4.8). It is worth mentioning here that the presented spike-based BP algorithm mitigates the vanishing gradient phenomena,

because the derivatives of the spiking neuronal activation (shown in equation (4.7)) do not saturate unlike saturating activation functions.

$$\triangle w^l = \frac{a^l}{max(a^l)} * (\delta^{l+1})^T \tag{4.9}$$

$$w^l = w^l - \eta_{BP} \triangle w^l \tag{4.10}$$

The derivative of the output loss with respect to the weights interconnecting the layers l and $l+1$ (∇w^l in equation (4.9)) is determined by multiplying the transposed error gradient at $l+1$ (δ^{l+1}) with the normalized activation of the neurons in layer l. In case of convolutional neural networks, we back-propagate the error in order to get the partial derivatives of the loss function with respect to the given output feature map. Then, we average the partial derivatives over the output map connections sharing the particular weight to account for the effective updates of kernel weights. Finally, the calculated partial derivatives of loss function are used to update the respective weights using a learning rate (η_{BP}) as illustrated in equation (4.10). Iteratively updating the weights over mini-batches of input patterns leads the network state to a local minimum, thereby enabling the network to capture hierarchical internal representations of the data.

4.4 Results

We demonstrate the capability of the proposed semi-supervised learning strategy on the handwritten digit MNIST dataset [31] using a MATLAB-based custom simulation framework. The MNIST dataset contains 60k training and 10k testing (grayscale) images belonging to 10 categories. For the experiments, we implement relatively shallow and deep multi-layer convolutional SNN topologies, which comprise of 28×28 input image, convolutional (C) layers using 5×5 sized weight kernels, spatial-pooling (P) layers with 2×2 non-overlapping pooling regions followed by successive fully-connected (FC) layers. The detailed multi-layer neural network topologies are as follows: the shallow network is $28 \times 28 - 36C5 - 2P - 10FC$ and the deep network is $28 \times 28 - 20C5 - 2P - 50C5 - 2P - 200FC - 10FC$. The initial synaptic weights are randomly assigned at each layer following the weights initialization

scheme [16]. The neuronal firing thresholds (v_{th}) are set proportional to the strength of the synaptic distribution as shown below.

$$w^l \in U[-\sqrt{\frac{3}{n^l}}, \sqrt{\frac{3}{n^l}}], v_{th} = \alpha\sqrt{\frac{3}{n^l}}, \alpha > 0 \tag{4.11}$$

where w^l denotes the synaptic weight matrix connecting layers l and $l+1$, $U[-k, k]$ indicates the uniform distribution in the interval between k and k, and n^l is the size of the l_{th} layer.

Table 4.1. Parameters used in the experiments

Parameter	**Value**
STDP Type	Positive STDP
Decay Constant of Membrane Potential (τ_m)	10 ms
Decay Constant of Synaptic Trace (τ_{pre})	1.5 ms
Decay Constant of Post-neuronal Activation Function (τ_p)	100 ms
Training Time Duration (STDP, BP)	25, 100, 50 ms
Inference Time Duration	200 ms
Mini-batch Size	100
Maximum Input Rate (STDP, BP, Inference)	200 , 500, 500 Hz
Convolutional Kernel Size/Stride	5×5, 1
Spatial-pooling Non-overlapping Region/Stride	2×2, 2
Threshold Initialization Constant (α) for,C, FC Layer without Pre-training	5, 3

We train the multi-layer convolutional SNNs using the proposed semi-supervised learning strategy, which comprises initial unsupervised pre-training and subsequent supervised fine-tuning (or spike-based BP) procedures using the parameters listed in Table 4.1. In every iteration of training, a subset (mini-batch) of randomly sampled training images are fed to the system such that the static inputs are converted stochastically into spike events, wherein the firing rate encodes the pixel intensity. During the unsupervised pre-training, we present a fraction of training data to the network for 25 ms (assuming a simulation time-step of 1 ms) and adjust each convolutional layer one at a time. After the layer-wise pre-training of convolutional layers, the kernel weights with respect to the neuronal firing threshold are appropriately initialized and conditioned for further fine-tuning. Next, we conduct gradient-based BP learning, which evaluates the gradients of a loss function with respect to the synaptic weights through forward and backward propagations. During supervised fine-tuning, we present all training samples (excluding the ones used for pre-training) for 100 ms in the first epoch and full-training samples for 50 ms in subsequent

epochs. Note that passing the full training examples once through a network denotes an epoch, which consists of 600 iterations in case of MNIST dataset given the mini-batch size of 100. The learning rate is kept constant throughout the unsupervised and the supervised learning, respectively.

Table 4.2. Learning rate and mean standard deviation of classification errors in shallow and deep multi-layer networks

Network Topology	**Shallow Multi-layer Network**		**Deep Multi-layer Network**		
Model (Corresponding Models in Fig. 4.3)	Without Pre-training (Red)	With Pre-training (Blue)	Without Pre-training (Red)	With Pre-training (Blue)	With Pre-training (Yellow)
Learning Rate	0.4	0.4	0.18	0.18	0.35
Variance (Mean STD)	10.57%	0.083%	0.146%	0.110%	0.099%

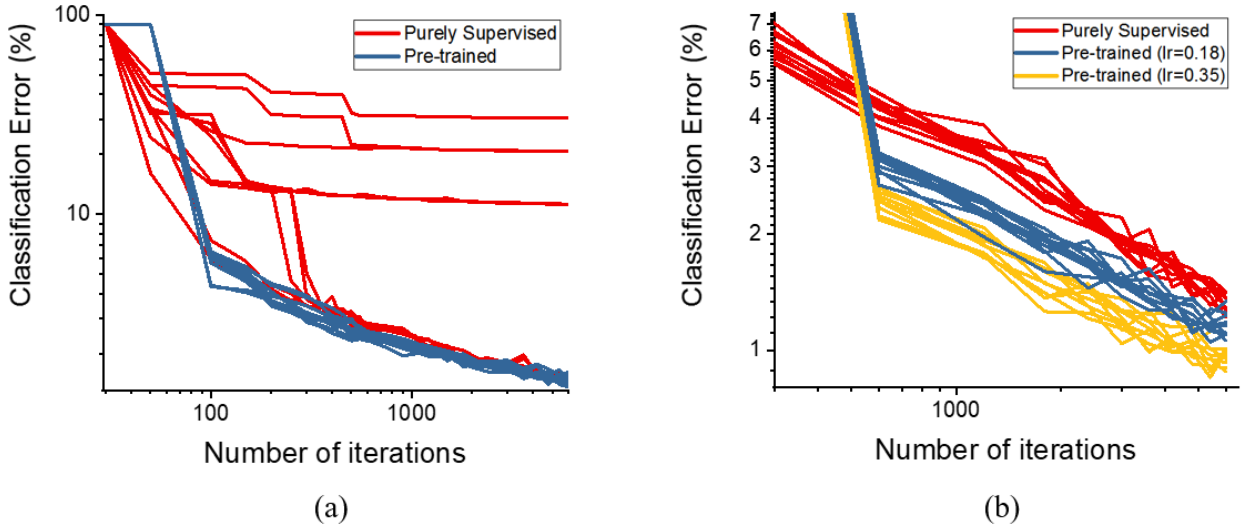


Figure 4.3. The classification accuracies (in log scale) on (a) shallow and (b) deep multi-layer convolutional spiking neural networks of pre-trained and supervised model starting from different states of randomly initialized synaptic weights.

First, we discuss the effectiveness of our semi-supervised learning methodology by evaluating the classification performance of the shallow and deep multi-layer networks on the MNIST test dataset. We compare our proposed semi-supervised training strategy (i.e. pre-trained model) against standalone gradient-based supervised optimization without pre-training (i.e. purely supervised model) for both shallow and deep networks. The spike-based gradient descent training follows an identical criterion in both pre-trained and purely supervised models

with the exception of parameter initialization (i.e., unsupervised STDP-based pre-training vs random initialization). Fig. 4.3(a) shows the classification error comparison between the two scenarios for shallow multi-layer network, which started from 10 different initialization of the weight state. Note, the learning rate across the 10 different cases for both pre-trained (blue) and purely supervised (red) models, in Fig. 4.3(a), is identical. The optimization procedure is greatly influenced by the learning rate, which should be kept within a moderate range to enable stable convergence without overshooting from the minima and diverging. As shown in Fig. 4.3(a), the purely supervised models (for certain weight initializations) get stuck in poor local minima, thereby yielding high variance (or standard deviations) on classification error. In contrast, the pre-trained models mostly enter the appropriate convergence routes without being trapped in poor local minima consistently yielding lower error with increasing number of iterations. Among the supervised models that did not get stuck in bad local minima, the pre-trained models still outperform them in terms of classification performance. We conduct a similar comparison as that of Fig. 4.3(a) for the deep network topology as illustrated in Fig. 4.3(b). We observe similar results with the pre-trained model (blue) yielding a lower classification error than a purely supervised model (red). In fact, the pre-trained model converges to a lower classification error with fewer number of iterations, which establishes the effectiveness of the STDP-based pre-training procedure. It is noteworthy to mention that deep networks (in case of purely supervised training) do not get stuck in poor local minima for different initializations due to the enriched parameter space available for optimization. This enriched parameter space also allows us to use a higher learning rate without overfitting. We observed that increasing the learning rate significantly lowers the classification error achieved with the pre-trained model (yellow in Fig. 4.3(b)). Additionally, the classification error of pre-trained model shows lower variance than the purely supervised networks that started independently from different initialized weights as described in Table 4.2. Thus, we can infer that STDP-based pre-training improves the robustness of the overall learning procedure.

To further quantify the benefits of the STDP-based pre-training method, we plotted the classification errors with respect to training efforts for both the purely supervised and pre-trained models that have identical weight initialization in the beginning of training.

Table 4.3. Comparison of the SNNs classification accuracies on MNIST digit recognition task.

Model	Architecture	Learning Method	Accuracy
Esser et al.[85]	Deep Fully-connected	Offline learning, conversion	99.42%
Hunsberger et al.[56]	Deep Fully-connected	Offline learning, conversion	98.37%
Diehl et al.[38]	Deep Convolutional	Offline learning, conversion	99.1%
Diehl et al.[13]	Two-layer Fully-connected	Unsupervised STDP	95.0%
Kheradpisheh et al.[17]	Deep Convolutional	Layerwise STDP + SVM classifier	98.4%
Panda et al.[19]	Deep Convolutional	Convolutional Autoencoder	99.05%
Lee et al.[16]	Deep Convolutional	Backpropagation	99.31%
Semi-supervised Learning (This work)	Deep Convolutional	STDP-based Pretraining + Backpropagation	99.28%

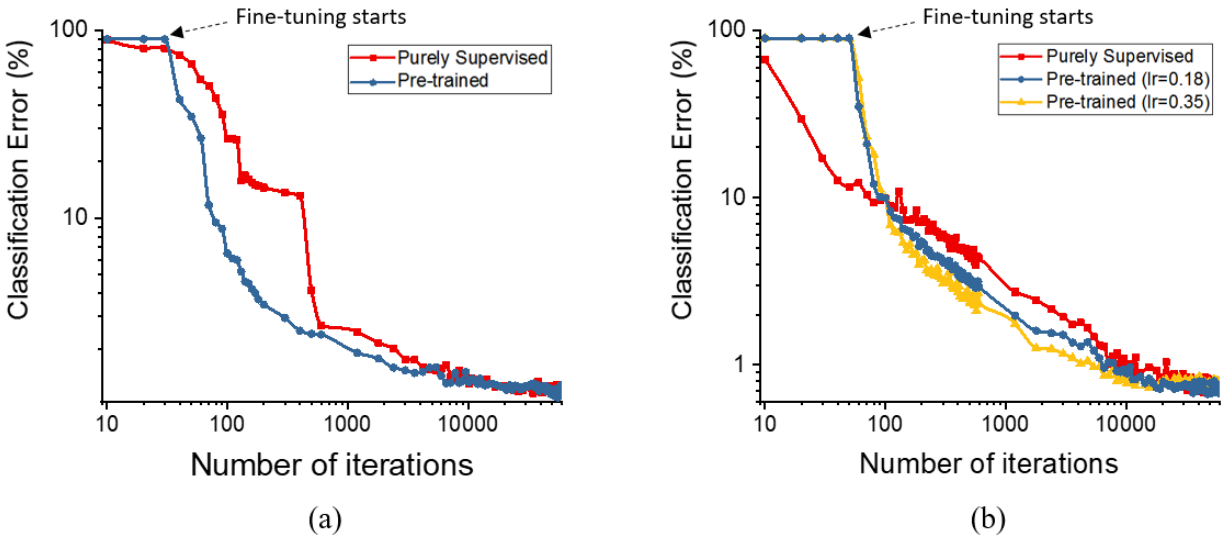


Figure 4.4. The plots show the classification accuracies on (a) shallow and (b) deep multi-layer convolutional spiking neural network as the semi-supervised optimization runs. The x-axis is the number of iterations (in log scale) and y-axis is classification accuracies (in log scale) on testing data.

We quantify training effort as the total number of training iterations required for error convergence. The plots in Fig. 4.4 illustrate the classification performance of the pre-trained model (blue, yellow) with respect to the purely supervised model (red). We observe that the pre-trained model (yielding very high error during the unsupervised pre-training stage) starts to outperform the purely supervised model with supervised fine-tuning yielding consistently lower error for both the shallow and deep topologies. Note, the classification error remains high initially ($\sim$90%) in case of a pre-trained model, because the fully-connected layers are not trained during the STDP-based pre-training phase. Besides lower error rate, the proposed

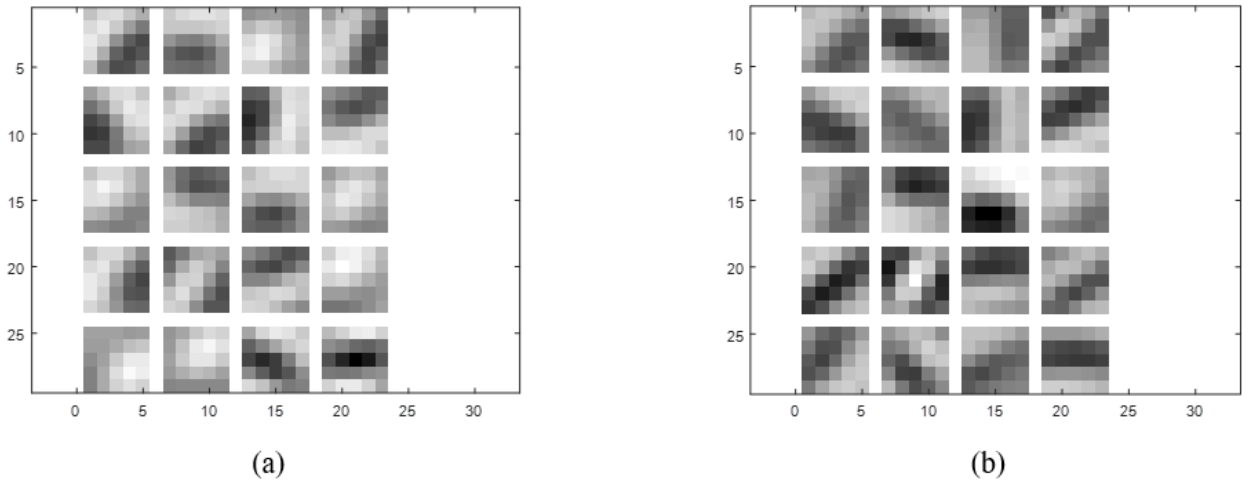


Figure 4.5. The weight kernels of (a) purely supervised and (b) pre-trained model in first convolutional layer.

semi-supervised training also yields faster training convergence. Specifically, the convergence time (in which the shallow multi-layer network reaches 2% classification error) with STDP-based pre-training (1200 iterations) is significantly lower than that of purely supervised case (3000 iterations). Similarly, the pre-trained deep network achieves 1% classification error after 4800 iterations, whereas the randomly initialized network with spike-based BP takes 10200 iterations. Essentially, the speed of optimization to reach certain amount of testing error improves by $\sim 2.5\times$ for both shallow and deep multi-layer network with STDP pre-training as compared to purely supervised gradient BP. The boosted performance of gradient-based supervised fine-tuning provides an insight that the efficient unsupervised feature learning prior to the fine-tuning phase significantly reduces the training effort to facilitate convergence. We believe that unsupervised initialization helps to mitigate the difficult highly non-convex optimization problem by better initializing and conditioning the network parameters. Eventually, the classification accuracies of shallow multi-layer network saturates at the amount of lowest error rates of 1.20% (purely supervised model) and 1.23% (pre-trained model) averaged over 130-150 (78000-90000) training epochs (iterations). The classification errors of purely supervised model and pre-trained model for training deep multi-layer networks saturate at the 0.77% and 0.72%, respectively. The classification results shown are comparable to the state-of-the-art results as compared in Table 4.3. Fig. 4.5

shows the adjusted weight kernels in first convolutional layer for purely supervised (a) and pre-trained (b) model after 150 training epochs. The weight kernels of the pre-trained model in Fig. 4.5(b) indicate more definite shapes of pattern characteristics compared to those from the purely supervised model in Fig. 4.5(a).

Table 4.4. Mean standard deviation of classification errors that are initialized with different weight initialization schemes in deep multi-layer networks

Model (Corresponding Models in Fig. 4.6)	Without Pre-training (Red)	With Pre-training (Blue)	With Pre-training (Yellow)
Lee Initialization [16]	0.146%	0.110%	0.099%
Glorot initialization [35]	0.171%	0.131%	0.116%

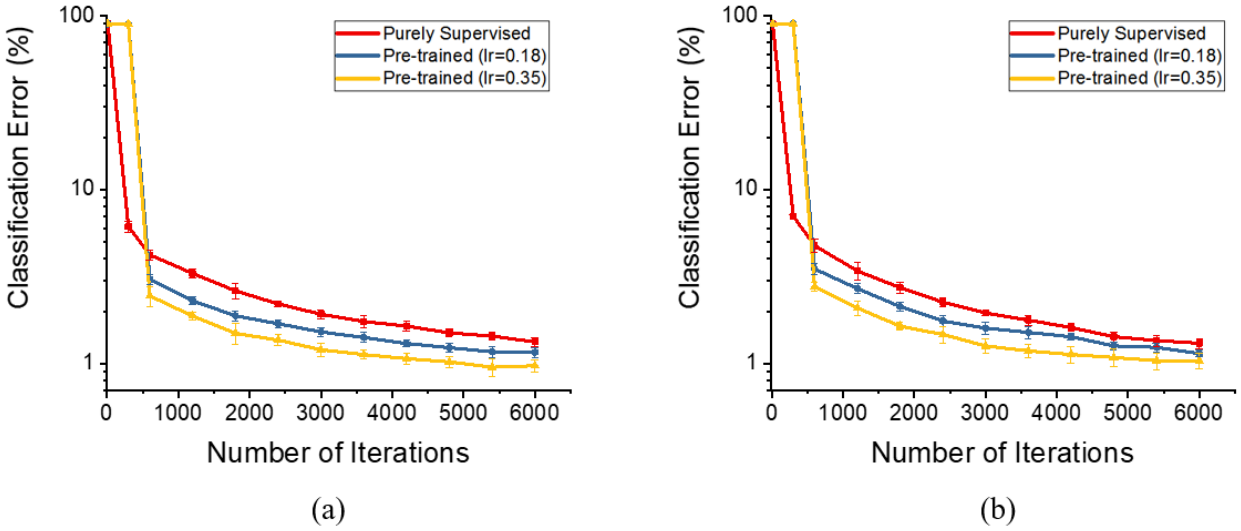


Figure 4.6. The classification accuracies (in log scale) on the deep multi-layer convolutional spiking neural networks of pre-trained and supervised model starting from (a) Lee initialization (b) Glorot initialization.

Lastly, lets try to answer the following question: Does the STDP-based pre-training also provide the benefits when the network is initialized with different random initialization? To address this question, we perform an experiment that initializes the parameters of deep multi-layer SNNs with another initialization scheme ('Glorot initialization' [35]) and train with the proposed semi-supervised learning strategy. We use unsupervised STDP to pre-train the SNNs (initialized with Glorot initialization) and measure the classification performances (that started from 10 different states of random weights) while fine-tuning the networks with gradient descent backpropagation algorithm. The classification performance shows faster

training convergence (1800 iterations to reach 2% error) and improved robustness compared to the networks without STDP-based pre-training (3000 iterations to reach 2% error). Note that pre-trained models (initialized with Glorot initialization) show slightly slower training convergence time compared to Lee Initialization [16] pre-trained models (1200 iterations to reach 2% error). Fig. 4.6 shows the classification performances with respect to training efforts for the purely supervised and pre-trained models of each initialization scheme ((a) Lee initialization versus (b) Glorot initialization). Fig. 4.6(b) and Table. 4.4 depict similar trends: pre-trained models achieve better classification performances and lower variances (measured from 10 different states of random weights) compared to purely supervised models. Therefore, we infer that STDP-based pre-training also helps to better initialize and condition the network parameters in different initialization scheme such as Glorot initialization.

4.5 Discussion

Our proposal of STDP-based unsupervised pre-training is demonstrated to achieve improved robustness and significant speed-up in training procedure. Conceptually, the benefits of the semi-supervised learning strategy come from the inherent attributes of two different learning mechanisms. First, the unsupervised STDP learning automatically determines the useful features from high-dimensional input patterns that strengthens the connections between strongly correlated neurons. Hence, the quick and simple modifications are facilitated so that the nonlinear representations are simply extracted based on the degree of correlation between neurons in adjacent layers. Moreover, the nature of unsupervised STDP learning is less prone to the overfitting problem than the supervised learning [17]. These peculiarities allow the unsupervised STDP mechanism to be an effective initializer for directing the network to an optimal starting point in the parameter space at the beginning of gradient descent optimization. On the other hand, supervised BP learning is a complex, global and gradient-based algorithm, which adjusts the synaptic weights proportional to the degree of their contributions to the final loss in the direction of minimizing the errors. The gradient descent algorithm is susceptible to the initial condition of network parameters, which causes variable convergence and necessitates large number of training data to generalize the

network well. Note that there are numerous studies to appropriately initialize the network parameters in the domain of ANN [35], [50], [81]. In SNNs, the conversion from adapted ANN to SNNs [11], [12], [37], [38], [56] is one popular methodology to take advantage of state-of-the-art deep learning algorithms and techniques. The conversion technique shows remarkable classification performances, nevertheless there are issues that prevent them from becoming universal. It is inevitable to avoid the classification accuracy degradation due to ANN-to-SNN conversion, which becomes higher when dealing with real sensory data from event-driven dynamic vision sensors [86], [87]. The weight-normalization scheme, which effectively converts the network parameters, is still an active research field. In addition, the privacy issues can not be overlooked in case of disclosing, sharing and destroying the personal (credential) data generated from edge devices for ANN training in cloud services (or data centers). Consequently, all-spiking neural network systems, which efficiently train and test the deep SNNs by direct input spike events, allow to protect privacy and increase the availability of private data to the artificial intelligence systems. As mentioned before, the initial conditions of SNN are pre-defined based on the network parameters, which are the synaptic weights and firing threshold of spiking neurons. However, it is still not evident how to initialize the multi-layer SNN systems in an optimal way. In this chapter, we leverage STDP unsupervised learning to appropriately initialize the network parameters in a data-driven manner prior to the supervised gradient descent BP learning.

Table 4.5. Final testing and training NLL costs (averaged out over 130-150 epochs) in shallow and deep multi-layer networks

Network Topology	**Shallow Multi-layer Network**		**Deep Multi-layer Network**		
Model (Corresponding Models in Fig. 4.7)	Without Pre-training (Red)	With Pre-training (Blue)	Without Pre-training (Red)	With Pre-training (Blue)	With Pre-training (Yellow)
Final Testing NLL	0.1317	0.1266	0.0658	0.0627	0.0659
Final Training NLL	0.0894	0.0861	0.0234	0.0169	0.0118

We performed an additional experiment to investigate how the proposed STDP-based unsupervised pre-training helps the subsequent gradient-based supervised fine-tuning compared to purely supervised training from random weight initialization. We hypothesize that unsupervised pre-training effect helps either optimize or generalize the systems. In this context, the optimization helps to locate the network to a better starting point in the pa-

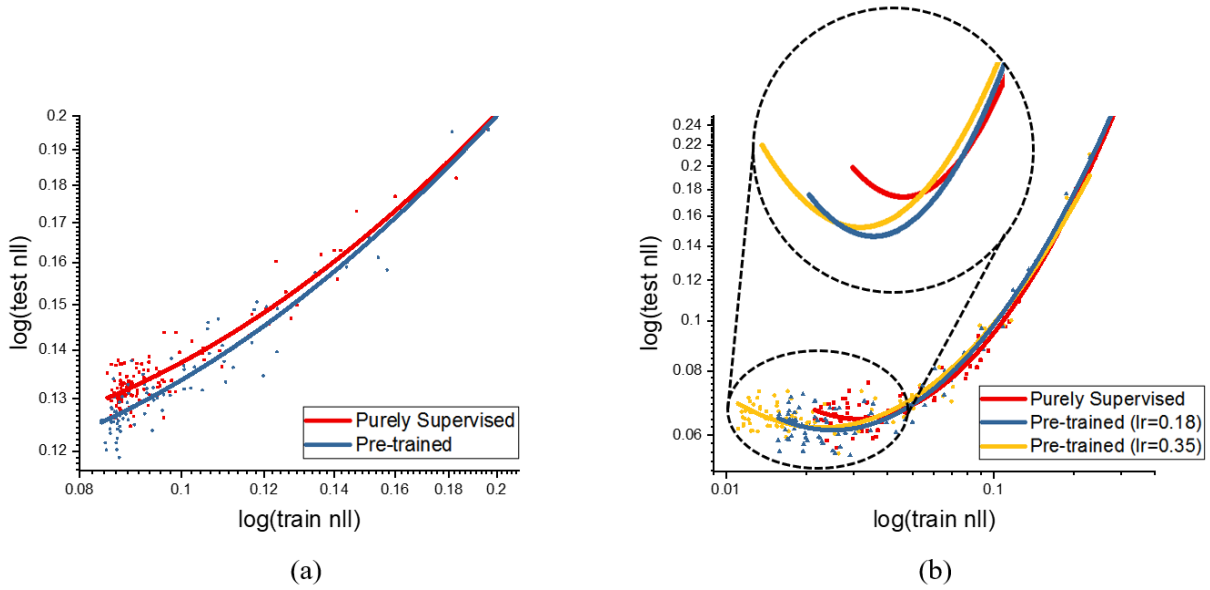


Figure 4.7. The plots show the NLL cost on (a) the shallow and (b) deep multi-layer convolutional spiking neural network. The horizontal and vertical axis indicate the NLL costs (in log scale) on training and testing data, respectively.

rameter space, which induces lower training error. On the other hand, the generalization effect prevents the network from overfitting too closely to training sample, which results in lowering the errors on data that are not seen during the training. We trained shallow and deep multi-layer networks over 150 epochs with and without pre-training and evaluated the component sum of negative-log-likelihood (NLL) costs on testing and training data to highlight the performance gap between the two scenarios. The negative-log-likelihood function is formulated below.

$$Negative\ Log\ Likelihood = \sum_{\mathrm{i}=1}^{n^L} x_{\mathrm{i}} \log p_{\mathrm{i}}(x) + (1 - x_{\mathrm{i}}) \log \left(1 - p_{\mathrm{i}}(x)\right) \tag{4.12}$$

where n^L represents the size of final layer, x is the output target labels and $p(x)$ denotes the normalized firing rate of final output neurons. Fig. 4.7 presents the testing NLL cost with respect to the training NLL cost for both shallow and deep network optimization. Table 4.5 shows the testing and training NLL costs averaged over 130-150 epochs. During the supervised BP learning, the pre-trained model yields a lower training NLL cost with

the same training effort (representative of faster convergence) and the final training NLL cost of the pre-trained model saturates at a lower range than the purely supervised model as depicted in Table 4.5. This trend indicates that the unsupervised initialization induces the systems to be rapidly optimized and achieves better training error. The unsupervised pre-training, in effect, initially deploys the network to a parameter space where the initial point is closer to the local optima. In addition, we analyzed the test cost with respect to the training cost to measure the generalization effect of unsupervised pre-training. As the optimization proceeds toward the end, the testing NLL cost value saturates or starts to slightly increase because of overfitting, whereas the training NLL cost continually decreases as shown in Fig. 4.7. However, we observe that the overfitting phenomenon occurs at the stage of lower training NLL cost in case of pre-trained models (for both shallow and deep cases) in comparison to the purely supervised training. The inset in Fig. 4.7(b) highlights this effect wherein we observe that the pre-trained models saturates to lower convergence region (testing NLL cost) while delaying the overfitting phenomena. Note, overfitted neural network systems perform worse on test data (or data unseen during the training). Therefore, we infer that the pre-trained model can generalize better than the purely supervised model by means of pre-conditioning of the network parameters such that overfitting issue is mitigated. In essence, the STDP-based unsupervised initialization scheme provides an equivalent effect of classic regularization techniques such as early stopping [88], $L1/L2$ weight decay [89] and dropout [2], which explicitly constrain the training model like adding penalty to the loss function or adding restriction on parameters.

4.6 Conclusion

Recent efforts in spiking neural networks have been focused toward building multi-layer systems to hierarchically represent highly nonlinear and complex functions. However, training hierarchical systems remains a difficult problem because of their inherent high dimensionality and non-convexity. In this chapter, we have shown that the convolutional spiking neural network comprising multiple hidden layers can be pre-trained with layer-wise unsupervised STDP learning and fine-tuned with supervised gradient descent BP algorithm. The

unsupervised pre-training extracts the underlying structures from high dimensional input patterns in order to better initialize the parameters and supervised gradient-based BP algorithm takes the hierarchical system to optimal local minima. The proposed semi-supervised strategy benefits the training procedure to be more invariant to randomly assigned initial parameters, yields faster training and better generalization compared to purely supervised optimization without pre-training. We believe that STDP-based unsupervised initialization scheme coupled with state-of-the-art deep learning backpropagation algorithm can pave the way toward effectively optimizing deep spiking neural networks.

5. TOWARDS UNDERSTANDING THE EFFECT OF LEAK IN SPIKING NEURAL NETWORKS

5.1 Introduction

A spiking neuron integrates the inputs over time and fires a spike-output whenever the membrane potential exceeds a threshold. Computational models for spiking neurons use a Leaky Integrate and Fire (LIF) model, which has a built-in leaky behavior in the membrane potential, or use simpler Integrate and Fire (IF) with no leak in the membrane potential [90]. Since biological neuron models have been reported to contain leak in the membrane potential [91], it would be important to quantitatively analyze the advantages and disadvantages of using leaky behavior.

To that end, we focus on two aspects of the leak effect on SNN models: robustness and spiking sparsity. Ideally, the neural network models are expected to predict reliable outcomes for unseen or even noisy data under sparse spiking events. In addition, compared to ANNs, the main advantage of SNNs is the energy-efficient event-based computing capability, in which the synaptic operations occur only when spike-inputs arrive. To that effect, the computational efficiency of SNNs considerably improves as spike signals become sparser for specialized SNN hardware platforms such as TrueNorth [92] and Loihi [93].

Although various models have been proposed that resemble realistic biological neuronal mechanisms [94]–[96], they are often too complex from a computational point of view. Also, there is a lack of understanding of how each of the factors determining the biological neuronal response can be effectively used in learning. Hence, we investigate the general and simple IF and LIF neuron models [90] that are analytically tractable. We present a comprehensive and comparative analysis between models with and without leak to delve deeper into the role that leak plays in learning.

The main contributions of this chapter are as follows,

- A theoretical analysis of the first-order phenomenological LIF neuron model is introduced to investigate its low-pass filtering effect. As a step toward this goal, from frequency domain analyses, we show that the presence of leak helps to cut-off some of the input

components beyond a certain frequency, thereby aiding the networks to predict more robust outcomes for noisy spike-inputs.

- We examine the effect of leak on computational requirements in multi-layered SNNs. Compared to SNNs with IF model, the ones with LIF model converges with decreased sparsity of spike signals when trained with surrogate-gradient based backpropagation, resulting in reduced computational efficiency.

- We conduct experiments to validate the robustness of multi-layered SNNs with IF and LIF neuron models using popular vision datasets including SVHN and CIFAR-10. Furthermore, we analyze the improved performance of LIF models by investigating the frequency spectrum of spikes and how well the network generalizes to previously unseen data.

5.2 Spiking Neural Network Fundamentals

5.2.1 Spiking Neuron Model

The spiking neurons (generally modeled as IF or LIF) are fundamental units in SNNs. The sub-threshold dynamics of an LIF neuron is governed by,

$$\tau_m \frac{dU}{dt} = -(U - U_{rest}) + RI, \qquad U \leq V_{th} \tag{5.1}$$

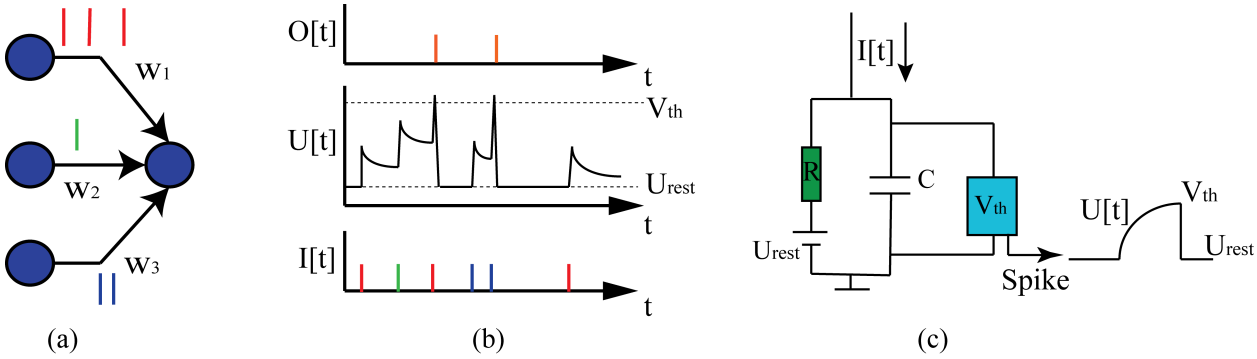


Figure 5.1. An LIF neuron, (a) a schematic connection between three pre-neurons to one post-neuron, (b) temporal dynamics of membrane potential in the post-neuron, (c) equivalent circuit model of the LIF neuron.

where U is the membrane potential, I denotes the input current that represents the weighted summation of spike-inputs, τ_m indicates the time constant for membrane potential decay, R represents membrane leakage path resistance and U_{rest} is resting potential. Fig. 5.1 depicts the dynamics of LIF neuron and an equivalent circuit model. The input current is accumulated in the membrane potential that decays exponentially over time. The degree of exponential decay is determined by the membrane time constant (τ_m). When the membrane potential exceeds the firing threshold (V_{th}), the neuron is triggered to emit an output-spike and resets the membrane potential to the resting state. The spike-output can be represented as,

$$O[t] = \begin{cases} 1, & if\ U[t] > V_{th} \\ 0, & otherwise \end{cases} \tag{5.2}$$

where $O[t]$ and $U[t]$ denote the spike-output and the membrane potential, respectively, at time instant t. The neuronal dynamics in equation (A.1) can be represented by an equivalent RC circuit model [97] as illustrated in Fig. 5.1(c). The parallel RC branch acts as a low-pass filter [98], which has the membrane time constant (τ_m) as RC where C is the membrane capacitance. The analysis of neuronal responses with respect to various frequency components stimulates the following discussion in the next subsection.

5.2.2 Frequency Domain Analyses

In this subsection, the response of an LIF neuron model is analyzed in relation to the membrane time constant (τ_m). We investigate the role of leaks in filtering out some of the signal components in the high-frequency range when driven by white Gaussian noise. In order to quantify the low-pass filtering effect, we employ the coherence function, $C(\omega)$ which is a commonly used metric in signal processing [99] to estimate the power transfer from the input to the output. When the input to a system is $s(t)$ and the corresponding output is $x(t)$, the coherence between them is defined as,

$$C_{x,s}(\omega) = \frac{|S_{x,s}(\omega)|^2}{S_{x,x}(\omega)S_{s,s}(\omega)}, \tag{5.3}$$

where $S_{x,s}(\omega)$ is the cross-spectrum of output (x) with input (s), $S_{x,x}(\omega)$ and $S_{s,s}(\omega)$ are the autopower spectrum of $x(t)$ and $s(t)$, respectively. To study the response of the neuron model described by equation (A.1), we measure the coherence as a function of frequency. We model the inputs to the neuron as white Gaussian noise current and derive the corresponding coherence between the noise input and the output spike train. The resulting coherence function $C_{x,s}(\omega)$ is as follows:

$$C_{x,s}(\omega) = \frac{2D_{st}}{D}\frac{r_0\omega^2}{1+\omega^2}\frac{\left|\mathscr{D}_{i\omega-1}\left(\frac{\mu-V_{th}}{\sqrt{D}}\right) - e^{\Delta}\mathscr{D}_{i\omega-1}\left(\frac{\mu-U_{rest}}{\sqrt{D}}\right)\right|^2}{\left|\mathscr{D}_{i\omega}\left(\frac{\mu-V_{th}}{\sqrt{D}}\right)\right|^2 - e^{2\Delta}\left|\mathscr{D}_{i\omega}\left(\frac{\mu-U_{rest}}{\sqrt{D}}\right)\right|^2}, \tag{5.4}$$

where D_{st} is the intensity of the white noise stimulus, D is total noise intensity (for our case $D{=}D_{st}$), r_0 is the output firing rate, $\mathscr{D}(x)$ is a parabolic cylinder function, μ is a parameter denoting DC part of the input (defined in appendix section 5.6.1) and $\Delta = \frac{U_{rest}^2 - V_{th}^2 + 2\mu(V_{th}-U_{rest})}{4D}$. The detailed derivation of equation (5.4) starting with equation (5.1) is provided in the appendix section 5.6.

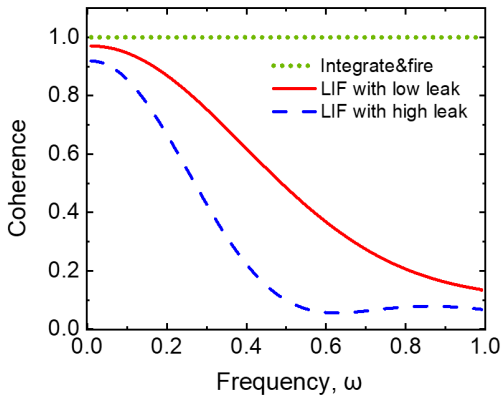


Figure 5.2. Illustration of frequency response for IF and LIF neuron models. The horizontal and vertical axes represent the frequency components and coherence function, respectively.

To analyze the frequency responses of the neuron model, the coherence functions of the IF and the LIF models with high and low leak cases in relation to frequency (ω) are plotted in Fig. 5.2. This figure shows that the IF model (green) transmits all input components to the outputs across the entire frequency spectrum. On the other hand, for LIF models (red and blue), the coherence function decreases as the frequency increases, thereby cutting-

off the high-frequency components propagating to the output. Hence, contrary to the IF model, the LIF model can negate the noise input components beyond a certain frequency limit. Similar low-pass filtering of information for LIF neurons has been reported in [98], [100]. The authors in [101] also discussed similar characteristics of LIF neurons from a neuroscience perspective. Drawing inspirations from such phenomenon, our next goal is to explore whether the low-pass filtering effect can enable multi-layered SNNs with leaky neuron models to be more robust against noisy inputs. The following subsections focus first on the training methodology adopted in this chapter, followed by the noisy input generation methods and corresponding experiments.

5.2.3 Gradient Descent Learning in SNNs

The gradient-based method, namely backpropagation (BP) learning [44], is a widely employed method for training traditional deep ANNs. While ANN neuron models with continuous functions (such as $sigmoid$, $tanh$ or $ReLU$) are compatible with the gradient-based learning, it has been a challenge to directly train SNNs with BP method in their native form. This is due to the spike-output being binary-valued (i.e., zero or one), which renders the spike generation function non-differentiable and discontinuous. To get around this issue, standard BP has been adapted for the spike-based learning domain which we refer to as 'spike-based backpropagation'. The spike-based BP method overcomes the discontinuous spiking functionality by approximately estimating the surrogate gradient of spike generation function. Several surrogate gradient methods have been introduced in the literature [16], [63], [70]. In this chapter, we employ the LIF neuronal surrogate gradient function that accounts for the leaky behavior as proposed in [102].

The training procedure is composed of two phases (e.$g.$, forward and backward). In the forward phase, the hidden layer neurons accumulate the weighted sum of spike-inputs in the membrane potential. When this potential exceeds the threshold, the neuron fires a spike-output and resets the potential to the resting state (zero). Otherwise, membrane potential decays exponentially. The final layer neurons do not generate spike output and decay over time, accumulating a weighted sum of spike-inputs. At the last time step, the final

prediction outcomes are estimated by dividing the final layer membrane potential ($U_L[T]$) by the total number of time-steps (T). Then, the final errors are evaluated by comparing the final prediction outcomes with the ground truth ($label$). The loss function ($Loss$) is obtained by computing the summation of squared error as shown below,

$$Loss = \frac{1}{2}(\frac{U_L[T]}{T} - label)^2, \quad \frac{\partial O_l[t]}{\partial U_l[t]} = \frac{1}{V_{th} + \epsilon}(O_l[t] > 0), \tag{5.5}$$

Algorithm 2 Procedure of spike-based backpropagation learning for an iteration.

Input: pixel-based inputs ($inputs$), total number of time steps ($\#timesteps$), number of layers (L), weights (W), membrane potential (U), membrane time constant (τ_m), firing threshold (V_{th})
Initialize: $U_l[t] = 0,\ \forall l = 1, ..., L$
// **Forward Phase**
for $t \leftarrow 1$ **to** $\#timesteps$ **do**
    // generate Poisson spike-inputs of a mini-batch data
    $O_1[t] = Poisson(inputs)$;
    for $l \leftarrow 2$ **to** $L - 1$ **do**
        // membrane potential integrates weighted sum of spike-inputs
        $U_l[t] \ = U_l[t-1] + W_l O_{l-1}[t]$
        if $U_l[t] > V_{th}$ **then**
            // if membrane potential exceeds V_{th}, a neuron fires a spike
            $O_l[t] = 1,\ U_l[t] = 0$
        else
            // else, membrane potential decays exponentially
            $O_l[t] = 0,\ U_l[t] = \mathrm{e}^{-\frac{1}{\tau_m}} * U_l[t]$
    // final layer neuron does not fire
    $U_L[t] \ = \mathrm{e}^{-\frac{1}{\tau_m}} * U_L[t-1] + W_L O_{L-1}[t]$
// **Backward Phase**
for $t \leftarrow \#timesteps$ **to** 1 **do**
    for $l \leftarrow L - 1$ **to** 1 **do**
        // evaluate partial derivatives of loss with respect to weight by unrolling the network over time
        $\triangle W_l[t] = \frac{\partial Loss}{\partial O_l[t]} \frac{\partial O_l[t]}{\partial U_l[t]} \frac{\partial U_l[t]}{\partial W_l[t]}$

where $\frac{U_L[T]}{T}$ is the final prediction outcome. In the backward phase, the final errors are propagated backward while unrolling the network in time using the surrogate gradient method. This procedure is often regarded as Backpropagation Through Time (BPTT) [71]. The sur-

rogate gradient of LIF neuronal function is computed by combining the straight through estimation [45] and leak correctional term (ϵ) as given by the second equation in equation (5.5). Here, the straight-through estimation (i.e., $\frac{1}{V_{th}}$) calculates the derivative of IF neuronal function and ϵ compensates the leaky effect of the membrane potential. Finally, the network parameters are updated based on the partial derivatives of the loss with respect to weights for all discrete time steps. The trained SNNs can incorporate temporal and leak statistics from direct spike-inputs over time. The pseudo-code of the spike-based BP learning is given in Algorithm 2.

5.3 Poisson Spike Generation under Noisy Environments

In section 5.4.2, the spike-inputs with external random noise are used for experimentally evaluating the noise robustness (i.e., the capability of maintaining a certain prediction accuracy under stochastic perturbations) of multi-layered SNNs. Keeping that goal in mind, here we explain the noisy spike-input generation methods used in this chapter. Specifically, two different sources of random noise are considered, namely *Gaussian noise* and *Impulse noise* [103]. Under each noise source, two noise injection scenarios are introduced for producing the noisy spike-inputs. Each noisy spike-input generation procedure is depicted in Algorithm 3. In our analysis, clean spike-inputs refer to homogeneous Poisson spikes where spike-firing probability remains constant in the entire period of input generation. Therefore, inter-spike-intervals (ISI) of such homogeneous Poisson spikes conform to Poisson statistics.

For scenario 1, an independent random noise is added to an image pixel at each time step. The combination of pixel input and noise is compared with an uniformly distributed random number to generate Poisson-distributed spike-inputs. Hence, for a given period of time, the stream of spike-inputs incorporates the noise over time. In this case, the firing probability varies at every time step, which results in inhomogeneous Poisson spikes with randomly varying firing probability every time step. Here, what varies is the ISI distribution. For scenario 2, an independent random noise is added (at each time step) to the Poisson spike trains generated from the original image pixels. The major difference between two scenarios is whether the random noise is added before or after comparing with a random

Algorithm 3 Poisson spike generation scheme under noise

Input: pixel-based inputs ($inputs$), total number of time steps ($\#timesteps$), external random noise (ξ), uniform random number ($\mathscr{X}$)
Output: spike-based inputs ($O_1[t]$)
for $t \leftarrow 1$ **to** $\#timesteps$ **do**
    if $Scenario\ 1$ **then**
        // External noise (ξ) is added to input pixel
        $inputs_c = inputs + \xi$
        // If noisy input ($inputs_c$) is greater than uniform random number, a spike-input ($O_1[t]$) is generated
        if $inputs_c > \mathscr{X}$ **then**
            $O_1[t] = 1$
        else
            $O_1[t] = 0$
    if $Scenario\ 2$ **then**
        // External noise (ξ) is added to input channel
        if $inputs > \mathscr{X}$ **then**
            $O_1[t] = 1 + \xi$
        else
            $O_1[t] = \xi$

number (Poisson spike generation process). Note, in scenario 1, spikes are generated as a post-process of adding noise to image pixels, making the input spike train strictly binary-valued, but in scenario 2, noise is added directly to the spikes, so the resultant noisy spikes contain perturbations around their clean spike values (0 or 1). Since the spikes are generated from the Poisson process in scenario 2, the ISI distribution remains identical to that of clean spike-inputs. Instead, the amplitude of each spike varies. The random noise injection process is performed in the input layer only.

5.4 Experiments

5.4.1 Experimental Setup

We examine the robustness of multi-layered SNNs against noisy spike-inputs on two standard vision benchmarks, namely SVHN and CIFAR-10, which are composed of color (three-dimensional) inputs. We experiment with multi-layered SNN models, which comprise of 32×32 color inputs, convolutional (C) layers with 3×3 weight kernels, average-pooling (P)

layers with fixed 2×2 kernel followed by fully-connected (FC) layers. The details of the chosen SNN models are as follows: model used for CIFAR-10 is (32×32-64C3-64C3-2P-128C3-128C3-2P-256C3-256C3-256C3-2s-1024FC-10o) and model used for SVHN is (32×32-64C3-64C3-2P-256C3-256C3-256C3-2s-1024FC-10o). We follow the training protocols as described in [102]. Each network model with different membrane time constant is independently trained with clean training data. Note, the membrane time constant is not considered as a trainable parameter and remains fixed during training and testing. All network models are trained with mini-batch spike-based BP for 150 epochs with a batch size of 64, while decreasing the learning rate at 70^{th} and 100^{th} epoch. After normalizing each image sample to zero mean and scaling to the range [-1, 1], Poisson spike trains are generated for 100 time-steps during training and testing. The reported results are the average score from three independently trained networks. We implemented the multi-layered SNNs using Pytorch deep learning package.

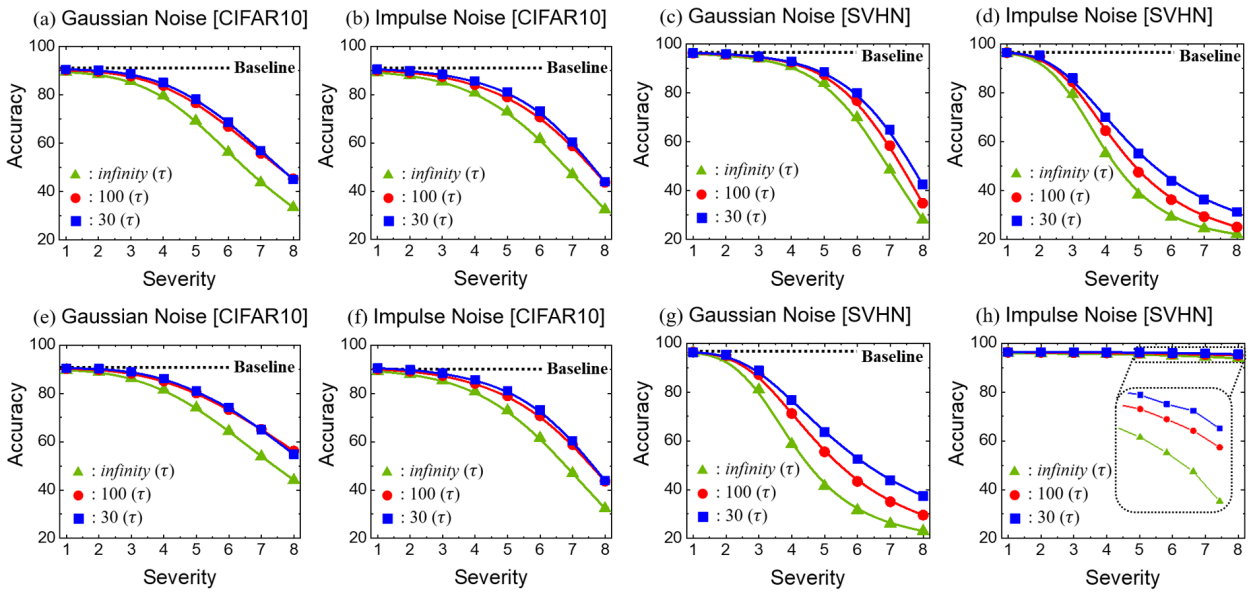


Figure 5.3. Classification accuracy at each level of noise severity. The horizontal and vertical axes present the input noise severity and classification accuracy, respectively. (a,b,c,d) Results from noisy input generation scenario 1. (e,f,g,h) Results from noisy input generation scenario 2.

Table 5.1. Comparison between the network models with different leak amounts. The first row corresponds to baseline accuracy. The second and the third rows correspond to the sum-squared errors averaged over 130–150 epochs for testing and training data, respectively. The fourth and the fifth rows correspond to average spiking activity and the total number of synaptic operations, respectively.

Dataset	**CIFAR-10**			**SVHN**		
τ_m	30	100	inf	30	100	inf
Accuracy (%)	89.65	90.19	90.3	96.12	96.32	96.32
SSE_{Test}	**2.93**	3.45	3.83	**0.72**	0.75	0.82
SSE_{Train}	**1.88**	1.92	2.2	**1.26**	1.4	1.6
Spikes (%)	**9.45**	5.26	4.94	**14.07**	12.09	11.85
#Synaptic Ops	**1.59E9**	7.92E8	7.18E8	**3.99E9**	3.88E9	3.79E9

5.4.2 Robustness against Noisy Spike-inputs

First, we compare the noise robustness results with different membrane time constants (e.g, τ_m= 30, 100 and $infinity$). The LIF neuron models are associated with relatively smaller membrane time constants (e.$g.$, τ_m= 30 and 100) compared to IF neuron model with an infinitely large membrane time constant (e.$g.$, τ_m= $infinity$). The robustness of each SNN model is measured in terms of the stability of the classification accuracy against noisy spike-inputs. The performances of SNNs are scored across eight severity levels with each noise type (e.$g.$, *Gaussian noise* and *Impulse noise*). The severity level indicates the strength of input noise.

In both benchmark datasets (CIFAR-10, SVHN), the baseline testing accuracy is almost the same under different leak parameters as presented in the first row of Table 5.1. Fig. 5.3 shows the accuracy results with increasing level of noise severity across different benchmarks (first row: noisy spike generation scenario 1, second row: noisy spike generation scenario 2). For both the noisy spike generation scenarios, SNNs with LIF neurons (blue, red) achieve improved noise robustness whereas the ones with IF neurons (green) suffer from severe accuracy degradation for high noise severity levels as displayed in Fig. 5.3. We would like to mention here that all network models are trained on clean spike-inputs, but tested with noisy ones. The models trained with the highest amount of leak (blue) retain the baseline accuracy to a greater extent compared to a non-leaky model. The LIF model with τ_m= 100

shows relatively higher accuracy degradation compared to one with τ_m= 30. However, both models show improved robustness compared to the IF model. These trends hold true for all noisy spike-input generation scenarios. In our experiments, we observed that training loss diverges when the chosen membrane time constant is too small ($\tau_m < 30$). In this case, the spiking activities decrease severely due to extremely high leak while passing through the layers, causing convergence issues in multi-layered SNN training [102]. However, for converged network models, SNNs with leaky neurons exhibit better stability against noisy spike-inputs.

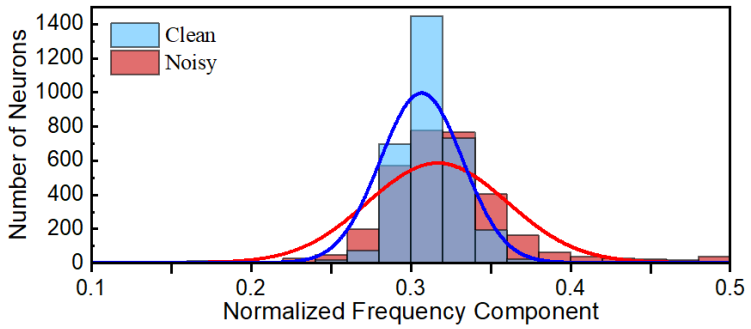


Figure 5.4. Histogram of the spectrum of spike trains per image for clean and noisy (Gaussian noise) inputs with corresponding distribution curves.

5.4.3 Spectrum Analysis

To analyze the improved noise robustness of LIF models, we perform a spectrum analysis of inputs and the corresponding network outputs for both clean and noisy data. In general, the noise spectrum contains components over a wide frequency band. The single-sided spectra of input-spike trains (averaged over test samples) for the clean and the noisy cases are shown in Fig. 5.4. It can be observed that the mean spectrum distribution with noisy spike-inputs remains roughly the same compared to the clean case. However, this spectrum of noisy data spreads over a wider band, resulting in more components in higher frequency bands. These changes in the spectrum distribution can significantly alter the spike patterns propagating through the layers compared to the clean input. As previously explained in section 5.2.2, the leaky neuron models only pass inputs with low-frequency components. Hence,

the leaky neuron models can eliminate some of the high-frequency noise components, thus helping to maintain the baseline performance. However, the low-frequency noise components pass through the LIF and IF models in a similar way. Thus, the accuracy degradation due to such components remains alike for both leaky and non-leaky neurons.

Next, let us consider the spectrum of the target output neuron (node corresponding to the ground truth label) in the final layer, since the changes concerning this output neuron largely determine the correct or wrong classification. For each image, we measure the average spectrum of the target output neuron and calculate the critical frequency up to which the significant power (70%) of the total spectrum resides. This critical frequency distribution is examined over all the samples and plotted in Fig. 5.5(a,b,c). Interestingly, with an increasing amount of leak, we found that the mean spectrum shifts towards the left. We anticipate this shift towards the lower frequency band is owing to the inherent low-pass filtering effect of leak. The normalized mean critical frequency components for the target neuron corresponding to $\tau_m = infinity$, 100 and 30 are 0.345, 0.317 and 0.255 respectively, for the clean testing samples, while for the noisy inputs (for noise severity level of five), the same frequency components become 0.35, 0.33 and 0.298, respectively. These outcomes along with Fig. 5.5(a,b,c) clearly indicate that frequency components of target neuron's output response become higher with noisy spike-inputs compared to the clean input case. As IF neurons have much wider pass-band, the higher frequency components are not filtered out as shown in Fig. 5.5(a), thus making the network more prone to have noise errors. In contrast, for the LIF models, most of the high-frequency components are eliminated through the low-pass filtering effect, as demonstrated in Fig. 5.5(b,c) which results in maintaining the baseline accuracy.

5.4.4 Analyses of Generalization

In order to ascertain the improved noise robustness from another perspective, we extend our analysis to generalization. We hypothesize that leaky neuron models enable SNNs to better generalize to previously unseen examples, and examine the impact of leak on generalization. While training multi-layered SNNs over 150 epochs, we recorded the sum of squared error (SSE) on the testing and the training samples to highlight the performance differences.

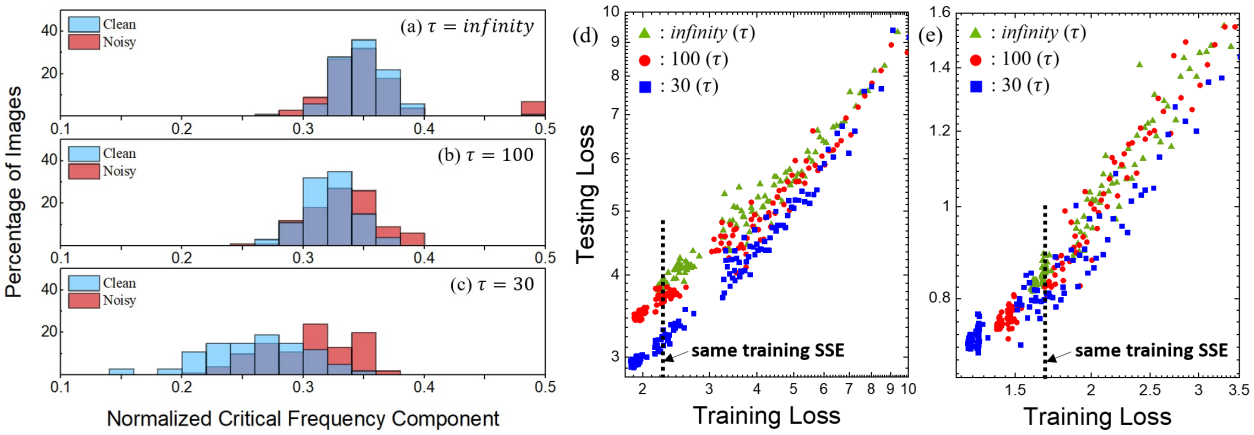


Figure 5.5. Histogram of average normalized critical frequency components of target output neuron for (a) $\tau_m = infinity$, (b) $\tau_m = 100$ and (c) $\tau_m = 30$. The sum-squared errors of test data with respect to the ones of training data on (d) CIFAR-10 and (e) SVHN benchmarks. The horizontal and vertical axes present the sum-squared error (in log scale) on train and test data, respectively.

As training progresses towards the end, we found that SNNs with LIF models yield lower testing SSE with the same training effort and reach lower final testing and training SSE than the ones with IF models. The second and the third rows of Table 5.1 present the testing and training SSE averaged over 130–150 epochs. We also analyze the testing SSE attained as a function of the training SSE. Fig. 5.5(d,e) shows the testing SSE with respect to the training SSE for different membrane time constants. We found that at the same training SSE, LIF models (blue, red) yield lower testing SSE than IF models (green), hinting towards better generalization. Notably, the advantage of better generalization is the mitigation of overfitting in large neural networks [66], [104].

5.4.5 Input Activity Analysis

While the enhanced robustness achieved through leaky neuron models is advantageous, it is also pivotal to consider the associated computations and energy costs of using LIF and IF models. To infer an output class, SNNs need the spike-inputs to be fed over a number of times steps, performing event-based synaptic operations that take place only when spike-inputs arrive. In this respect, the total number of synaptic operations is typically considered

as a metric for benchmarking the computational costs in neuromorphic hardware [8], [93]. This subsection explores the impact of leaky neuron models on the spiking sparsity and the number of computations, two critical factors that directly determine the computational efficiency of SNNs. In Table 5.1, the fourth and fifth rows present the average spike activities and the total synaptic operations, respectively, for different leak parameters. We found that the overall spiking activities increase with a higher leak, thereby resulting in more synaptic computations. An important insight from here is that, with respect to the degree of leak, there exists a trade-off between noise robustness and compute requirements.

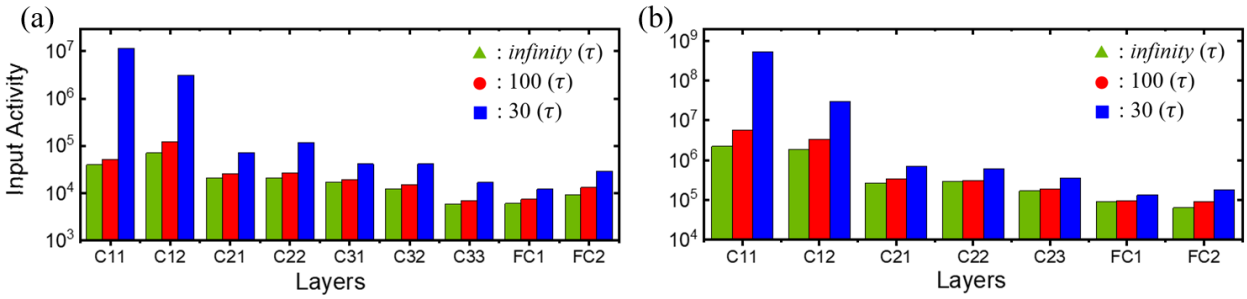


Figure 5.6. Layer-wise Euclidean norm of the weighted sum of spike-inputs of multi-layered SNNs for (a) CIFAR-10 and (b) SVHN datasets.

To investigate the reason behind the increased spiking activities with higher leak, we measure the Euclidean norm of the weighted sum of spike-inputs (referred to as 'ENWSI' subsequently) over time for each hidden layer. We would like to note that ENWSI is representative of a combination of spiking activities and weights that determine the net input information to the corresponding layers. Fig. 5.6 illustrates the ENWSI for different membrane time constants (e.$g.$, τ_m= 30, 100 and $infinity$). We found that SNNs with LIF neurons (blue, red) receive higher ENWSI across the layers than those with IF neurons (green). The model with the largest leak (blue) receives the highest ENWSI compared to the other models under consideration as evidenced in Fig. 5.6.

It is widely understood that if IF and LIF neurons were to receive the same weighted sum of inputs, the LIF neurons would produce comparatively lesser outputs due to their inherent leak. However, that would lead the LIF models not to have enough spikes in the deeper layers due to the layerwise gradual reduction in spiking activities. Hence, the resultant

network would fail to converge with acceptable accuracy. To overcome the leak effect and to have sufficient spiking activities for proper training, spike-based BP training tailors the LIF models to increase the weighted sum of hidden layer input activities beyond what is needed for IF models. Consequently, LIF models converge to configurations with increased spiking activities, allowing for sufficient weighted sum of input activities in the deeper layers.

5.5 Discussion and Conclusion

In the neuroscience literature, the existence of leak in biological neurons has been reported in the context of sodium ion channels [91], [105], synaptic transmission in visual cortex [106], [107], etc. SNN models take the bio-plausibility of leak into account through the leaky neuron models [94]–[96]. In those models, the leak acts as a hyperparameter that controls the decay of membrane potentials in the neurons over time. However, the effect of leak in learning and the resultant neuronal responses have not been studied comprehensively, to the best of our knowledge. Recognizing this gap, in this chapter, we investigate IF and LIF neuron models to analyze the role that leak plays in learning and their impact on noise robustness and spiking sparsity. It is to be noted that there are opportunities to further explore the effect of leak, especially in terms of other more complicated neuron models closer to biological ones (e.*g*., Hodgkin-Huxley [94] and exponential integrate-and-fire [96]). However, till date, there exists a lack of suitable methodologies for training deep SNNs for complex learning tasks with acceptable accuracy using such sophisticated neuron models. Therefore, the study of leak in the context of such models has not been considered in this chapter.

As regards to studying the consequences of leak, we first focused on the robustness of SNNs to common corruptions, which has been a significant concern in neural networks and motivated a number of recent works. Data augmentation [56], [103], [108] and quantization [109], [110] have been shown to achieve robust performance in both SNNs and ANNs. However, data augmentation-based techniques usually do not generalize well to other types of noise than those used during training, necessitating expensive iterative training efforts using diverse augmented samples. Moreover, the input and weight quantization techniques are reported to be susceptible to error amplification due to enlarged quantization noise in

multi-layered networks [111], leading to considerable loss of accuracy. Our work is pertinent in this respect, since the experimental results show that the leaky neuron models enable improved robustness against random noise, without the need for costly re-training procedures or error amplification. We attribute this enhanced robustness to the better generalization and low-pass filtering effects of the LIF neuron models.

However, introducing the leak in the SNN models (while training with backpropagation) comes at the expense of higher spiking activities compared to the IF models. To that effect, with respect to the usage of leaky neuron models, there is a trade-off between noise robustness and computational efficiency. At τ_m= 100, SNNs with LIF neurons achieve substantially improved robustness compared to the ones with IF neurons while maintaining reasonable spiking sparsity. Training with higher leak (τ_m= 30) further improves the robustness; however, the spiking activities also increase considerably. It would be interesting to further validate our findings on large-scale datasets such as Imagenet [112], however it has remained a challenging problem to train SNNs on Imagenet type of datasets satisfactorily without using ANN-SNN conversion techniques. But in those conversion methods, the learning mainly occurs in the ANN domain and hence the effect of leak in the context of SNN domain learning would not be obvious. Since our concentration is not proposing a new learning paradigm, rather exploring the impact of leak parameter in SNN models, we focus on training the SNNs using spike-based backpropagation from scratch and use relatively smaller datasets.

As we analyze the impacts of leak on robustness and sparsity, we expect our study to be particularly useful for designing resource-constrained edge applications in noisy environments (e.*g.*, self-driving vehicles in adverse weather and rescue robots in disasters etc). Considering that leak is an essential bio-plausible element in SNN models, we believe a better understanding of its effects will help to design improved bio-inspired architectures by making optimal choices concerning the involved trade-offs. Furthermore, an efficient algorithm-hardware co-design considering the leak impacts would be of interest for future research directions. To conclude, the understanding of leak provides another knob for designing SNNs, enabling us to obtain a robustly trained network without sacrificing compute-efficiency significantly.

Our anticipation is that the findings of this work will contribute towards bridging the two seemingly disparate fields of neuroscience and machine learning.

5.6 Appendix: Detailed Formulation of Coherence Function

In this section, we present the derivations of the coherence function, $C(\omega)$ between the input stimulus for the Leaky Integrate and Fire (LIF) neuron and output spike train, guided by [113] and [114]. The discussion is divided into two parts: first we formulate the equation describing the neuron model in subsection 5.6.1, next using this formulated model, we derive equations to calculate coherence in subsection 5.6.2.

5.6.1 LIF Model Equation

The dynamics of an LIF neuron is modeled as follows:

$$\tau_m \frac{dU}{dt} = -(U - U_{rest}) + RI, \quad U \leq V_{th} \tag{A.1}$$

where U is the membrane potential, I denotes the input current, τ_m indicates the membrane time constant, R represents membrane resistance and U_{rest} is the resting potential. Note, an equivalent parallel resistor-capacitor (RC) circuit model of the LIF neuron is illustrated in Fig. 5.1(c) in the main manuscript.

Let us consider the case where the input $I(t)$ to the model described in equation (A.1) is a white Gaussian noise with a constant mean value $\langle I \rangle$ and a correlation function $\langle (I(t) - \langle I \rangle)(I(t) - \langle I \rangle) \rangle = 2D_I\delta(t - t)$ (here, we denote the mean of a parameter H as $\langle H \rangle$). Let us make the following variable changes:

$$v = \frac{U - U_{rest}}{V_{th} - U_{rest}}, t \longrightarrow \frac{t}{\tau_m}. \tag{A.a}$$

When the membrane potential is U, the input current through the resistance branch of the RC circuit model becomes $(U - U_{rest})/R$. We denote the opposite of this input current

as $I_{model}(U) = -(U - U_{rest})/R$. Taking the variable changes from (A.a) into account and differentiating v with respect to time, we get:

$$\frac{dv}{dt} = \dot{v} = \frac{\tau_m}{V_{th} - U_{rest}} \frac{dU}{dt}. \tag{A.b}$$

In addition, using the scaling property of the delta function [99], the correlation function of $I(t)$ becomes $\frac{2D_I}{\tau_m}\delta(t-t)$ (since $\delta(\tau_m t) = \frac{1}{\tau_m}\delta(t)$). Accordingly, we denote input $I(t)$ as follows:

$$I(t) = \langle I \rangle + \sqrt{\frac{2D_I}{\tau_m}}\xi(t), \tag{A.c}$$

where $\xi(t)$ is a zero-mean white Gaussian noise with $\langle \xi(t)\xi(t) \rangle = \delta(t-t)$. Using the relations from (A.b) and (A.c) and dividing both sides by $(V_{th} - U_{rest})$ in equation (A.1), we obtain the following equation:

$$\dot{v} = -\frac{(U - U_{rest})}{V_{th} - U_{rest}} + \frac{R}{V_{th} - U_{rest}}\langle I \rangle + \frac{R}{V_{th} - U_{rest}}\sqrt{\frac{2D_I}{\tau_m}}\xi(t). \tag{A.2}$$

Considering $v = \frac{U - U_{rest}}{V_{th} - U_{rest}}$, we acquire $(V_{th} - U_{rest})v + U_{rest} = U$. Therefore, we can get:

$$I_{model}((V_{th} - U_{rest})v + U_{rest}) = I_{model}(U) = -\frac{U - U_{rest}}{R} \tag{A.3}$$

Based on equation (A.3), the first term on the right-hand side in equation (A.2) can be written as:

$$\begin{aligned}
\frac{R}{V_{th} - U_{rest}} \frac{-(U - U_{rest})}{R} &= \frac{R}{V_{th} - U_{rest}}[I_{model}((V_{th} - U_{rest})v + U_{rest})] \\
&= \frac{R}{V_{th} - U_{rest}}[I_{model}((V_{th} - U_{rest})v + U_{rest}) - I_{model}(U_{rest})] \\
&+ \frac{R}{V_{th} - U_{rest}}[I_{model}(U_{rest})].
\end{aligned}$$

Next, by merging the time-invariant term, $\frac{R}{V_{th} - U_{rest}}[I_{model}(U_{rest})]$ with the $\frac{R}{V_{th} - U_{rest}}\langle I \rangle$ term on the right-hand side in equation (A.2), we can define the $f_{model}(v)$, μ and D as-

$$f_{model}(v) = \frac{R}{V_{th} - U_{rest}}[I_{model}((V_{th} - U_{rest})v + U_{rest}) - I_{model}(U_{rest})], \tag{A.d}$$

$$\mu = \frac{R}{V_{th} - U_{rest}}[\langle I \rangle + I_{model}(U_{rest})], \tag{A.e}$$

and

$$D = \frac{D_I R^2}{\tau_m (V_{th} - U_{rest})^2}. \tag{A.f}$$

Here, μ and D are input parameters that represent the mean and the intensity of the fluctuating input in our model, respectively. Using the definitions from (A.d), (A.e) and (A.f), equation (A.2) can be further written as follows:

$$\dot{v} = f_{model}(v) + \mu + \sqrt{2D}\xi(t). \tag{A.4}$$

Now, $I_{model}(U_{rest}) = -(U_{rest} - U_{rest})/R = 0$. Therefore, from equation (A.d), $f_{model}(v)$ for the LIF model can be transformed as follows:

$$\begin{aligned} f_{LIF} = f_{model}(v) &= \frac{R}{V_{th} - U_{rest}}[I_{model}((V_{th} - U_{rest})v + U_{rest}) - I_{model}(U_{rest})] \\ &= \frac{R}{V_{th} - U_{rest}}[I_{model}((V_{th} - U_{rest})v + U_{rest})]; [\because I_{model}(U_{rest}) = 0] \\ &= \frac{R}{V_{th} - U_{rest}} \frac{-(U - U_{rest})}{R}; [using\ equation\ (3)] \\ &= \frac{-(U - U_{rest})}{V_{th} - U_{rest}} = -v; [from(a)], \end{aligned}$$

Therefore, equation (A.4) becomes as follows:

$$\dot{v} = -v + \mu + \sqrt{2D}\xi(t), \tag{A.5}$$

which is the formalism also used in [113] and will be followed for the remaining discussions in this study.

5.6.2 Coherence Function

Our analysis is based on the parallel RC circuit model of the LIF neuron [97] as depicted by equation (A.1). Here, the membrane capacitance C integrates the input currents over

time and the resistance branch R represents the leakage path of membrane potential. For IF neuron model, since there is no leak path, the R branch is considered as an open circuit. Hence, in this case, $I_{model} = \frac{-(U-U_{rest})}{R} = 0$ and the RC circuit model only contains the capacitor C path. This implies that, for the IF model, R becomes infinity, and correspondingly the membrane time constant τ_m, which is equal to RC, also becomes infinity. On the other hand, for LIF neuron model, the R branch plays a role as the leakage path of membrane potential. When the leakage current through the resistance path increases, the resistance value R and the membrane time constant τ_m gradually decrease. Furthermore, the parameters μ and D ($= \frac{D_I R^2}{\tau_m (V_{th}-U_{rest})^2} = \frac{D_I R}{C(V_{th}-U_{rest})^2}$) become proportional to R according to equation (A.e) and (A.f), respectively. Therefore, for the LIF neuron models, D and μ gradually decrease with the increase in leak amount.

The author in [113] considered $D = D_{bg} + D_{st}$ in equation A.5, where D_{bg} is the background noise intensity, D_{st} is the intensity of the stimulus (Gaussian white noise input) and D is the total noise intensity. For our analysis, by assuming $D_{bg} = 0$, we get $D = D_{st}$ (note, a similar consideration was made in [113] for the results and analysis). Now, let us consider the output spike train of the model described by equation (A.5) is $x(t) = \sum \delta[t - t_k]$, where t_k is the k^{th} instant of spike timing, when the input stimulus (s) is Gaussian white noise input. We quantify the information transmission of the spiking model by means of the spectral coherence function. To that end, the Fourier transform of $x(t)$ in a time window $[0, T]$ becomes as follows: $\tilde{x}_T(\omega) = \int_0^T x(t) \mathrm{e}^{\mathrm{j}\omega t} dt$. The cross-spectrum of output spike train (x) and input stimulus (s) is given as [99]: $S_{x,s}(\omega) = \lim_{T\to\infty} \frac{\langle \tilde{x}(\omega)\tilde{s}^*(\omega)\rangle}{T}$, and the spike train power spectrum is defined as: $S_{x,x}(\omega) = \lim_{T\to\infty} \frac{\langle \tilde{x}(\omega)\tilde{x}^*(\omega)\rangle}{T}$. The coherence function is formally defined as the squared correlation coefficient between the input and output as follows:

$$C_{x,s}(\omega) = \frac{|S_{x,s}(\omega)|^2}{S_{x,x}(\omega) S_{s,s}(\omega)}. \tag{A.6}$$

The coherence function $C_{x,s}(\omega)$ generates an output number between 0 and 1 at each measurement frequency. The amount of information transmission at each frequency is proportional to the coherence at that particular frequency, with 1 and 0 denoting full and null

transmission, respectively. For a system acting as a low-pass filter, the coherence output under white-noise stimulation decreases in the high-frequency domain.

Next, we analyze the low-pass filtering effect of the LIF neuron model as described by equation (A.5). The analytical expression for $S_{x,s}(\omega)$ is given as follows [113], [115], [116]:

$$S_{x,s}(\omega) = \frac{2D_{st}}{\sqrt{D}} \frac{r_0 \mathrm{i}\omega}{\mathrm{i}\omega - 1} \frac{\mathscr{D}_{\mathrm{i}\omega-1}\left(\frac{\mu-V_{th}}{\sqrt{D}}\right) - \mathrm{e}^{\Delta}\mathscr{D}_{\mathrm{i}\omega-1}\left(\frac{\mu-U_{rest}}{\sqrt{D}}\right)}{\mathscr{D}_{\mathrm{i}\omega}\left(\frac{\mu-V_{th}}{\sqrt{D}}\right) - \mathrm{e}^{\Delta}\mathrm{e}^{\mathrm{i}\omega\tau_r}\mathscr{D}_{\mathrm{i}\omega}\left(\frac{\mu-U_{rest}}{\sqrt{D}}\right)}, \tag{A.7}$$

where $\Delta = \frac{U_{rest}^2 - V_{th}^2 + 2\mu(V_{th}-U_{rest})}{4D}$, τ_r is the refractory period and $\mathscr{D}(x)$ is the parabolic cylinder function. In our case, we follow the same assumptions as in [113] where $U_{rest} = 0$, $\tau_r = 0$ and $V_{th} = 1$. The firing rate r_0 is given by calculating the following [113]:

$$r_0 = \left[\tau_r + \sqrt{\pi}\int_{\frac{\mu-V_{th}}{\sqrt{2D}}}^{\frac{\mu-U_{rest}}{\sqrt{2D}}} dz \mathrm{e}^{z^2} erfc(z)\right]^{-1}.$$

The power spectrum of the output spike train is given by [117], calculated as follows:

$$S_{x,x}(\omega) = r_0 \frac{\left|\mathscr{D}_{\mathrm{i}\omega}\left(\frac{\mu-V_{th}}{\sqrt{D}}\right)\right|^2 - \mathrm{e}^{2\Delta}\left|\mathscr{D}_{\mathrm{i}\omega}\left(\frac{\mu-U_{rest}}{\sqrt{D}}\right)\right|^2}{\left|\mathscr{D}_{\mathrm{i}\omega}\left(\frac{\mu-V_{th}}{\sqrt{D}}\right) - \mathrm{e}^{\Delta}\mathrm{e}^{\mathrm{i}\omega\tau_r}\mathscr{D}_{\mathrm{i}\omega}\left(\frac{\mu-U_{rest}}{\sqrt{D}}\right)\right|^2}, \tag{A.8}$$

and the noise input spectrum becomes [113]:

$$S_{s,s}(\omega) = 2D_{st}. \tag{A.9}$$

Taking the magnitude square of the quantity in equation (A.7), we derive the following,

$$|S_{x,s}(\omega)|^2 = \frac{4D_{st}^2}{D} \frac{r_0^2\omega^2}{1+\omega^2} \frac{|\mathscr{D}_{\mathrm{i}\omega-1}\left(\frac{\mu-V_{th}}{\sqrt{D}}\right) - \mathrm{e}^{\Delta}\mathscr{D}_{\mathrm{i}\omega-1}\left(\frac{\mu-U_{rest}}{\sqrt{D}}\right)|^2}{|\mathscr{D}_{\mathrm{i}\omega}\left(\frac{\mu-V_{th}}{\sqrt{D}}\right) - \mathrm{e}^{\Delta}\mathrm{e}^{\mathrm{i}\omega\tau_r}\mathscr{D}_{\mathrm{i}\omega}\left(\frac{\mu-U_{rest}}{\sqrt{D}}\right)|^2}. \tag{A.10}$$

Finally, plugging the values of $S_{x,x}(\omega)$, $S_{s,s}(\omega)$ and $|S_{x,s}(\omega)|^2$ into equation (A.6), we obtain the resultant coherence function as follows:

$$C_{x,s}(\omega) = \frac{2D_{st}}{D} \frac{r_0\omega^2}{1+\omega^2} \frac{\left|\mathscr{D}_{\mathrm{i}\omega-1}\left(\frac{\mu-V_{th}}{\sqrt{D}}\right) - \mathrm{e}^{\Delta}\mathscr{D}_{\mathrm{i}\omega-1}\left(\frac{\mu-U_{rest}}{\sqrt{D}}\right)\right|^2}{\left|\mathscr{D}_{\mathrm{i}\omega}\left(\frac{\mu-V_{th}}{\sqrt{D}}\right)\right|^2 - \mathrm{e}^{2\Delta}\left|\mathscr{D}_{\mathrm{i}\omega}\left(\frac{\mu-U_{rest}}{\sqrt{D}}\right)\right|^2}. \tag{A.11}$$

6. SPIKE-FLOWNET: EVENT-BASED OPTICAL FLOW ESTIMATION WITH ENERGY-EFFICIENT HYBRID NEURAL NETWORKS

6.1 Introduction

The dynamics of biological species such as winged insects serve as prime sources of inspiration for researchers in the field of neuroscience, machine learning as well as robotics. The ability of winged insects to perform complex, high-speed maneuvers effortlessly in cluttered environments clearly highlights the efficiency of these resource-constrained biological systems [118]. The estimation of motion patterns corresponding to spatio-temporal variations of structured illumination - commonly referred to as optical flow, provides vital information for estimating ego-motion and perceiving the environment. Modern deep Analog Neural Networks (ANNs) aim to achieve this at the cost of being computationally intensive, placing significant overheads on current hardware platforms. A competent methodology to replicate such energy efficient biological systems would greatly benefit edge-devices with computational and memory constraints

Over the past years, the majority of optical flow estimation techniques relied on images from traditional frame-based cameras, where the input data is obtained by sampling intensities on the entire frame at fixed time intervals irrespective of the scene dynamics. Although sufficient for certain computer vision applications, frame-based cameras suffer from issues such as motion blur during high speed motion, inability to capture information in low-light conditions, and over- or under-saturation in high dynamic range environments.

Event-based cameras, often referred to as bio-inspired silicon retinas, overcome these challenges by detecting log-scale brightness changes asynchronously and independently on each pixel-array element [20], similar to retinal ganglion cells. Having a high temporal resolution (in the order of microseconds) and a fraction of power consumption compared to frame-based cameras make event cameras suitable for estimating high-speed and low-light visual motion in an energy-efficient manner. However, because of their fundamentally different working principle, conventional computer vision as well as ANN-based methods become

no longer effective for event camera outputs. This is mainly because these methods are typically designed for pixel-based images relying on photo-consistency constraints, assuming the color and brightness of object remain the same in all image sequences. Thus, the need for development of handcrafted-algorithms for handling event camera outputs is paramount.

SNNs can naturally encapsulate the event-based asynchronous processing capability across layers, leading to energy-efficient computing on specialized neuromorphic hardware such as IBM's TrueNorth [8] and Intel's Loihi [119]. However, recent works have shown that the number of spikes drastically vanish at deeper layers, leading to performance degradations in deep SNNs [120]. Thus, there is a need for an efficient hybrid architecture, with SNNs in the initial layers, to exploit their compatability with event camera outputs while having ANNs in the deeper layers in order to retain performance.

In regard to this, we propose a deep hybrid neural network architecture, accommodating SNNs and ANNs in different layers, for energy efficient optical flow estimation using sparse event camera data. To the best of our knowledge, this is the first SNN demonstration to report the state-of-art performance on event-based optical flow estimation, outperforming its corresponding fully-fledged ANN counterpart.

The main contributions of this work can be summarized as:

- We present an input representation that efficiently encodes the sequences of sparse outputs from event cameras over time to preserve the spatio-temporal nature of spike events.
- We introduce a deep hybrid architecture for event-based optical flow estimation referred to as Spike-FlowNet, integrating SNNs and ANNs in different layers, to efficiently process the sparse spatio-temporal event inputs.
- We evaluate the optical flow prediction capability and computational efficiency of Spike-FlowNet on the Multi-Vehicle Stereo Event Camera dataset (MVSEC) [121] and provide comparison results with current state-of-the-art approaches.

The following contents are structured as follows. In Section 2, we elucidate the related works. In Section 3, we present the methodology, covering essential backgrounds on the spiking neuron model followed by our proposed input event (spike) representation. This sec-

tion also discusses the self-supervised loss, Spike-FlowNet architecture, and the approximate backpropagation algorithm used for training. Section 4 covers the experimental results, including training details and evaluation metrics. It also discusses the comparison results with the latest works in terms of performance and computational efficiency.

6.2 Related Work

In recent years, there have been an increasing number of works on estimating optical flow by exploiting the high temporal resolution of event cameras. In general, these approaches have either been adaptations of conventional computer vision methods or modified versions of deep ANNs to encompass discrete outputs from event cameras.

For computer vision based solutions to estimate optical flow, gradient-based approaches using the Lucas-Kanade algorithm [122] have been highlighted in [123], [124]. Further, plane fitting approaches by computing the slope of the plane for estimating optical flow have been presented in [125], [126]. In addition, bio-inspired frequency-based approaches have been discussed in [127]. Finally, correlation-based approaches are presented in [128], [129] employing convex optimization over events. In addition, [130] interestingly uses an adaptive block matching technique to estimate sparse optical flow.

For deep ANN-based solutions, optical flow estimation from frame-based images has been discussed in Unflow [131], which utilizes a U-Net [132] architecture and computes a bidirectional census loss in an unsupervised manner with an added smoothness term. This strategy is modified for event camera outputs in EV-FlowNet [133] incorporating a self-supervised loss based on gray images as a replacement for ground truth. Other previous works employ various modifications to the training methodology, such as [134], which imposes certain brightness constancy and smoothness constraints to train a network and [135] which adds an adversarial loss over the standard photometric loss. In contrast, [136] presents an unsupervised learning approach using only event camera data to estimate optical flow by accounting for and then learning to rectify the motion blur.

All the above strategies employ ANN architectures to predict the optical flow. However, event cameras produce asynchronous and discrete outputs over time, and SNNs can naturally

capture their spatio-temporal dynamics, which are embedded in the precise spike timings. Hence, we posit that SNNs are suitable for handling event camera outputs. Recent SNN-based approaches for event-based optical flow estimation include [137]–[139]. Researchers in [137] presented visual motion estimation using SNNs, which accounts for synaptic delays in generating motion-sensitive receptive fields. In addition, [138] demonstrated real-time model-based optical flow computations on TrueNorth hardware for evaluating patterns including rotating spirals and pipes. Authors of [139] presented a methodology for optical flow estimation using convolutional SNNs based on Spike-Time-Dependent-Plasticity (STDP) learning [13]. The main limitation of these works is that they employ shallow SNN architectures, because deep SNNs suffer in terms of performance. Besides, the presented results are only evaluated on relatively simple tasks. In practice, they do not generally scale well to complex and real-world data, such as that presented in MVSEC dataset [121]. In view of these, a hybrid approach becomes an attractive option for constructing deep network architectures, leveraging the benefits of both SNNs and ANNs.

6.3 Method

6.3.1 Spiking Input Event Representation

An event-based camera tracks the changes in log-scale intensity (I) at every element in the pixel-array independently and generates a discrete event whenever the change exceeds a threshold (θ):

$$\| \log(I_{t+1}) - \log(I_t) \| \geq \theta \tag{6.1}$$

A discrete event contains a 4-tuple $\{x, y, t, p\}$, consisting of the coordinates: x, y; timestamp: t; and polarity (direction) of brightness change: p. This input representation is called Address Event Representation (AER), and is the standard format used by event-based sensors.

There are prior works that have modified the representations of asynchronous event camera outputs to be compatible with ANN-based methods. To overcome the asynchronous nature, event outputs are typically recorded for a certain time period and transformed into a synchronous image-like representation. In EV-FlowNet [133], the most recent pixel-wise timestamps and the event counts encoded the motion information (within a time window) in

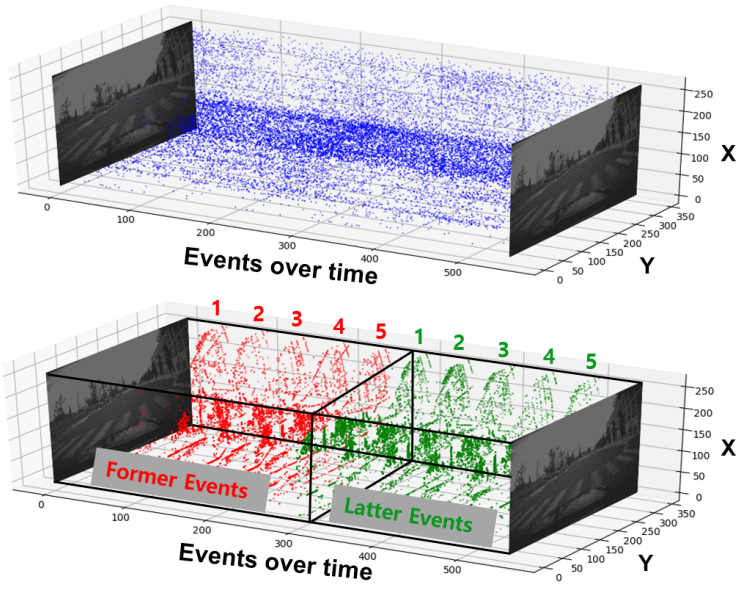


Figure 6.1. Input event representation. (*Top*) Continuous raw events between two consecutive grayscale images from an event camera. (*Bottom*) Accumulated event frames between two consecutive grayscale images to form the former and the latter event groups, serving as inputs to the network.

an image. However, fast motions and dense events (in local regions of the image) can vastly overlap per-pixel timestamp information, and temporal information can be lost. In addition, [136] proposed a discretized event volume that deals with the time domain as a channel to retain the spatio-temporal event distributions. However, the number of input channels increases significantly as the time dimensions are finely discretized, further aggravating the computation and parameter overheads.

In this chapter, we propose a discretized input representation (fine-grained in time) that preserves the spatial and temporal information of events for SNNs. Our proposed input encoding scheme discretizes the time dimension within a time window into two groups (former and latter). Each group contains N number of event frames obtained by accumulating raw events from the timestamp of the previous frame till the current timestamp. Each of these event frames is also composed of two channels for ON/OFF polarity of events. Hence, the input to the network consists of a sequence of N frames with four channels (one frame each from the former and the latter groups having two channels each). The proposed

input representation is displayed in Fig. 6.1 for one channel (assuming the number of event frames in each group equals to five). The main characteristic of our proposed input event representation (compared to ANN-based methods) are as follows:

- Our spatio-temporal input representations encode only the presence of events over time, allowing asynchronous and event-based computations in SNNs. In contrast, ANN-based input representation often requires the timestamp and the event count images in separate channels.
- In Spike-FlowNet, each event frame from the former and the latter groups sequentially passes through the network, thereby preserving and utilizing the spatial and temporal information over time. On the contrary, ANN-based methods feed-forward all input information to the network at once.

6.3.2 Self-Supervised Loss

The DAVIS camera [140] is a commercially available event-camera, which simultaneously provides synchronous grayscale images and asynchronous event streams. The number of available event-based camera datasets with annotated labels suitable for optical flow estimation is quite small, as compared to frame-based camera datasets. Hence, a self-supervised learning method that uses proxy labels from the recorded grayscale images [133], [134] is employed for training our Spike-FlowNet.

The overall loss incorporates a photometric reconstruction loss ($\mathcal{L}_{\text{photo}}$) and a smoothness loss ($\mathcal{L}_{\text{smooth}}$) [134]. To evaluate the photometric loss within each time window, the network is provided with the former and the latter event groups and a pair of grayscale images, taken at the start and the end of the event time window (I_t, I_{t+dt}). The predicted optical flow from the network is used to warp the second grayscale image to the first grayscale image. The photometric loss ($\mathcal{L}_{\text{photo}}$) aims to minimize the discrepancy between the first grayscale image and the inverse warped second grayscale image. This loss uses the photo-consistency

assumption that a pixel in the first image remains similar in the second frame mapped by the predicted optical flow. The photometric loss is computed as follows:

$$\mathcal{L}_{\text{photo}}(u, v; I_t, I_{t+dt}) = \sum_{x,y} \rho(I_t(x, y) - I_{t+dt}(x + u(x, y), y + v(x, y))) \quad (6.2)$$

where, I_t, I_{t+dt} indicate the pixel intensity of the first and second grayscale images, u, v are the flow estimates in the horizontal and vertical directions, ρ is the Charbonnier loss $\rho(x) = (x^2 + \eta^2)^r$, which is a generic loss used for outlier rejection in optical flow estimation [141]. For this work, $r = 0.45$ and η =1e-3 show the optimum results for the computation of photometric loss.

Furthermore, a smoothness loss ($\mathcal{L}_{\text{smooth}}$) is applied for enhancing the spatial collinearity of neighboring optical flow. The smoothness loss minimizes the difference in optical flow between neighboring pixels and acts as a regularizer on the predicted flow. It is computed as follows:

$$\mathcal{L}_{\text{smooth}}(u, v) = \frac{1}{HD} \sum_{\text{j}}^{H} \sum_{\text{i}}^{D} (\|u_{\text{i,j}} - u_{\text{i+1,j}}\| + \|u_{\text{i,j}} - u_{\text{i,j+1}}\| + \|v_{\text{i,j}} - v_{\text{i+1,j}}\| + \|v_{\text{i,j}} - v_{\text{i,j+1}}\|) \quad (6.3)$$

where H is the height and D is the width of the predicted flow output. The overall loss is computed as the weighted sum of the photometric and smoothness loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{photo}} + \lambda \mathcal{L}_{\text{smooth}} \quad (6.4)$$

where λ is the weight factor.

6.3.3 Spike-FlowNet Architecture

Spike-FlowNet employs a deep hybrid architecture that accommodates SNNs and ANNs in different layers, enabling the benefits of SNNs for sparse event data processing and ANNs for maintaining the performance. The use of a hybrid architecture is attributed to the fact that spike activities reduce drastically with growing the network depth in the case of full-

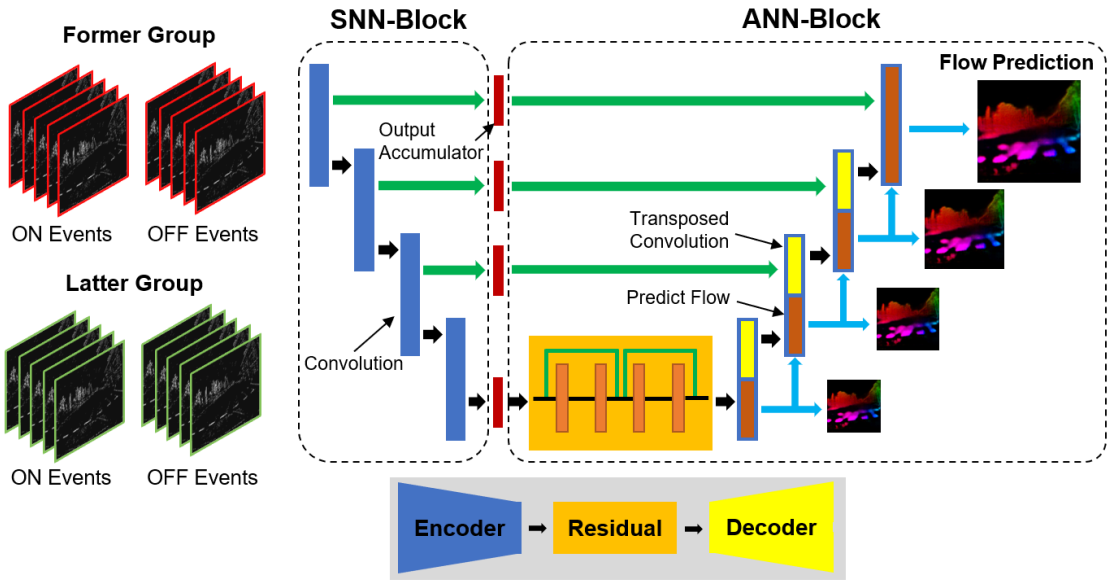


Figure 6.2. Spike-FlowNet architecture. The four-channeled input images, comprised of ON/OFF polarity events for former and latter groups, are sequentially passed through the hybrid network. The SNN-block contains the encoder layers followed by output accumulators, while the ANN-block contains the residual and decoder layers. The loss is evaluated after forward propagating all consecutive input event frames (a total of N inputs, sequentially taken in time from the former and the latter event groups) within the time window. The black arrows denote the forward path, green arrows represent residual connections, and blue arrows indicate the flow predictions.

fledged SNNs. This is commonly referred to as the vanishing spike phenomenon [142], and potentially leads to performance degradation in deep SNNs. Furthermore, high numerical precision is essentially required for estimating the accurate pixel-wise network outputs, namely the regression tasks. Hence, very rare and binary precision spike signals (in input and intermediate layers) pose a crucial issue for predicting the accurate flow displacements. To resolve these issues, only the encoder block is built as an SNN, while the residual and decoder blocks maintain an ANN architecture.

Spike-FlowNet's network topology resembles the U-Net [132] architecture, containing four encoder layers, two residual blocks, and four decoder layers as shown in Fig. 6.2. The events are represented as the four-channeled input frames as presented in Section 6.3.1, and are sequentially passed through the SNN-based encoder layers over time (while being

downsampled at each layer). Convolutions with a stride of two are employed for incorporating the functionality of dimensionality reduction in the encoder layers. The outputs from encoder layers are collected in their corresponding output accumulators until all consecutive event images have passed. Next, the accumulated outputs from final encoder layer are passed through two residual blocks and four decoder layers. The decoder layers upsample the activations using transposed convolution. At each decoder layer, there is a skip connection from the corresponding encoder layer, as well as another convolution layer to produce an intermediate flow prediction, which is concatenated with the activations from the transposed convolutions. The total loss is evaluated after the forward propagation of all consecutive input event frames through the network and is applied to each of the intermediate dense optical flows using the grayscale images.

Algorithm 4 Backpropagation Training in Spike-FlowNet for an Iteration

Input: Event-based inputs ($inputs$), total number of discrete time-steps (N), number of SNN/ANN layers (L_S/L_A), SNN/ANN outputs (o/o_A) membrane potential (V), firing threshold (V_{th}), ANN nonlinearity (f)
Initialize: $V^l[n] = 0,\ \forall l = 1, ..., L_S$
// ***Forward Phase in SNN-blocks***
for $n \leftarrow 1$ **to** N **do**
    $o^1[n] = inputs[n]$
    for $l \leftarrow 2$ **to** $L_S - 1$ **do**
        $V^l[n] = V^l[n-1] + w^l o^{l-1}[n]$ //*weighted spike-inputs are integrated to V*
        if $V^l[n] > V_{th}$ **then**
            $o^l[n] = 1,\ V^l[n] = 0$ //*if V exceeds V_{th}, a neuron emits a spike and reset V*
    $o_A^{L_S} = V^{L_S}[n] = V^{L_S}[n-1] + w^{L_S} o^{L_S-1}[n]$ //*final SNN layer does not fire*
// ***Forward Phase in ANN-blocks***
for $l \leftarrow L_S + 1$ **to** $L_S + L_A$ **do**
    $o_A^l = f(w^l o_A^{l-1})$
// ***Backward Phase in ANN-blocks***
for $l \leftarrow L_S + L_A$ **to** L_S **do**
    $\triangle w^l = \frac{\partial \mathcal{L}_{total}}{\partial o_A^l} \frac{\partial o_A^l}{\partial w^l}$
// ***Backward Phase in SNN-blocks***
for $n \leftarrow N$ **to** 1 **do**
    for $l \leftarrow L_S - 1$ **to** 1 **do**
        //*evaluate partial derivatives of loss w.r.t. w_S by unrolling the SNN over time*
        $\triangle w^l[n] = \frac{\partial \mathcal{L}_{total}}{\partial o^l[n]} \frac{\partial o^l[n]}{\partial V^l[n]} \frac{\partial V^l[n]}{\partial w^l[n]}$

6.3.4 Backpropagation Training in Spike-FlowNet

The spike generation function of an IF neuron is a hard threshold function that emits a spike when the membrane potential exceeds a firing threshold. Due to this discontinuous and non-differentiable neuron model, standard backpropagation algorithms cannot be applied to SNNs in their native form. Hence, several approximate methods have been proposed to estimate the surrogate gradient of spike generation function. In this chapter, we adopt the approximate gradient method proposed in [16], [120] for back-propagating errors through SNN layers. The approximate IF gradient is computed as $\frac{1}{V_{th}}$, where the threshold value accounts for the change of the spiking output with respect to the input. Algorithm 4 illustrates the forward and backward pass in ANN-block and SNN-block.

In the forward phase, neurons in the SNN layers accumulate the weighted sum of the spike inputs in membrane potential. If the membrane potential exceeds a threshold, a neuron emits a spike at its output and resets. The final SNN layer neurons just integrate the weighted sum of spike inputs in the output accumulator, while not producing any spikes at the output. At the last time-step, the integrated outputs of SNN layers propagate to the ANN layers to predict the optical flow. After the forward pass, the final loss ($\mathcal{L}_{total}$) is evaluated, followed by backpropagation of gradients through the ANN layers using standard backpropagation.

Next, the backpropagated errors ($\frac{\partial \mathcal{L}_{total}}{\partial o^L_S}$) pass through the SNN layers using the approximate IF gradient method and BackPropagation Through Time (BPTT) [71]. In BPTT, the network is unrolled for all discrete time-steps, and the weight update is computed as the sum of gradients from each time-step. This procedure is displayed in Fig. 6.3 where the final loss is back-propagated through an ANN-block and a simple SNN-block consisting of a single input IF neuron. The parameter updates of the l^{th} SNN layers are described as follows:

$$\triangle w^l = \sum_n \frac{\partial \mathcal{L}_{total}}{\partial o^l[n]} \frac{\partial o^l[n]}{\partial V^l[n]} \frac{\partial V^l[n]}{\partial w^l}, \text{ where } \frac{\partial o^l[n]}{\partial V^l[n]} = \frac{1}{V_{th}}(o^l[n] > 0) \tag{6.5}$$

where o^l represents the output of spike generation function. This method enables the end-to-end self-supervised training in the proposed hybrid architecture.

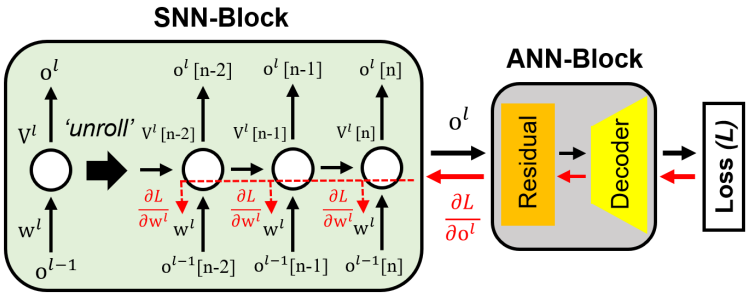


Figure 6.3. Error backpropagation in Spike-FlowNet. After the forward pass, the gradients are back-propagated through the ANN block using standard backpropagation whereas the backpropagated errors pass through the SNN layers using the approximate IF gradient method and BPTT technique.

6.4 Experimental Results

6.4.1 Dataset and Training Details

We use the MVSEC dataset [121] for training and evaluating the optical flow predictions. MVSEC contains stereo event-based camera data for a variety of environments (e.g., indoor flying and outdoor driving) and also provides the corresponding ground truth optical flow. In particular, the indoor and outdoor sequences are recorded in dissimilar environments where the indoor sequences (indoor_flying) have been captured in a lab environment and the outdoor sequences (outdoor_day) have been recorded while driving on public roads.

Even though the indoor_flying and outdoor_day scenes are quite different, we only use outdoor_day2 sequence for training Spike-FlowNet. This is done to provide fair comparisons with prior works [133], [136] which utilized only outdoor_day2 sequence for training. During training, input images are randomly flipped horizontally and vertically (with 0.5 probability) and randomly cropped to 256×256 size. Adam optimizer [51] is used, with the initial learning rate of 5e-5, and scaled by 0.7 every 5 epochs until 10 epoch, and every 10 epochs thereafter. The model is trained on the left event camera data of outdoor_day2 sequence for 100 epochs with a mini-batch size 8. Training is done for two different time windows lengths (i.e, 1 grayscale image frame apart ($dt = 1$) and 4 grayscale image frames apart ($dt = 4$)). The number of event frame (N) and weight factor for the smoothness loss (λ) are set to 5,

10 for a $dt = 1$ case and 20, 1 for a $dt = 4$ case, respectively. The threshold of the IF neurons are set to 0.5 ($dt = 4$) and 0.75 ($dt = 1$) in SNN layers.

6.4.2 Algorithm Evaluation Metric

The evaluation metric for optical flow prediction is the Average End-point Error (AEE), which represents the mean distance between the predicted flow (y_{pred}) and the ground truth flow (y_{gt}). It is given by:

$$\text{AEE} = \frac{1}{m} \sum_{\text{m}} \|(u, v)_{\text{pred}} - (u, v)_{\text{gt}}\|_2 \tag{6.6}$$

where m is the number of active pixels in the input images. Because of the highly sparse nature of input events, the optical flows are only estimated at pixels where both the events and ground truth data is present. We compute the AEE for $dt = 1$ and $dt = 4$ cases.

6.4.3 Average End-point Error (AEE) Results

During testing, optical flow is estimated on the center cropped 256 × 256 left camera images of the indoor_flying 1,2,3 and outdoor_day 1 sequences. We use all events for the indoor_flying sequences, but we take events within 800 grayscale frames for the outdoor_day1 sequence, similar to [133]. Table 6.1 provides the AEE evaluation results in comparison with the prior event camera based optical flow estimation works. Overall, our results show that Spike-FlowNet can accurately predict the optical flow in both the indoor_flying and out-

Table 6.1. Average Endpoint Error (AEE) comparisons with Zhu et al. [136] and EV-FlowNet [133].

	dt=1 frame				dt=4 frame			
	indoor1	indoor2	indoor3	outdoor1	indoor1	indoor2	indoor3	outdoor1
Zhu et al. [136]	**0.58**	**1.02**	**0.87**	**0.32**	**2.18**	3.85	**3.18**	1.30
EV-FlowNet [133]	1.03	1.72	1.53	0.49	2.25	4.05	3.45	1.23
This work	0.84	1.28	1.11	0.49	2.24	**3.83**	**3.18**	**1.09**

* EV-FlowNet also uses a self-supervised learning method, providing the the fair comparison baseline compared to Spike-FlowNet.

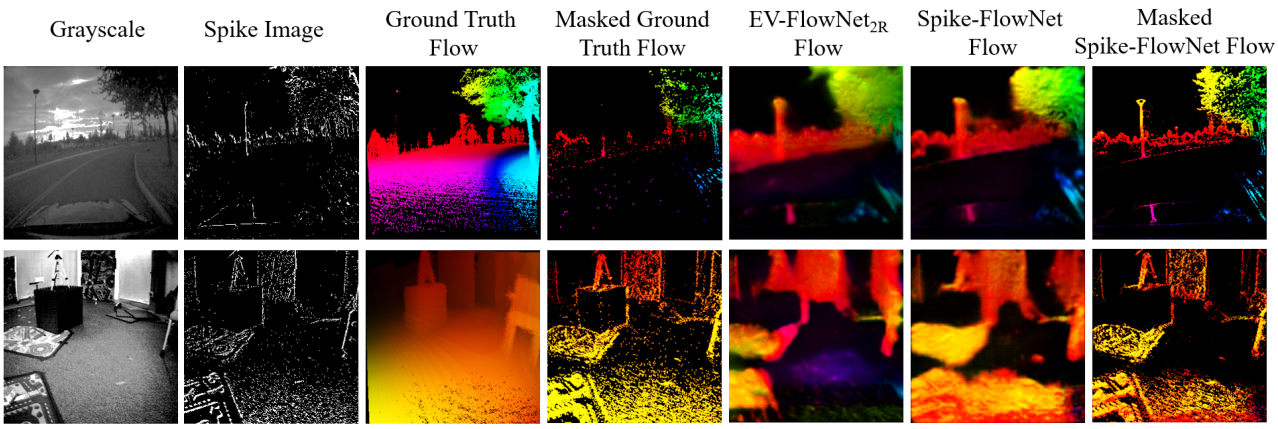


Figure 6.4. Optical flow evaluation and comparison with EV-FlowNet. The samples are taken from (*top*) *outdoor_day*1 and (*bottom*) i*ndoor_day*1. The Masked Spike-FlowNet Flow is basically a sparse optical flow computed at pixels at which events occurred. It is computed by masking the predicted optical flow with the spike image.

door_day1 sequences. This demonstrates that the proposed Spike-FlowNet can generalize well to distinctly different environments. The grayscale, spike event, ground truth flow and the corresponding predicted flow images are visualized in Fig. 6.4 where the images are taken from (*top*) *outdoor_day*1 and (*bottom*) i*ndoor_day*1, respectively. Since event cameras work based on changing light intensity at pixels, the regions having low texture produce very sparse events due to minimal intensity changes, resulting in scarce optical flow predictions in the corresponding areas such as the flat surfaces. Practically, the useful flows are extracted by using flow estimations at points where significant events exist in the input frames.

Moreover, we compare our quantitative results with the recent works [133], [136] on event-based optical flow estimation, as listed in Table 6.1. We observe that Spike-FlowNet outperforms EV-FlowNet [133] in terms of AEE results in both the $dt = 1$ and $dt = 4$ cases. It is worth noting here that EV-FlowNet employs a similar network architecture and self-supervised learning method, providing a fair comparison baseline for fully ANN architectures. In addition, Spike-FlowNet attains AEE results slightly better or comparable to [136] in the $dt = 4$ case, while underperforming in the $dt = 1$ case. The authors in [136]

presented an image deblurring based unsupervised learning that employed only the event streams. Hence, it seems to not suffer from the issues related to grayscale images such as motion blur or aperture problems during training. In view of these comparisons, Spike-FlowNet (with presented spatio-temporal event representation) is more suitable for motion detection when the input events have a certain minimum level of spike density.

6.4.4 Ablation Study

Next, we present the ablation studies to explore the optimal design choices of hybrid networks, input data representation and weight factor (λ) of the smoothness loss in the loss function.

Hybrid Network

In addition to the described architecture (denoted Spike-FlowNet), we train additional network topologies to test different hybrid design options. We use two more networks in which residual blocks are composed of SNN layers: one where only first residual block is converted to SNN (Spike-FlowNet_1R), and second where both residual blocks are converted to SNN (Spike-FlowNet_2R). Note, results for a fully ANN architecture are given in EV-FlowNet [133]. We do not consider converting the decoder layers to construct a fully SNN architecture, as they use analog inputs from intermediate optical flows and output accumulators.

Table 6.2. Analysis for Spike-FlowNet in terms of the mean spike activity, the total and normalized number of SNN operations in an encoder-block, the encoder-block and overall computational energy benefits.

	indoor1		indoor2		indoor3		outdoor1	
	dt=1	dt=4	dt=1	dt=4	dt=1	dt=4	dt=1	dt=4
Encoder Spike Activity (%)	**0.33**	0.87	0.65	1.27	0.53	1.11	0.41	0.78
Encoder SNN # Operation ($\times 10^8$)	**0.16**	1.69	0.32	2.47	0.26	2.15	0.21	1.53
Encoder Normalized # Operation (%)	**1.68**	17.87	3.49	26.21	2.81	22.78	2.29	16.23
Encoder Compute-energy Benefit ($\times$)	**305**	28.6	146.5	19.5	182.1	22.44	223.2	31.5
Overall Compute-energy Reduction (%)	**17.57**	17.01	17.51	16.72	17.53	16.84	17.55	17.07

* For an ANN, the number of synaptic operations is 9.44×10^8 for the encoder-block and 5.35×10^9 for overall network.

Rows 1-3 in Table 6.3 show the AEE results for the different network topologies. We find that AEE results degrade as more layers are transferred to SNNs for both $dt = 1$ and $dt = 4$. This is because the spike vanishing phenomenon aggravates with the network depth, leading to the degradation in the quality of predicted optical flow. The best AEE results are achieved by Spike-FlowNet case which is advocated throughout the manuscript.

Input Representation

We validate the influence of the number of groups (N) in input representation. In the case of $N = 3$ and $N = 4$, AEE results are provided in rows 4-5 in Table 6.3. Note, Spike-FlowNet represents $N = 2$ case. With the increase in the number of input groups (N), the results show that $dt = 1$ case achieves worse AEE while $dt = 4$ converges to a reasonably accurate flow estimate. This is because each input group requires to have a certain number of events for proper training, and we find that $N = 2$ case provides optimal results for both $dt = 1$ and $dt = 4$.

Table 6.3. Average Endpoint Error (AEE) for ablation studies with different design choices

	dt=1 frame				dt=4 frame			
	indoor1	indoor2	indoor3	outdoor1	indoor1	indoor2	indoor3	outdoor1
Spike-FlowNet	0.84	1.28	1.11	0.49	2.24	3.83	3.18	1.09
Spike-FlowNet_1R	0.88	1.55	1.31	0.51	2.73	4.46	3.66	1.15
Spike-FlowNet_2R	0.90	1.56	1.29	0.56	2.75	4.61	3.76	1.19
N=3	0.92	1.34	1.18	0.50	2.34	4.05	3.29	1.12
N=4	1.07	1.76	1.57	0.60	2.27	3.81	3.10	1.15
λ=1	0.91	1.38	1.23	0.50	2.24	3.83	3.18	1.09
λ=10	0.84	1.28	1.11	0.49	2.42	4.22	3.44	1.18
λ=100	0.84	1.30	1.14	0.49	2.50	4.01	3.28	1.19

Loss Function

To find the optimal ratio between photometric and smoothness losses, we train networks with a variety of weight factors (λ) over the range $[1, 100]$. Rows 6-8 in Table 6.3 highlight AEE results for $\lambda = 1$, 10, 100. We observe that $\lambda = 10$, 100 cases converge to more accurate flow estimate for $dt = 1$ while $\lambda = 1$ case works better for $dt = 4$. This is because inputs

are greatly sparse in $dt = 1$, hence its corresponding flow outputs have more scarce and discontinuous structures, requiring a higher degree of smoothness.

6.4.5 Computational Efficiency

To further analyze the benefits of Spike-FlowNet, we estimate the gain in computational costs compared to a fully ANN architecture. Typically, the number of synaptic operations is used as a metric for benchmarking the computational energy of neuromorphic hardware [8], [12], [120]. Also, the required energy consumption per synaptic operation needs to be considered. Now, we describe the procedures for measuring the computational costs in SNN and ANN layers.

In a neuromorphic hardware, SNNs carry out event-based computations only at the arrival of input spikes. Hence, we first measure the mean spike activities at each time-step in the SNN layers. As presented in the first row of Table 6.2, the mean spiking activities (averaged over indoor1,2,3 and outdoor1 sequences) are 0.48% and 1.01% for $dt = 1$ and $dt = 4$ cases, respectively. Note that the neuronal threshold is set to a higher value in $dt = 1$ case; hence the average spiking activity becomes sparser compared to $dt = 4$ case. The extremely rare mean input spiking activities are mainly due to the fact that event camera outputs are highly sparse in nature. This sparse firing rate is essential for exploiting efficient event-based computations in SNN layers. In contrast, ANNs execute dense matrix-vector multiplication operations without considering the sparsity of inputs. In other words, ANNs simply feed-forward the inputs at once, and the total number of operations are fixed. This leads to the high energy requirements (compared to SNNs) by computing both zero and non-zero entities, especially when inputs are very sparse.

Essentially, SNNs need to compute the spatio-temporal spike images over a number of time-steps. Given M is the number of neurons, C is number of synaptic connections and F indicates the mean firing activity, the number of synaptic operations at each time-step in the l^{th} layer is calculated as $M_l \times C_l \times F_l$. The total number of SNN operations is the summation of synaptic operations in SNN layers during the N time-steps. Hence, the total number of SNN and ANN operations become $\sum_l (M_l \times C_l \times F_l) \times N$ and $\sum_l M_l \times C_l$, respectively.

Based on these, we estimate and compare the average number of synaptic operations on Spike-FlowNet and a fully ANN architecture. The total and the normalized number of SNN operations compared to ANN operations on the encoder-block are provided in the second and the third row of Table 6.2, respectively.

Due to the binary nature of spike events, SNNs perform only accumulation (AC) per synaptic operation. On the other hand, ANNs perform the multiply-accumulate (MAC) computations since the inputs consist of analog-valued entities. In general, AC computation is considered to be significantly more energy-efficient than MAC. For example, AC is reported to be $\sim 5.1\times$ more energy-efficient than a MAC in the case of 32-bit floating-point numbers (45nm CMOS process) [143]. Based on this principle, the computational energy benefits of encoder-block and overall Spike-FlowNet are obtained, as provided in the fourth and the fifth rows of Table 6.2, respectively. These results reveal that the SNN-based encoder-block is 214.2× and 25.51× more computationally efficient compared to ANN-based one (averaged over indoor1,2,3 and outdoor1 sequences) for $dt = 1$ and $dt = 4$ cases, respectively. The number of time-steps (N) is four times less in $dt = 1$ case than in $dt = 4$ case; hence, the computational energy benefit is much higher in $dt = 1$ case.

From our analysis, the proportion of required computations in encoder-block compared to the overall architecture is 17.6%. This reduces the overall energy benefits of Spike-FlowNet. In such a case, an approach of interest would be to perform a distributed edge-cloud implementation where the SNN- and ANN-blocks are administered on the edge device and the cloud, respectively. This would lead to high energy benefits on edge devices, which are limited by resource constraints while not compromising on algorithmic performance.

6.5 Conclusion

In this chapter, we propose Spike-FlowNet, a deep hybrid architecture for energy-efficient optical flow estimations using event camera data. To leverage the benefits of both SNNs and ANNs, we integrate them in different layers for resolving the spike vanishing issue in deep SNNs. Moreover, we present a novel input encoding strategy for handling outputs from event cameras, preserving the spatial and temporal information over time. Spike-

FlowNet is trained with a self-supervised learning method, bypassing expensive labeling. The experimental results show that the proposed architecture accurately predicts the optical flow from discrete and asynchronous event streams along with substantial benefits in terms of computational efficiency compared to the corresponding ANN architecture.

7. FUSION-FLOWNET: ENERGY-EFFICIENT OPTICAL FLOW ESTIMATION USING SENSOR FUSION AND DEEP FUSED SPIKING-ANALOG NETWORK ARCHITECTURES

7.1 Introduction

Frame-based or event sensors themselves can not efficiently capture all the relevant information in the scenes. The limited applicability of each individual camera gives rise to the need for an optimal sensor-fusion technique enabling the sensors to complement the limitations of each other. Such a technique would provide a practical solution towards accurately estimating the dense pixel motion in challenging scenarios such as high dynamic range and rapid motion environments.

When considering event-based cameras, conventional computer vision and ANN-based methods turn out to be incompatible at handling the discrete and asynchronous event streams in their native form. This is due to the fact that these methods are generally designed for frame-based images, assuming brightness consistency over frames. In this regard, Spiking Neural Networks (SNNs) show great promise for directly handling event-camera outputs. Furthermore, SNNs perform efficient event-based computations by carrying out operations only at the arrival of the input events, exploiting the inherent sparsity of spatio-temporal event streams, and thus, enabling energy-efficient computations on specialized neuromorphic hardware such as Intel's Loihi [119] and IBM's TrueNorth [8].

However, recent works have shown a noticeable drawback – the number of spikes drastically reduce in the deeper layers of SNNs, termed as the "spike vanishing" phenomenon. This hinders learning, leading to performance degradation. Such issues can be efficiently addressed by fusing SNNs and ANNs in a network where SNNs form the initial layers enabling efficient event stream handling [144]. While the later layers are ANNs, addressing the "spike vanishing" problem and helping to retain performance. To that effect, we propose Fusion-FlowNet, a deep fused spiking-analog architecture for estimating optical flow that uses sensors of different modalities (standard frame-based image and event stream). The

proposed architecture achieves state-of-the-art optical-flow estimation performance on the Multi-Vehicle Stereo Event Camera (MVSEC) dataset [121].

The main contributions of this work are as follows:

- We propose Fusion-FlowNet architecture composed of a fusion of SNNs and ANNs, for simultaneously processing event streams and frame-based images, respectively, leveraging their complementary sensing capabilities.
- We present a Signed Integrate-and-Fire (S-IF) neuron model for SNNs which can generate spike outputs with both positive and negative polarity. The S-IF model coupled with a surrogate gradient method, enables backpropagation based training in SNNs.
- We evaluate the optical flow predictability of Fusion-FlowNet on the large scale event-camera dataset (MVSEC) and provide comparisons with corresponding state-of-the-art methods. We also analyze the benefits of Fusion-FlowNet in terms of network parameter reduction, computational energy and memory costs.

The remainder of our paper is organized as follows. First, we discuss the related works. Second, we introduce the concepts related to individual sensors and sensor-fusion for generating inputs. Third, we explain the analog and spiking neuron models employed in this work, and we propose the Fusion-FlowNet architecture consisting of SNNs and ANNs. Next, we describe the surrogate gradient-based backpropagation algorithm to enable end-to-end unsupervised training. Finally, we analyze the experimental results, including comparisons with other recent works accompanied by various ablation studies.

7.2 Related Works

Over the past few years, there have been major advancements towards optical flow estimation using event-cameras. Conventional computer vision algorithms have been adapted to encompass discrete outputs from event cameras in [125], [126], [129]. Recently, several works for handling asynchronous event streams using the ANN- and SNN-based approaches have gained immense popularity.

In ANN-based approaches, the asynchronous events are essentially accumulated for fixed time intervals to generate synchronous frames. In EV-FlowNet [133], the recent event counts as well as pixel-wise last timestamp information is encoded in a frame-based representation. However, this approach greatly suffers during rapid motions and dense localized events resulting in loss of rich temporal information. Researchers in [136] further extended this to a 3D input representation that considers time domain as a channel and uses an input interpolation scheme for retaining the spatio-temporal event distributions. Nevertheless, this approach still suffers at predicting dense pixel-wise outputs in regions where events are extremely sparse.

Among SNN-based approaches, Spike-FlowNet [144] introduced an input encoding scheme that transformed the raw event stream into two groups consisting of multiple discretized event frames. This scheme allowed for encoding the presence of events over time, preserving the spatio-temporal information and enabled event-based computations in SNNs. However, since only the event stream is used as input, the predictions are accurate only where events are present, thereby limiting dense motion prediction.

Meanwhile, [145] presents an optimization based optical flow estimation method by jointly using a set of events and a single frame-based image. It uses a variational approach to recover a sharp image from these inputs. Contrary to this, our method utilizes all available event streams as well as frame-based images within a time window. This enables accurate flow estimations for longer time windows. Furthermore, we explore a neural network-based approach and directly handle event streams and frame-based images.

7.3 Sensors and Input Representation

7.3.1 Frame-based Images

Frame-based images have been widely popular for computer vision applications. They provide highly accurate pixel-wise intensity information as frames over regular time intervals. This intensity frame information is pivotal in numerous applications such as face and object recognition tasks [146]. Likewise, for ANN-based optical flow estimation, the consecutive

frame-based images are passed as inputs in separate channels to the network. In this chapter, we utilize this input representation for the ANN part of Fusion-FlowNet.

7.3.2 Stream of Events

Event-based cameras are bio-inspired vision sensors that provide a stream of events as an outcome of tracking intensity changes (I) at each pixel element. Whenever the logarithmic intensity change surpasses a specified threshold (θ), a discrete event is asynchronously generated as follows:

$$\| \log(I_{t+1}) - \log(I_t) \| \geq \theta \tag{7.1}$$

Event cameras output data in Address Event Representation (AER) format which incorporates a tuple {x, y, t, p}, composed of the pixel address (x and y coordinates), timestamp (t), and polarity of the intensity change (p). The polarity can have values "ON" or "OFF" corresponding to the increase or decrease in intensity of that pixel. This four-dimensional data format, referred to as address event representation (AER), is able to encapsulate asynchronous event data with high temporal resolution and is the standard communication protocol used by event sensors.

Event cameras may not be generally suited for tasks which require dense input information such as face recognition [146], where frame-based cameras dominate. However, event cameras promise colossal benefits in challenging conditions such as high-speed motion detection and environments with high dynamic range in addition to having low power consumption. Thus, the task of optical flow estimation can greatly benefit from the usage of event cameras in terms of high temporal resolution, robustness to high dynamic range scenes and energy-efficiency.

For providing the inputs to SNN part of Fusion-FlowNet, we transform the raw event stream into the groups of discretized event frames and predict optical flow between frames belonging to these groups. The input to the SNN encoder-branch consists of a sequence of event frames with four channels, each from the ON/OFF polarity of event frames from the former and the latter groups. This scheme preserves the temporal information in the

event stream to display the superior algorithmic performance and promising energy-efficiency. Fig. 6.1 depicts the employed input representation for event data.

7.3.3 Sensor-fusion

Interestingly, numerous sensors such as the Dynamic and Active Vision Sensor (DAVIS) [140] are capable of simultaneously generating the asynchronous events as well as synchronous grayscale frames at fixed intervals, simplifying the hardware costs of sensor-fusion. In addition, since there is a single camera coordinate system for both data modalities, the requirement for expensive transformation and synchronization between multiple coordinate systems is eliminated. We therefore employ the DAVIS sensor for our purpose.

For this work, the frame-based images serve two purposes. First, they are provided as network inputs and allow for dense optical flow predictions. Second, they are used for computing the unsupervised loss required for training, as proposed in [134]. On the other hand, the event stream is only provided as network input and enables sparse optical flow prediction in aforementioned challenging scenes. In this regards, the proposed sensor fusion framework would therefore allow computing dense optical flow at high-speed in challenging environments

7.4 Neuron Models

The primary difference between ANN and SNN operations is the notion of time. While ANNs feed-forward the dense analog-valued inputs at once, SNNs process the sparse binary inputs as a function of time. Accordingly, different neuron models are employed in ANNs and SNNs.

7.4.1 LeakyReLU Model

In ANNs, the LeakyReLU [147] replaces the negative part of the popular ReLU model by a linear function with a relatively small slope as below:

$$y = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases} \tag{7.2}$$

where α is typically set to 0.01-0.1. This concept fixes the "dead neuron" problem that some neurons get stuck in the negative side and play no role in discriminating between inputs. LeakyReLU model has been reported to be useful especially for hard regression tasks such as motion estimations, predicting pixel-wise and high-resolution outputs. In this chapter, we therefore employ the LeakyReLU model for the ANN parts of Fusion-FlowNet.

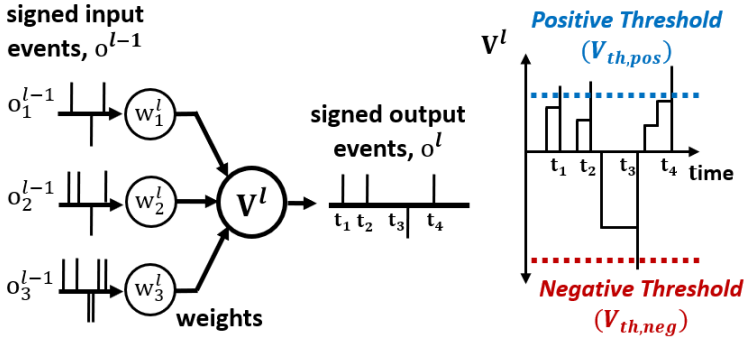


Figure 7.1. Dynamics of Signed Integrate-and-Fire (S-IF) neuron model. Whenever the membrane potential crosses either positive- or negative-threshold, the neuron fires a signed spike and resets its membrane potential.

7.4.2 Signed Integrate-and-Fire (S-IF) Model

Spiking neurons are inspired by biological models that emulate efficient event-based operations in the human brain. In the literature, the Integrate-and-Fire (IF) neuron model [27] is commonly used for building SNNs because of its simplicity. In an IF neuron, input spikes are modulated by weight (w) and accumulated in an internal state of the neuron, called membrane potential over time. Whenever the membrane potential (v) crosses a firing

threshold, the neuron emits a binary output (1 or 0) and resets the membrane potential as follows,

$$v^l[n+1] = v^l[n] + w^l o^{l-1}[n] \tag{7.3}$$

where $o^{l-1}[n]$ indicates the spike output from the l-1^{th} layer at time-step n. However, the IF neuron too suffers from the "dead neuron" problem as discussed previously. Thus, in this chapter, we present a signed integrate-and-fire (S-IF) neuron model that can generate signed spike outputs. The S-IF neuron model contains positive- and negative-thresholds that play a role in generating positive- and negative-valued spike outputs, respectively. This operation is illustrated in Fig. 7.1 and formulated as follows:

$$o^l = \begin{cases} +1, & \text{if } v^l > v_{th,pos} \\ -1, & \text{elif } v^l < v_{th,neg} \\ 0, & \text{otherwise} \end{cases} \tag{7.4}$$

However, the discontinuous S-IF's spike generation function poses a challenge for gradient-based training. To overcome this challenge, we propose a surrogate gradient method for enabling an end-to-end backpropagation training which will be discussed in a later section.

7.5 Fusion-FlowNet Architecture

Fusion-FlowNet incorporates a deep fused network architecture that supports the end-to-end training as illustrated in Fig. 7.2. It is built upon the U-Net architecture [132], containing four encoder layers, two residual blocks, and four decoder layers. The distinctions in this chapter involve the addition of dual pathways starting at the encoder, namely the SNN- and ANN-based branches. Each branch is composed of narrow convolution layers (similar to grouped convolution used in AlexNet [4]) containing half the number of intermediate feature maps, compared to original wide convolution layers in U-Net. This is possible because of the usage of different modalities of input data, leading to reduction in network parameters without compromising performance. The branches merge later to together generate an optical flow map.

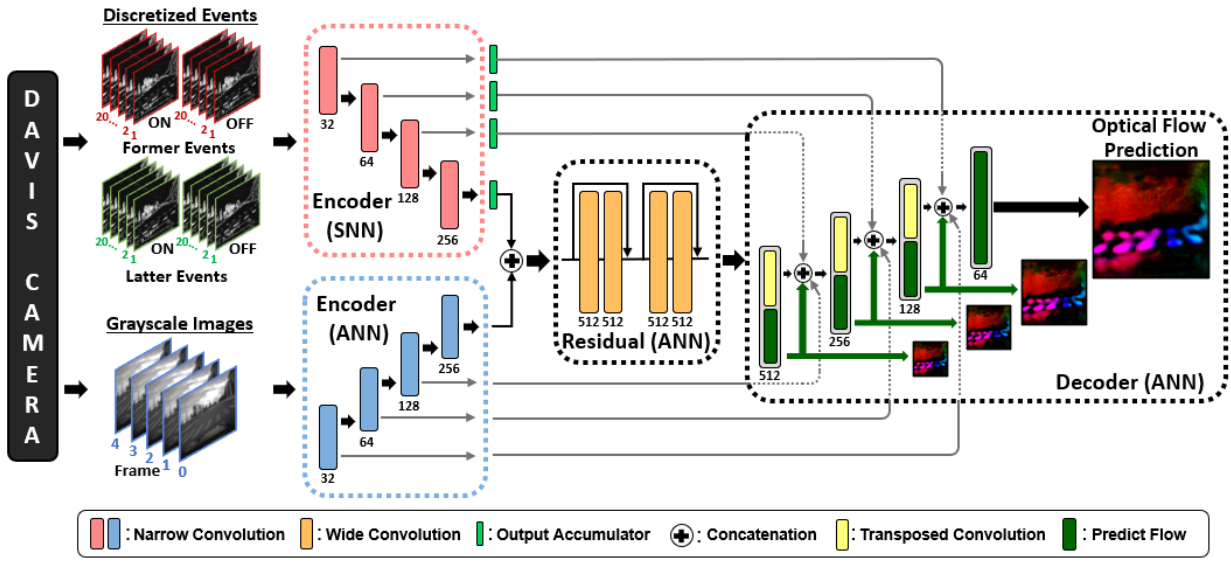


Figure 7.2. The detailed illustration of the Fusion-FlowNet$_{\text{Early}}$. The network contains the SNN- and ANN-based encoder-branches to extract features from event streams and grayscale images, respectively. The rest of networks, involving residual and decoder blocks, are composed of ANN layers. The colors represent the types of layers.

In the SNN-based encoder-branch, the four-channeled input event frames sequentially pass through the narrow convolution layers consisting of S-IF neurons over time while being downsampled at each layer. At every time-step, the weighted spike outputs from each layer are integrated into the corresponding output accumulator. After passing all consecutive event images, the output accumulator outcome becomes the part of input to the subsequent ANN layers.

In the ANN-based encoder-branch, the consecutive frame-based images in the time-window pass through the narrow ANN layers at once. Each ANN layer comprises of a convolution and a batch-norm layer [3] before the LeakyReLU activation layer. Here too, the feature maps are downsampled at each layer.

After completing forward propagation in both encoder-branches, the final outputs are fused before passing through the rest of the network, namely the residual and the decoder blocks composed of standard wide ANN layers. To fuse the hierarchical features from the encoders, the intermediate activations are concatenated at the same spatial locations from both

the SNN and ANN branches. The fused activations pass through two residual blocks, and then through the four decoders layers which upsample them using transposed convolutions. At each decoder layer, skip connections from the corresponding encoder layers and another convolution layer produce a multi-scale dense optical flow prediction, which is concatenated with the activations from transposed convolution layers. Finally, a full-scale optical flow having the same dimension as the input frames are predicted at the output.

7.6 Unsupervised Training Method

Due to the limited availability of event-camera datasets containing ground-truth labels, we adopt an unsupervised approach to train unsupervised optical flow [134]. Fusion-FlowNet is trained using unlabeled sequences, utilizing frame-based images to act as proxy labels for computing the loss. The overall loss functions are composed of two parts:

$$l_{\text{total}} = l_{\text{photo}} + \lambda l_{\text{smooth}} \tag{7.5}$$

where l_{photo} represents a photometric loss, l_{smooth} indicates the smoothness loss, and (λ) denotes the weight factor between the two loss.

7.6.1 Photometric Loss

Photometric loss helps realize the motion of pixels in an image over time by tracking the pixel intensities between images. The network is provided with start and end-frame grayscale images (I_t, I_{t+dt}). A spatial transformer inversely warps the last frame-based image with the current estimated optical flow to compute an image prediction. Then, the photometric loss (l_{photo}) aims to minimize the discrepancy between the first frame-based image and this image prediction. Photometric loss is computed as follows:

$$l_{photo} = \sum_{x,y} \rho(I_t(x,y) - I_{t+dt}(x + u, y + v)) \tag{7.6}$$

where, I_t (I_{t+dt}) indicate the pixel intensity of the first (last) frame-based images, u, v are the flow estimates in the x, y directions, ρ is the robust Charbonnier loss $\rho(x) = (x^2 + \eta^2)^r$, used for outlier rejection [141]. We set $r = 0.45$ and η =1e-3 as they show optimum results in prior works [133], [144].

7.6.2 Smoothness Loss

Smoothness loss (l_{smooth}) is applied to reduce optical flow deviations between neighboring pixels by adding a regularizing effect on the predicted flow. It is computed as follows:

$$l_{smooth} = \sum_{j}\sum_{i}(\|u_{i,j} - u_{i+1,j}\| + \|u_{i,j} - u_{i,j+1}\| + \|v_{i,j} - v_{i+1,j}\| + \|v_{i,j} - v_{i,j+1}\|) \quad (7.7)$$

7.7 Backpropagation in Fusion-FlowNet

After the forward propagation, the final loss (l_{total}) is evaluated, followed by the backward propagation of gradients.

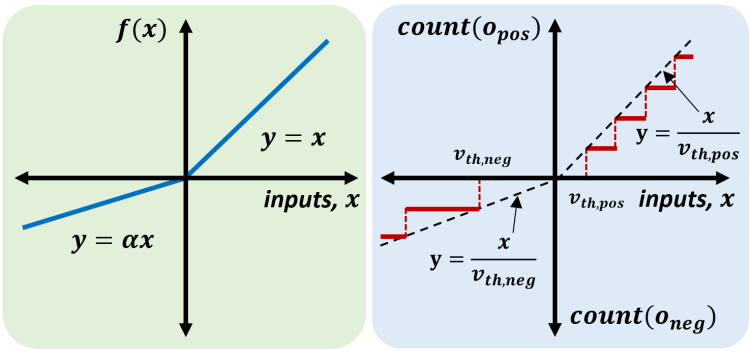


Figure 7.3. The illustrations of activation function and its derivative ($left$) LeakyReLU neuron ($right$) S-IF neuron.

In ANN layers, the LeakyReLU is a differentiable activation that can be represented by the linear functions where the slope differs in positive and negative parts of input. The derivative of LeakyReLU activation ($\frac{\partial f(x)}{\partial x}$) is unity when input is positive and α when input is negative, and zero otherwise. Hence, standard backpropagation can calculate the gradient

of the loss function with respect to each weight using chain rule. The parameter updates for the l^{th} ANN layer are described as follows:

$$\triangle w^l_{ANN} = \frac{\partial \text{loss}}{\partial f(x^l)} \frac{\partial f(x^l)}{\partial o^l} \frac{\partial o^l}{\partial w^l} \tag{7.8}$$

On the other hand, the spike generation mechanism of S-IF neuron results in a hard threshold function making it discontinuous and non-differentiable. Thus, standard back-propagation cannot be applied to SNNs in its native form. To overcome this impediment, we present a surrogate gradient method, similar to [16], [120], for approximately estimating the S-IF neuronal spike generation function. The surrogate gradient of S-IF model is herein computed as follows:

$$\frac{\partial o[n]}{\partial v[n]} = \begin{cases} \frac{1}{V_{th,pos}}, & \text{if } v^l > v_{th,pos}. \\ \frac{1}{V_{th,neg}}, & \text{if } v^l < v_{th,neg}. \\ 0, & \text{otherwise.} \end{cases} \tag{7.9}$$

where each threshold $(V_{th,pos}, V_{th,neg})$ accounts for the change in the signed spike outputs with respect to the inputs. During the backward pass, the errors $(\frac{\partial l_{total}}{\partial o^l})$ are backpropagated through the SNN layers using the surrogate gradient above and BackPropagation Through Time (BPTT) [71]. In BPTT, the network is unrolled for all time-steps, and the weight update is assessed as the sum of gradients over each time-step. The parameter updates of the l^{th} SNN layer are described as follows:

$$\triangle w^l_{SNN} = \sum_n \frac{\partial \text{loss}}{\partial o^l[n]} \frac{\partial o^l[n]}{\partial v^l[n]} \frac{\partial v^l[n]}{\partial w^l} \tag{7.10}$$

7.8 Experiments

7.8.1 Dataset and Training Details

We train Fusion-FlowNet on the MVSEC dataset [121] which contains events as well as grayscale frames recorded using the DAVIS346 camera [140] in multiple environments. The indoor_flying sequences were collected with a drone flying indoors and the outdoor_day sequences were recorded from a car driving on public roads. We employ the outdoor_day2

sequence as well as use two different time-window lengths (i.e, 1 (dt=1) and 4 (dt=4)) grayscale images apart, to provide fair comparisons with prior works [133], [136], [144].

The events and frame-based images from left-camera images are used for training and are randomly cropped to 256×256 size, flipped horizontally and vertically (with 0.5 probability). The learning rate is scaled by 0.7 every 5 epochs until 20 epoch, and every 10 epochs thereafter. The number of event frames in each group (N) are set to 5 for the $dt = 1$ case and 20 for the $dt = 4$ case. In ANN layers, LeakyReLU model is employed with an α of 0.1. In the SNN layers, the positive and negative thresholds of the S-IF neuron are set to 0.75 and 7.5, respectively. The weight factor between losses λ is set to 0.0003.

Table 7.1. AEE_{event} comparison with previous works (lower is better).

	dt=1 frame				dt=4 frame			
AEE_{event}	ind1	ind2	ind3	out1	ind1	ind2	ind3	out1
Zhu et al.'19	0.58	1.02	0.87	**0.32**	2.18	3.85	3.18	1.30
EV-FlowNet	1.03	1.72	1.53	0.49	2.25	4.05	3.45	1.23
Spike-FlowNet	0.84	1.28	1.11	0.49	2.24	3.83	3.18	**1.09**
Fusion-FlowNet	**0.56**	**0.95**	**0.76**	0.59	**1.68**	**3.24**	**2.43**	1.17

7.8.2 Evaluation of Optical Flow

The center cropped images of 256 × 256 size are taken from indoor_flying1,2,3 and outdoor_day1 sequences. We take all events for indoor_flying, but use the event stream within 800 grayscale frames for the outdoor_day1 sequence, as suggested in [133], [144]. For quantitative estimation results, we calculate the standard Average End-point Error (AEE), which is the mean Euclidean distance between the predicted flow (y_{pred}) and the provided ground-truth (y_{gt}). In this chapter, we measure two types of AEE results: over all pixels (AEE_{all}) and over pixels only where events are present within the time-window (AEE_{event}):

$$\text{AEE} = \frac{1}{m} \sum_{\text{m}} \left\| (u, v)_{\text{pred}} - (u, v)_{\text{gt}} \right\|_2 \tag{7.11}$$

where m indicates the count of active pixels in the event frames for AEE_{event} and every pixels of images for AEE_{all}.

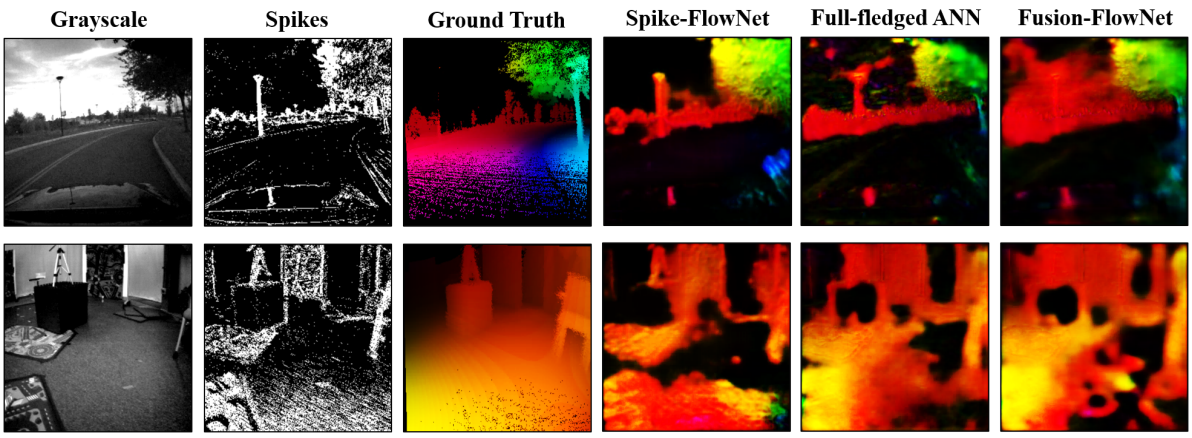


Figure 7.4. Predicted optical flow compared with Spike-FlowNet and Full-fledged ANN. The samples are taken from (*top*) *outdoor_day*1 and (*bottom*) *indoor_flying*2. Best viewed in color.

Table 7.2. Average endpoint error (AEE) results for ablation studies (lower is better)

	indoor1				indoor2				indoor3				outdoor1			
	dt=1		dt=4		dt=1		dt=4		dt=1		dt=4		dt=1		dt=4	
	event	all	event	all	event	all	event	all	event	all	event	all	event	all	event	all
Fusion-FlowNet$_{\text{Early}}$	**0.56**	**0.62**	**1.68**	**1.81**	**0.95**	**0.89**	**3.24**	**2.90**	**0.76**	**0.85**	**2.43**	**2.46**	0.59	1.02	1.17	**3.06**
Fusion-FlowNet$_{\text{Late}}$	0.57	0.63	1.71	1.89	0.99	0.92	3.26	2.93	0.79	0.87	2.46	2.54	0.55	1.00	1.34	3.48
Fusion$_{\text{Early}}$ [IF model]	**0.56**	**0.62**	1.72	1.93	0.97	0.90	3.36	3.07	0.78	0.87	2.51	2.63	0.58	1.04	1.37	3.52
Fusion$_{\text{Late}}$ [IF model]	0.57	0.64	1.71	1.90	1.00	0.93	3.41	3.08	0.80	0.88	2.56	2.64	0.55	**0.99**	1.38	3.53
Spike-FlowNet	0.84	0.91	2.24	2.94	1.28	1.23	3.83	4.09	1.11	1.20	3.18	3.92	**0.49**	1.42	**1.09**	3.28
Full-fledged ANN	0.60	0.68	1.73	1.90	1.00	0.97	3.35	3.03	0.83	0.97	2.52	2.62	0.83	1.53	1.27	3.19

7.8.3 Results

We compare the performance of Fusion-FlowNet with corresponding state-of-the-art implementations [133], [136], [144] in terms of the AEE$_{event}$ metric as discussed above. The relevant results are listed in Table 7.1. Only AEE$_{event}$ results are compared here since the other works do not provide AEE values for dense optical flow. We observe that Fusion-FlowNet outperforms other implementations in almost all scenarios. The outdoor_day1 sequence is known to have suffered from certain issues with its grayscale images during dataset creation, leading to anomalous results for AEE$_{event}$ as well as AEE$_{all}$. Fig. 7.4 visualizes the predicted flows for Spike-FlowNet [144] which uses only event-inputs, fully-ANN architecture which uses only grayscale images and Fusion-FlowNet which uses both.

7.8.4 Ablation studies

Architectural Variations

We evaluate another architecture where the dual pathway branches are extended to residual blocks. We denote the first architecture as Fusion-FlowNet$_{\text{Early}}$ and this second architecture as Fusion-FlowNet$_{\text{Late}}$. Rows 1−2 in Table 7.2 highlight the optical flow prediction capability of both the architectures. We find that Fusion-FlowNet$_{\text{Early}}$ outperforms Fusion-FlowNet$_{\text{Late}}$ in predicting accurate optical flow outputs. Fusion-FlowNet$_{\text{Early}}$ contains a large number of parameters and fuses the intermediate features from the branches in early layers, leading to better AEE results. On the other hand, Fusion-FlowNet$_{\text{Late}}$ has promising advantages in further reducing the network parameters and computational costs, as will be discussed in Table 7.3.

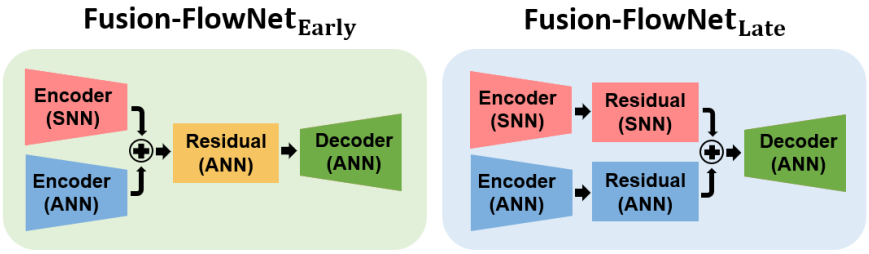


Figure 7.5. The architectures of $(left)$ Fusion-FlowNet$_{\text{Early}}$ and $(right)$ Fusion-FlowNet$_{\text{Late}}$.

Neuron Model Choice

For investigating the benefits of S-IF neuron model, we compare variations of Fusion-FlowNet, consisting of S-IF and IF neuron models for SNN blocks. Rows 3−4 in Table 7.2 provide the AEE results for Fusion-FlowNet with IF neurons in the SNN layers. The results show that networks with S-IF neuron model can predict more accurate flow outputs, compared to networks with IF neuron model. Thus, the S-IF model helps mitigate the "dead neuron" problem in deep SNN layers similar to the LeakyReLU model.

Table 7.3. Comparison of number of parameters and computational energy cost for different architectures for dt=1 and dt=4 cases.

	#Parameters ($\times 10^6$)		#OPS_{ANN}($\times 10^9$)		Spiking Activity (%)		#OPS_{SNN}($\times 10^6$)		$E_{Total}(mJ)$		Improvement	
	dt=1	dt=4	dt=1	dt=4	dt=1	dt=4	dt=1	dt=4	dt=1	dt=4	dt=1	dt=4
Full-fledged ANN	13.044	13.046	5.339	5.367	–	–	–	–	24.536	24.666	1.00×	1.00×
Spike-FlowNet	13.039	13.039	4.409	4.409	0.480	1.008	15.81	195.99	20.296	20.458	1.21×	1.21×
Fusion-$FlowNet_{Early}$	12.269	12.270	4.648	4.648	0.173	0.174	1.03	4.18	21.381	21.384	1.15×	1.15×
Fusion-$FlowNet_{Late}$	7.549	7.550	2.849	2.849	0.147	0.179	5.24	6.44	13.113	13.114	**1.87**×	**1.88**×

* Results averaged over all indoor and outdoor1 sequences

Sensor Fusion

We perform a study to verify the usefulness of our sensor fusion approach against each type of single sensor approach using inputs as either the event streams or frame-based images. For event only approach, we investigate Spike-FlowNet [144] which is a hybrid neural architecture where the initial layers are composed of SNNs and the deeper layers are composed of ANNs. Note, Spike-FlowNet utilizes an equivalent event-based input representation scheme and unsupervised learning method, providing a fair comparison baseline. For frame-based image only approach, we implement a custom full-fledged ANN architecture that resembles the U-Net [132], and train it with an equivalent unsupervised method using the same input representation as Fusion-FlowNet.

Rows 5−6 of Table 7.2 summarize the results on the single sensor approaches. Unsurprisingly, both $FlowNet_{Early}$ and $FlowNet_{Late}$ achieve better AEE performances in dt=1 and dt=4 scenarios, compared to single sensor approach. This verifies that our fusion approach benefits from utilizing the complementary characteristics of event- and frame-based images, leading to better performance in both slow- and fast-motion scenarios. Furthermore, in comparison to prior works as listed in Table 7.1, both fusion options provide superior results, establishing a new state-of-the-art optical flow estimation framework.

7.8.5 Computational Efficiency

We evaluate the computational efficiency of Fusion-FlowNet in terms of the number of network parameters, energy and memory cost for inference.

We see that both Fusion-FlowNet$_{\text{Early}}$ and Fusion-FlowNet$_{\text{Late}}$ contain fewer number of parameters compared to a full-fledged ANN architecture and Spike-FlowNet (Row1 in Table 7.3). This is due to the usage of narrow convolution layers, which greatly reduce the number of parameters and computations. In particular, Fusion-FlowNet$_{\text{Late}}$ contains the least number of network parameters ($\sim$ 58% compared to full-fledged ANN), as the residual blocks contain the majority of the parameters and migrating them to utilize narrow convolutional layers helps reduce the total network parameters drastically.

For computing the energy cost for the different architectures, we briefly look into how computations in SNNs and ANNs differ from each other. SNNs perform highly sparse asynchronous accumulate (AC) operations over time. These synaptic operations are executed only at the arrival of input spikes due to the nature of binary-valued inputs. On the other hand, ANNs perform expensive multiply-and-accumulate (MAC) operations for computing dense matrix-vector multiplications (MVMs). Based on the findings in [143], a MAC operation requires a total of E_{MAC}=4.6pJ of energy, while an AC operation requires only E_{AC}=0.9pJ for a 32-bit floating-point computation (45nm CMOS technology). This leads to the AC operation being $\sim$ 5.1$\times$ more energy-efficient compared to the MAC operation. We utilize this result along with the methodology to calculate the number of synaptic operations as described in [144] for estimating the energy cost.

We initially calculate the total number of synaptic operations for every layer. In SNN layers, the number of synaptic operations are obtained by multiplying the pre-spike activities, number of synaptic connections and time-steps. Also, the computational energy of AC and MAC computations are taken into considerations for SNNs and ANNs, respectively. The energy cost computation can be formalized as:

$$\#\text{OPS}_{\text{SNN}} = N \sum_{l} M_l C_l F_l, \quad \#\text{OPS}_{\text{ANN}} = \sum_{l} M_l C_l \tag{7.12}$$

$$\text{E}_{\text{Total}} = \#\text{OPS}_{\text{SNN}} \times \text{E}_{\text{AC}} + \#\text{OPS}_{\text{ANN}} \times \text{E}_{\text{MAC}} \tag{7.13}$$

where M is the number of neurons, C is the number of synaptic connections, F represents the mean spiking activity, N is the number of timesteps, $\#\text{OPS}_{\text{SNN}}$ and $\#\text{OPS}_{\text{ANN}}$ indicate

the number of operations for SNN and ANN portions, respectively and E_{Total} denotes the total energy cost.

Row 5 in Table 7.3 provides the total energy cost. It is observed that Fusion-FlowNet$_{Late}$ demonstrates the highest improvement in terms of energy cost ($\sim 1.88\times$) compared to full-fledged ANN. This is because more layers utilize narrow convolutions, leading to reduction in the number of parameters and consequently reduction in the energy cost. Furthermore, SNN pathway contributes negligibly towards increasing the energy cost when compared to ANN pathway.

The number of memory accesses can be estimated by the sum of total ANN and SNN operations. The ANN portion dominates the memory cost as $\#OPS_{ANN}$ are much greater than $\#OPS_{SNN}$, as depicted in Table 7.3. Both Fusion-FlowNet models show significant reduction in $\#OPS_{ANN}$ compared to full-fledged ANN, thereby leading to lower memory costs. Again, Fusion-FlowNet$_{Late}$ requires a fraction of ($\sim 53\%$) memory accesses compared to full-fledged ANN.

7.9 Conclusion

In this chapter, we propose a sensor/architecture fusion framework for accurately estimating optical flow. We leverage the complementary characteristics of event- and frame-based sensors as well as ANNs and SNNs. Our framework (Fusion-FlowNet) reports state-of-the-art optical flow prediction results, while substantially reducing network parameters and computational costs. This work contributes two different deep fused architectures (Fusion-FlowNet$_{Early}$ and Fusion-FlowNet$_{Late}$), having different applications of interest. Fusion-FlowNet$_{Early}$ provides highly accurate dense optical flow, proving to be appropriate for safety-critical applications. While, Fusion-FlowNet$_{Late}$ promises immense benefits in terms of computational efficiency, making it suitable for the edge applications on resource-constrained hardware.

8. SUMMARY AND FUTURE WORK

In recent times, SNNs have been explored toward realizing robust and energy-efficient machine intelligence guided by the cues from neuroscience experiments [148]. However, the typical shallow spiking network architectures have limited capacity for expressing complex representations, while training a very deep spiking network has not been successful.

This thesis proposes the spike-based learning algorithms that enable deep SNNs to achieve competitive accuracy, energy-efficiency and robustness for image classifications. First, we propose a layer-wise unsupervised STDP for training deep convolutional SNNs (chapter 2). Second, we develop an approximate derivative method to overcome the discontinuous and non-differentiable nature of spike generation function and to enable training deep convolutional SNNs with input spike events using the supervised spike-based backpropagation algorithm (chapter 3). Third, we present a pre-training scheme using biologically plausible unsupervised STDP learning in order to better initialize the network parameters prior to supervised spike-based backpropagation (chapter 4). In addition, we analyze the neuron models with and without leak to investigate the impacts of leak on noise robustness and spike sparsity in deep SNNs (chapter 5).

Moreover, this thesis explores the event-based vision applications where SNNs outperform the corresponding ANNs in terms of output quality while providing significant computational efficiency. Event-based camera shows promising advantages, namely high temporal resolution, high dynamic range and low power consumption. However, conventional computer vision methods as well as deep ANNs are no longer compatible in their native form with the asynchronous and discrete nature of event camera outputs. We show the great potential of SNNs for directly handling event-camera outputs. Furthermore, we demonstrate that SNNs can effectively exploit the inherent sparsity of event streams by performing efficient event-based computations, carrying out operations only at the arrival of the input events (chapter 6). Finally, we propose Fusion-FlowNet, a sensor/architecture fusion framework, leveraging the complementary characteristics of event- and frame-based sensors as well as ANNs and SNNs (chapter 7).

In the future, there will be several interesting pathways in this research field. First, there is a need to further explore novel SNN applications that can take full advantage of asynchronous and sparse event-based computation capability. Such ideal applications would largely contain the temporal and sequential data processing that necessitate huge deep ANNs or computationally intensive recurrent networks. One promising SNN application would be to estimate emotion detection using an electroencephalogram (EEG) which records the sequential electrical activity in the brain. Next, while the computational energy benefit of SNNs has been demonstrated in theoretical analysis, SNNs have yet to be tightly integrated into the neuromorphic hardware. The recent efforts of neuromorphic community have resulted in the realization of the specialized hardware such as IBM's TrueNorth and Intel's Loihi. Accordingly, we will need to integrate SNNs for those hardware to fully exploit the asynchronous event-based computations and thus efficiently deliver them to challenging real-world application such as flying drones or autonomous driving.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[5] 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.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[7] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[8] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[9] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, Ieee, 2008, pp. 2849–2856.

[10] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[11] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *arXiv preprint arXiv:1802.02627*, 2018.

[12] 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.

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

[14] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco, and H. Tang, "Feedforward categorization on aer motion events using cortex-like features in a spiking neural network," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 9, pp. 1963–1978, 2015.

[15] J. M. Brader, W. Senn, and S. Fusi, "Learning real-world stimuli in a neural network with spike-driven synaptic dynamics," *Neural computation*, vol. 19, no. 11, pp. 2881–2912, 2007.

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

[17] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "Stdp-based spiking deep neural networks for object recognition," *arXiv preprint arXiv:1611.01421*, 2016.

[18] P. O'Connor and M. Welling, "Deep spiking networks," *arXiv preprint arXiv:1602.08323*, 2016.

[19] P. Panda and K. Roy, "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *Neural Networks (IJCNN), 2016 International Joint Conference on*, IEEE, 2016, pp. 299–306.

[20] 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, Feb. 2008, ISSN: 1558-173X. DOI: 10.1109/JSSC.2007.914337.

[21] T. V. Bliss, G. L. Collingridge, *et al.*, "A synaptic model of memory: Long-term potentiation in the hippocampus," *Nature*, vol. 361, no. 6407, pp. 31–39, 1993.

[22] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International journal of neural systems*, vol. 19, no. 04, pp. 295–308, 2009.

[23] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," *Neural networks*, vol. 14, no. 6, pp. 715–725, 2001.

[24] R. Van Rullen and S. J. Thorpe, "Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex," *Neural computation*, vol. 13, no. 6, pp. 1255–1283, 2001.

[25] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS computational biology*, vol. 3, no. 2, e31, 2007.

[26] W. Gilles, T. Michèle, and P. Khashayar, "Intrinsic variability of latency to first-spike," *Biological cybernetics*, vol. 103, no. 1, p. 43, 2010.

[27] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biological cybernetics*, vol. 95, no. 1, pp. 1–19, 2006.

[28] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers in neuroscience*, vol. 5, 2011.

[29] A. Jaiswal, S. Roy, G. Srinivasan, and K. Roy, "Proposal for a leaky-integrate-fire spiking neuron based on magnetoelectric switching of ferromagnets," *IEEE Transactions on Electron Devices*, vol. 64, no. 4, pp. 1818–1824, 2017.

[30] A. Sengupta, P. Panda, P. Wijesinghe, Y. Kim, and K. Roy, "Magnetic tunnel junction mimics stochastic cortical spiking neurons," *Scientific reports*, vol. 6, p. 30 039, 2016.

[31] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[32] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[33] P. Dayan and L. F. Abbott, *Theoretical neuroscience*. Cambridge, MA: MIT Press, 2001, vol. 806.

[34] A. Huertas and G. Medioni, "Detection of intensity changes with subpixel accuracy using laplacian-gaussian masks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 5, pp. 651–664, 1986.

[35] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[37] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[38] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *Rebooting Computing (ICRC), IEEE International Conference on*, IEEE, 2016, pp. 1–8.

[39] M. Mozafari, S. R. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini, and M. Ganjtabesh, "First-spike based visual categorization using reward-modulated stdp," *arXiv preprint arXiv:1705.09132*, 2017.

[40] A. Tavanaei and A. S. Maida, "Bio-inspired spiking convolutional neural network using layer-wise sparse coding and STDP learning," *CoRR*, vol. abs/1611.03000, 2016. arXiv: 1611.03000. [Online]. Available: http://arxiv.org/abs/1611.03000.

[41] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral cortex*, vol. 17, no. 10, pp. 2443–2452, 2007.

[42] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Frontiers in neural circuits*, vol. 9, p. 85, 2016.

[43] 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.

[44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[45] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[46] D. Kappel, S. Habenschuss, R. Legenstein, and W. Maass, "Network plasticity as bayesian inference," *PLoS computational biology*, vol. 11, no. 11, e1004485, 2015.

[47] D. Kappel, R. Legenstein, S. Habenschuss, M. Hsieh, and W. Maass, "A dynamic connectome supports the emergence of stable computational function of neural circuits through reward-based learning," *Eneuro*, vol. 5, no. 2, 2018.

[48] E. O. Neftci, B. U. Pedroni, S. Joshi, M. Al-Shedivat, and G. Cauwenberghs, "Unsupervised learning in synaptic sampling machines," *arXiv preprint arXiv:1511.04484*, 2015.

[49] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[52] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, 2011, p. 5.

[53] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[54] 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.

[55] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120 db 15μs latency asynchronous temporal contrast vision sensor," *IEEE journal of solid-state circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[56] E. Hunsberger and C. Eliasmith, "Spiking deep networks with lif neurons," *arXiv preprint arXiv:1510.08829*, 2015.

[57] S. S. Sarwar, G. Srinivasan, B. Han, P. Wijesinghe, A. Jaiswal, P. Panda, A. Raghunathan, and K. Roy, "Energy efficient neural computing: A study of cross-layer approximations," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2018.

[58] G. Srinivasan, P. Panda, and K. Roy, "Spilinc: Spiking liquid-ensemble computing for unsupervised speech and image recognition," *Frontiers in Neuroscience*, vol. 12, p. 524, 2018.

[59] G. Srinivasan, P. Panda, and K. Roy, "Stdp-based unsupervised feature learning using convolution-over-time in spiking neural networks for energy-efficient neuromorphic computing," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 4, p. 44, 2018.

[60] Y. Jin, P. Li, and W. Zhang, "Hybrid macro/micro level backpropagation for training deep spiking neural networks," *arXiv preprint arXiv:1805.07866*, 2018.

[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, 2018.

[62] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Frontiers in neuroscience*, vol. 11, p. 324, 2017.

[63] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE transactions on neural networks and learning systems*, 2017.

[64] A. Tavanaei and A. S. Maida, "Bio-inspired spiking convolutional neural network using layer-wise sparse coding and stdp learning," *arXiv preprint arXiv:1611.03000*, 2016.

[65] A. Tavanaei and A. S. Maida, "Multi-layer unsupervised learning in a spiking convolutional neural network," in *Neural Networks (IJCNN), 2017 International Joint Conference on*, IEEE, 2017, pp. 2023–2030.

[66] 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.

[67] S. Esser, P. Merolla, J. Arthur, A. Cassidy, R. Appuswamy, A. Andreopoulos, D. Berg, J. McKinstry, T. Melano, D. Barch, *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing. 2016," *Preprint on ArXiv. http://arxiv. org/abs/1603.08270. Accessed*, vol. 27, 2016.

[68] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Direct training for spiking neural networks: Faster, larger, better," *arXiv preprint arXiv:1809.05793*, 2018.

[69] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, 2018, pp. 1412–1421.

[70] D. Huh and T. J. Sejnowski, "Gradient descent for spiking neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 1440–1450.

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

[72] T. P. Lillicrap and A. Santoro, "Backpropagation through time and the brain," *Current opinion in neurobiology*, vol. 55, pp. 82–89, 2019.

[73] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "A solution to the learning dilemma for recurrent networks of spiking neurons," *bioRxiv*, p. 738 385, 2019.

[74] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[75] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, 2016, pp. 2849–2858.

[76] C. Lee, G. Srinivasan, P. Panda, and K. Roy, "Deep spiking convolutional neural network trained with unsupervised spike timing dependent plasticity," *IEEE Transactions on Cognitive and Developmental Systems*, 2018.

[77] P. Panda, G. Srinivasan, and K. Roy, "Convolutional spike timing dependent plasticity based feature learning in spiking neural networks," *arXiv preprint arXiv:1703.03854*, 2017.

[78] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[79] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, 2007, pp. 153–160.

[80] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[81] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," in *Artificial Intelligence and Statistics*, 2009, pp. 153–160.

[82] P. Ferré, F. Mamalet, and S. J. Thorpe, "Unsupervised feature learning with winner-takes-all based stdp," *Frontiers in computational neuroscience*, vol. 12, p. 24, 2018.

[83] C. Clopath, L. Büsing, E. Vasilaki, and W. Gerstner, "Connectivity reflects coding: A model of voltage-based stdp with homeostasis," *Nature neuroscience*, vol. 13, no. 3, p. 344, 2010.

[84] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[85] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Back-propagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, 2015, pp. 1117–1125.

[86] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 x 128 120db 30mw asynchronous vision sensor that responds to relative intensity change," in *Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers. IEEE International*, IEEE, 2006, pp. 2060–2069.

[87] T. Delbrück, B. Linares-Barranco, E. Culurciello, and C. Posch, "Activity-driven, event-based vision sensors," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, IEEE, 2010, pp. 2426–2429.

[88] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Advances in neural information processing systems*, 2001, pp. 402–408.

[89] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back-propagation," in *Advances in neural information processing systems*, 1989, pp. 177–185.

[90] P. Dayan and L. F. Abbott, *Theoretical neuroscience*. Cambridge, MA: MIT Press, 2001, vol. 806.

[91] T. P. Snutch and A. Monteil, "The sodium "leak" has finally been plugged," *Neuron*, vol. 54, no. 4, pp. 505–507, 2007.

[92] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[93] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[94] A. L. Hodgkin and A. F. Huxley, "Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo," *The Journal of physiology*, vol. 116, no. 4, pp. 449–472, 1952.

[95] R. FitzHugh, "Impulses and physiological states in theoretical models of nerve membrane," *Biophysical journal*, vol. 1, no. 6, p. 445, 1961.

[96] R. Brette and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity," *Journal of neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005.

[97] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics*. Cambridge University Press, 2014.

[98] N. Fourcaud-Trocmé, D. Hansel, C. Van Vreeswijk, and N. Brunel, "How spike generation mechanisms determine the neuronal response to fluctuating inputs," *Journal of Neuroscience*, vol. 23, no. 37, pp. 11 628–11 640, 2003.

[99] J. G. Proakis, *Digital signal processing: principles algorithms and applications*. Pearson Education India, 2001.

[100] N. Sharafi, J. Benda, and B. Lindner, "Information filtering by synchronous spikes in a neural population," *Journal of computational neuroscience*, vol. 34, no. 2, pp. 285–301, 2013.

[101] W. M. Connelly, M. Laing, A. C. Errington, and V. Crunelli, "The thalamus as a low pass filter: Filtering at the cellular level does not equate with filtering at the network level," *Frontiers in neural circuits*, vol. 9, p. 89, 2016.

[102] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, "Enabling spike-based backpropagation for training deep neural network architectures," *Frontiers in neuroscience*, 2020.

[103] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *arXiv preprint arXiv:1903.12261*, 2019.

[104] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 625–660, 2010.

[105] D. Ren, "Sodium leak channels in neuronal excitability and rhythmic behaviors," *Neuron*, vol. 72, no. 6, pp. 899–911, 2011.

[106] Ö. B. Artun, H. Z. Shouval, and L. N. Cooper, "The effect of dynamic synapses on spatiotemporal receptive fields in visual cortex," *Proceedings of the National Academy of Sciences*, vol. 95, no. 20, pp. 11 999–12 003, 1998.

[107] D. Millman, S. Mihalas, A. Kirkwood, and E. Niebur, "Self-organized criticality occurs in non-conservative neuronal networks during 'up'states," *Nature physics*, vol. 6, no. 10, pp. 801–805, 2010.

[108] I. Vasiljevic, A. Chakrabarti, and G. Shakhnarovich, "Examining the impact of blur on recognition by convolutional networks," *arXiv preprint arXiv:1611.05760*, 2016.

[109] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[110] P. Panda, I. Chakraborty, and K. Roy, "Discretization based solutions for secure machine learning against adversarial attacks," *IEEE Access*, vol. 7, pp. 70 157–70 168, 2019.

[111] J. Lin, C. Gan, and S. Han, "Defensive quantization: When efficiency meets robustness," *arXiv preprint arXiv:1904.08444*, 2019.

[112] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[113] B. Lindner, "Low-pass filtering of information in the leaky integrate-and-fire neuron driven by white noise," in *International Conference on Theory and Application in Nonlinear Dynamics (ICAND 2012)*, Springer, 2014, pp. 249–258.

[114] R. D. Vilela and B. Lindner, "Are the input parameters of white noise driven integrate and fire neurons uniquely determined by rate and cv?" *Journal of Theoretical Biology*, vol. 257, no. 1, pp. 90–99, 2009.

[115] B. Lindner and L. Schimansky-Geier, "Transmission of noise coded versus additive signals through a neuronal ensemble," *Physical Review Letters*, vol. 86, no. 14, p. 2934, 2001.

[116] B. Lindner, J. Garcıa-Ojalvo, A. Neiman, and L. Schimansky-Geier, "Effects of noise in excitable systems," *Physics reports*, vol. 392, no. 6, pp. 321–424, 2004.

[117] B. Lindner, L. Schimansky-Geier, and A. Longtin, "Maximizing spike train coherence or incoherence in the leaky integrate-and-fire model," *Physical Review E*, vol. 66, no. 3, p. 031 916, 2002.

[118] A. Borst, J. Haag, and D. F. Reiff, "Fly motion vision," *Annual Review of Neuroscience*, vol. 33, no. 1, pp. 49–70, 2010, PMID: 20225934. DOI: 10.1146/annurev-neuro-060909-153155. eprint: https://doi.org/10.1146/annurev-neuro-060909-153155. [Online]. Available: https://doi.org/10.1146/annurev-neuro-060909-153155.

[119] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018, ISSN: 1937-4143. DOI: 10.1109/MM.2018.112130359.

[120] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, "Enabling spike-based backpropagation for training deep neural network architectures," *Frontiers in Neuroscience*, vol. 14, p. 119, 2020.

[121] A. Z. Zhu, D. Thakur, T. Özaslan, B. Pfrommer, V. Kumar, and K. Daniilidis, "The multivehicle stereo event camera dataset: An event camera dataset for 3d perception," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2032–2039, 2018.

[122] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'81, Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679. [Online]. Available: http://dl.acm.org/citation.cfm?id=1623264.1623280.

[123] R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, and M. Srinivasan, "Asynchronous frameless event-based optical flow," *Neural Networks*, vol. 27, pp. 32–37, 2012, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2011.11.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608011002930.

[124] T. Brosch, S. Tschechne, and H. Neumann, "On event-based optical flow detection," *Frontiers in neuroscience*, vol. 9, p. 137, Apr. 2015. DOI: 10.3389/fnins.2015.00137.

[125] R. Benosman, C. Clercq, X. Lagorce, S. Ieng, and C. Bartolozzi, "Event-based visual flow," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 2, pp. 407–417, Feb. 2014, ISSN: 2162-2388. DOI: 10.1109/TNNLS.2013.2273537.

[126] M. T. Aung, R. Teo, and G. Orchard, "Event-based plane-fitting optical flow for dynamic vision sensors in fpga," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5. DOI: 10.1109/ISCAS.2018.8351588.

[127] F. Barranco, C. Fermuller, and Y. Aloimonos, "Bio-inspired motion estimation with event-driven sensors," in *Advances in Computational Intelligence*, I. Rojas, G. Joya, and A. Catala, Eds., Cham: Springer International Publishing, 2015, pp. 309–321, ISBN: 978-3-319-19258-1.

[128] A. Z. Zhu, N. Atanasov, and K. Daniilidis, "Event-based feature tracking with probabilistic data association," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 4465–4470. DOI: 10.1109/ICRA.2017.7989517.

[129] G. Gallego, H. Rebecq, and D. Scaramuzza, "A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation," *CoRR*, vol. abs/1804.01306, 2018. arXiv: 1804.01306. [Online]. Available: http://arxiv.org/abs/1804.01306.

[130] M. Liu and T. Delbrück, "ABMOF: A novel optical flow algorithm for dynamic vision sensors," *CoRR*, vol. abs/1805.03988, 2018. arXiv: 1805.03988. [Online]. Available: http://arxiv.org/abs/1805.03988.

[131] S. Meister, J. Hur, and S. Roth, "Unflow: Unsupervised learning of optical flow with a bidirectional census loss," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[132] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. arXiv: 1505.04597. [Online]. Available: http://arxiv.org/abs/1505.04597.

[133] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "Ev-flownet: Self-supervised optical flow estimation for event-based cameras," *arXiv preprint arXiv:1802.06898*, 2018.

[134] J. Y. Jason, A. W. Harley, and K. G. Derpanis, "Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness," in *European Conference on Computer Vision*, Springer, 2016, pp. 3–10.

[135] W.-S. Lai, J.-B. Huang, and M.-H. Yang, "Semi-supervised learning for optical flow with generative adversarial networks," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 354–364. [Online]. Available: http://papers.nips.cc/paper/6639-semi-supervised-learning-for-optical-flow-with-generative-adversarial-networks.pdf.

[136] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "Unsupervised event-based learning of optical flow, depth, and egomotion," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 989–997.

[137] G. Orchard, R. B. Benosman, R. Etienne-Cummings, and N. V. Thakor, "A spiking neural network architecture for visual motion estimation," *2013 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 298–301, 2013.

[138] G. Haessig, A. Cassidy, R. Alvarez, R. Benosman, and G. Orchard, "Spiking optical flow for event-based sensors using ibm's truenorth neurosynaptic system," *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 4, pp. 860–870, 2018.

[139] 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.

[140] C. Brandli, R. Berner, M. Yang, S. 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, Oct. 2014, ISSN: 1558-173X. DOI: 10.1109/JSSC.2014.2342715.

[141] D. Sun, S. Roth, and M. J. Black, "A quantitative analysis of current practices in optical flow estimation and the principles behind them," *Int. J. Comput. Vision*, vol. 106, no. 2, pp. 115–137, Jan. 2014, ISSN: 0920-5691. DOI: 10.1007/s11263-013-0644-x. [Online]. Available: http://dx.doi.org/10.1007/s11263-013-0644-x.

[142] 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, p. 653, 2020, ISSN: 1662-453X.

[143] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, IEEE, 2014, pp. 10–14.

[144] C. Lee, A. Kosta, A. Z. Zhu, K. Chaney, K. Daniilidis, and K. Roy, "Spike-flownet: Event-based optical flow estimation with energy-efficient hybrid neural networks," in *European Conference on Computer Vision*, Springer, 2020, pp. 366–382.

[145] L. Pan, M. Liu, and R. Hartley, "Single image optical flow estimation with an event camera," *arXiv preprint arXiv:2004.00347*, 2020.

[146] I. Masi, Y. Wu, T. Hassner, and P. Natarajan, "Deep face recognition: A survey," in *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*, IEEE, 2018, pp. 471–478.

[147] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

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

VITA

Chankyu Lee has been pursuing his PhD degree in the School of Electrical and Computer Engineering at Purdue University, West Lafayette IN, under Prof. Kaushik Roy since Fall 2015. He received his B.S. in Electrical and Electronics Engineering from Sungkyunkwan University (SKKU), Republic of Korea, in 2015. He conducted research on brain-machine interface at the Advanced Institutes of Convergence Technology, Republic of Korea, in summer 2014. In addition, he worked as the graduate intern at Nokia Bell Labs, Murray Hill NJ, in summer 2018. His primary research interests lie at the intersection of deep learning and edge computing. He has been focused on developing energy-efficient/robust deep learning algorithms, with special interests in spiking neural networks, computer vision for event-based cameras, and low-power/high-performance VLSI design for deep learning hardware.