DEEP CONVOLUTIONAL SPIKING NEURAL NETWORKS FOR IMAGE CLASSIFICATION

by

Ruthvik Vaila

A dissertation

submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Boise State University

May 2021

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the dissertation submitted by

Ruthvik Vaila

Dissertation Title: Deep Convolutional Spiking Neural Networks for Image Classification

Date of Final Oral Examination: 03 March 2021

The following individuals read and discussed the dissertation submitted by student Ruthvik Vaila, and they evaluated his presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

John N. Chiasson, Ph.D.	Chair, Supervisory Committee
Vishal Saxena, Ph.D.	Co-Chair, Supervisory Committee
Hao Chen, Ph.D.	Member, Supervisory Committee
Hani Mehrpouyan, Ph.D.	Member, Supervisory Committee
Saeed Reza Kheradpisheh, Ph.D.	External Examiner

The final reading approval of the dissertation was granted by John N. Chiasson, Ph.D., Chair of the Supervisory Committee. The dissertation was approved by the Graduate College.

DEDICATION

"Truth alone triumphs; not falsehood"

To my mother and father, who worked hard to provide for me and my brother.

To my wife, for her support.

To my brother, friends and other family members.

ACKNOWLEDGEMENTS

I would like to express my appreciation to Prof. John Chiasson for guiding me through my PhD. Dr. Chiasson played a key role in inspiring me to take a topic in Neuromorphic computing. We spent numerous hours reading papers that guided me in my research endeavors. He also encouraged me to take Statistics/Probability courses from the Mathematics and ECE departments, these courses furthered my mathematical maturity. Dr. Chiasson offered excellent courses such as Linear Systems, Stochastic Signals & Systems, and Control Systems that benefited me in furthering my understanding of Machine Learning and Artificial Intelligence. He also encouraged me to take up an internship position at ON semiconductor at a crucial time in my career. The experience and expertise that I gained during my internship at ON semiconductor has significantly advanced my qualitative, quantitative, and programming skills in the new and upcoming Deep Learning area. He also played a significant role in imparting scientific temper and spirit of inquiry.

I would also like to express my gratitude towards Prof. Vishal Saxena for being my co-advisor, he played a crucial role in discussing research areas and topics to work on. He was also a key contributor to all the research papers that I had written during my PhD.

I would like to thank my colleagues Dr. Roohollah Amiri, Dr. Mojtaba Ahmadi Almasi, Dr. Luka Daoud, Dr. Kamran Latif, and Dr. Sumedha Gandharava Dahl for their support during my stay at BSU. Dr. Roohollah Amiri played a significant role in discussing ideas pertaining to implementation of some of the algorithms during my PhD. I thank Dr. Roohollah Amiri for being a true friend in need. Furthermore, I would like to express my gratitude to Prof. Hao Chen for his course offerings and encouraging discussions that we had during my stay at BSU. I would also like to thank Prof. Hani Mehrpouyan for accommodating me in his lab. I would like to thank the ECE department and the state of Idaho, USA for providing a Graduate Fellowship for 3.5 years. Finally, I want to thank the administrative and IT staff in

the ECE department, especially, Dr. Jennifer Ambrose, Bailey Hazzard, Kristina Martin, Jason Cook, Maureen Moore and others who tirelessly worked during the pandemic years to keep the department running efficiently.

Furthermore, I would also like to thank my manager at ON semiconductor Dr. Steve Nicholes, for providing an internship opportunity to me. Finally, I would also like to thank Denver Lloyd, he played a key role in refining much of the code that I had developed during my stay at ON semiconductor.

BIOGRAPHICAL SKETCH

Ruthvik Vaila is a doctoral candidate from Hyderabad, India. He obtained his Junior college degree from Narayana Junior College in 2010, he achieved a position in top 0.9 percentile among 1 million candidates that took the All-India Engineering Entrance Exam (AIEEE) in 2010. As a result, he gained admission to National Institute of Technology-Calicut (NIT-C) and obtained his Bachelor of Technology (B.Tech) in Electrical and Electronics Engineering (EEE) in 2014. During his stay at NIT-C he qualified for the merit scholarship that was awarded to selected students from the erstwhile state of Andhra Pradesh by the Ministry of Human Resources Development (MHRD), Government of India. Subsequently he spent the Summer of 2014 at Yuktix Technologies, Bangalore where he worked on automation for Industrial and Agricultural applications. He spent the Fall of 2014 and the Spring of 2015 as an intern at the Robotics Research Center (RRC), International Institute of Information Technology, Hyderabad (IIIT-H) under the guidance of Prof. K. Madhava Krishna. He worked on topics such as lane detection and parallel parking pertaining to the Autonomous driving field during his tenure at RRC, IIIT-H.

He joined Boise State University, (BSU), Boise as a Master of Science (MS) student in Electrical and Computer Engineering in the Fall of 2015. After consulting with Prof. John Chiasson and Prof. Vishal Saxena he was convinced to join the PhD program at BSU in the Spring of 2016. From the Spring of 2016 to Summer of 2019, he worked towards his PhD full time in Neuromorphic Computing. During his full-time tenure at BSU, he was a Teaching Assistant (TA) for courses such as System Modelling & Control, and Electronic Circuits lab. He joined ON semiconductor as an intern during the Summer of 2019, subsequently the internship was extended till the Summer of 2020. During his tenure at ON semiconductor, he worked on applying modern Deep Learning tools and techniques towards reducing the time required for calibrating an image sensor. He remained a part time PhD student at BSU during his tenure at ON semiconductor. In the Summer of 2020

he joined Bastian Solution (Toyota Advanced Logistics) in Boise, Idaho as a Machine Learning Engineer where works in the field of Deep Learning for warehouse automation.

ABSTRACT

Spiking neural networks are biologically plausible counterparts of artificial neural networks. Artificial neural networks are usually trained with stochastic gradient descent (SGD) and spiking neural networks are trained with bio-inspired spike timing dependent plasticity (STDP). Spiking networks could potentially help in reducing power usage owing to their binary activations. In this work, we use unsupervised STDP in the feature extraction layers of a neural network with instantaneous neurons to extract meaningful features. The extracted binary feature vectors are then classified using classification layers containing neurons with binary activations. Gradient descent (backpropagation) is used only on the output layer to perform training for classification. Surrogate gradients are proposed to perform backpropagation with binary gradients. The accuracies obtained for MNIST and the balanced EMNIST data set compare favorably with other approaches. The effect of the stochastic gradient descent (SGD) approximations on learning capabilities of our network are also explored. We also studied catastrophic forgetting and its effect on spiking neural networks (SNNs). For the experiments regarding catastrophic forgetting, in the classification sections of the network we use a modified synaptic intelligence that we refer to as cost per synapse metric as a regularizer to immunize the network against catastrophic forgetting in a Single-Incremental-Task scenario (SIT). In catastrophic forgetting experiments, we use MNIST and EMNIST handwritten digits datasets that were divided into five and ten incremental sub-tasks respectively. We also examine behavior of the spiking neural network and empirically study the effect of various hyper-parameters on its learning capabilities using the software tool SPYKEFLOW that we developed. We employ MNIST, EMNIST

and N-MNIST data sets to produce our results.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE: INTRODUCTION

Deep learning, i.e., the use of deep convolutional neural networks (DCNN), is a powerful tool for pattern recognition (image classification) and natural language (speech) processing [80][66]. Deep convolutional networks use multiple convolution layers to learn the input data [43] [82] [19]. They have been used to classify the large data set IMAGENET [40] with an accuracy of 96.6% [8]. In this work deep spiking networks are considered [72]. This is a new paradigm for implementing artificial neural networks using mechanisms that incorporate spike-timing dependent plasticity which is a learning algorithm discovered by neuroscientists [24] [56]. Advances in deep learning have opened up a multitude of new avenues that once were limited to science fiction [96]. The promise of spiking networks is that they are less computationally intensive and much more energy efficient as the spiking algorithms can be implemented on a neuromorphic chip such as Intel's LOIHI chip [12] (operates at low power because it runs asynchronously using spikes). Our work is based on the work of Masquelier and Thorpe [58] [57], and Kheradpisheh et al. [35] [34]. In particular a study is done of how such networks classify MNIST image data [46] and N-MNIST spiking data [67]. The networks used in [35] [34] consist of multiple convolution/pooling layers of spiking neurons trained using spike timing dependent plasticity (STDP [83]) and a final classification layer done using a support vector machine (SVM) [29].

1.1 Spike Timing Dependent Plasticity (STDP)

Spike timing dependent plasticity (STDP) [55] has been shown to be able to detect hidden (in noise) patterns in spiking data [57]. Figure 1-1 shows a simple

2 layer fully connected network with N input (pre-synaptic) neurons and 1 output neuron. The spike signals $s_i(t)$ are modelled as being either 0 or 1 in one millisecond increments. That is, 1 msec pulse of unit amplitude represents a spike while a value of 0 represents no spike present. See the left side of the Figure 1-1. Each spike signal has a weight (synapse) associated with it which multiplies the signal to obtain $w_i s_i(t)$ which is called the *post synaptic potential* due to the i^{th} input neuron. These potentials are then summed as

$$V(t) = \sum_{i=1}^{N} w_k s_k(t).$$

$V(t)$ is called the *membrane potential* of the output neuron. At any time t if the membrane potential $V(t)$ is greater than a specified threshold γ, i.e., if

$$\sum_{t=0}^{\tau} V(t) > \gamma,$$

then the output neuron spikes. τ is the entire duration of the simulation. By this we mean that the output neuron produces a 1 msec pulse of unit amplitude. See the right side of Figure 1-1.

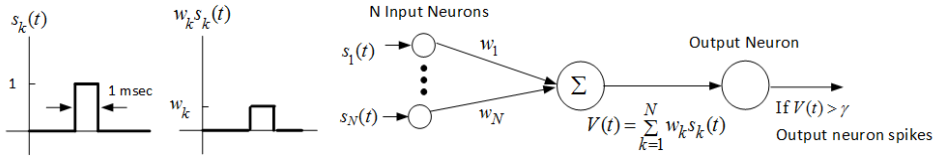

Figure 1-1: The neurons $s_i, i = 1, ..., N$ are the pre-synaptic neurons and the output neuron is the post-synaptic neuron.

Denote the input spike pattern $s(t)$ as

$$s(t) = \begin{bmatrix} s_1(t) \\ s_2(t) \\ \vdots \\ s_N(t) \end{bmatrix}. \tag{1.1}$$

Let $t_1 < t_2 < t_3 < \cdots$ be a sequence of times for which the spike pattern in Equation 1.1 is fixed, that is, $s_{fixed} = s(t_1) = s(t_2) = s(t_3) = \cdots$ while at all other times the values $s_i(t)$ are *random* (E.g., $P(s_i(t) = 1) = 0.01$ and $P(s_i(t) = 0) = 0.99$). The idea here is that the weights can be updated according to an unsupervised learning rule that results in the output spiking if and only if the fixed pattern is present. The learning rule used here is called spike timing dependent plasticity or STDP. Specifically, we used a simplified STDP model as in given as [35]

$$w_i \leftarrow w_i + \Delta w_i, \;\; \Delta w_i = \begin{cases} +a^+ w_i(1 - w_i), & \text{if } t_{out} - t_{in} \leq 0 \\ -a^- w_i(1 - w_i), & \text{if } t_{out} - t_{in} > 0. \end{cases}$$

Here t_{in} and t_{out} are the spike times of the pre-synaptic (input) and the post-synaptic (output) neuron, respectively. That is, if the i^{th} input neuron spikes before the output neuron spikes then the weight w_i is increased otherwise the weight is decreased.[1] Learning refers to the change Δw_i in the synaptic weight w_i with a^+ and a^- denoting the learning rate constants. These rate constants are initialized with low values $(0.004, 0.003)$ and are typically increased as learning progresses. This STDP rule is considered simplified because the amount of weight change doesn't depend on the time duration between pre-synaptic and post-synaptic spikes.

To summarize, if the pre-synaptic (input) neuron spikes before post-synaptic (output) neuron, then the synapse is increased. If the pre-synaptic neuron doesn't spike before the post-synaptic neuron then it is assumed that the pre-synaptic neuron will spike later and the synapse is decreased. The membrane potential profile of the type of output neuron considered here looks as shown in the Figure 1-2. In Figure 1-2 the output neuron is shown to receive a spike at 1 msec, two spikes at 2 msec and another two spikes at 3 msec. The output neuron spikes at time 3 msec as its membrane potential exceeded the threshold ($\gamma = 4.5$).

[1] The input neuron is assumed to have spiked *after* the output neuron spiked.

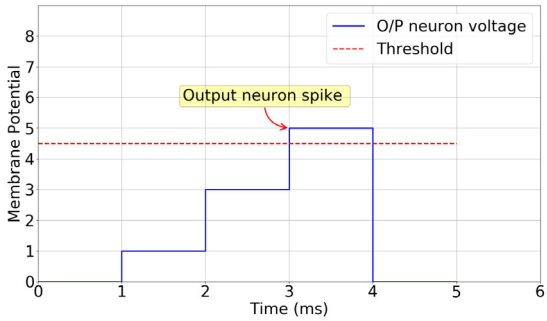

Figure 1-2: Spike generation by an output neuron.

Figure 1-3 shows a raster plot of an input neuron versus its spike times for the first 54 msecs. Figure 1-3 shows $N = 100$ input neurons and at any time t a dot $(*)$ denotes a spike while an empty space denotes no spike. Red dots in the plot indicates a spike as part of the fixed pattern of spikes s_{fixed}. In Figure 1-3 the pattern presented to the output neuron is 5 msec long in duration. The blue part of Figure 1-3 denotes random spikes being produced by the input neurons (noise).

On close observation of Figure 1-3 one can see that fixed spike pattern in red is presented at time 0, time 13, and time 38.

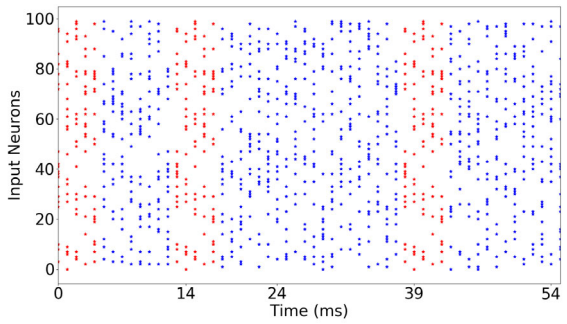

Figure 1-3: The pattern s_{fixed} is red and has a duration of 5 miliseconds. This pattern is presented recurrently to the network at random times. The random noisy spikes are represented in blue.

Using only the above STDP learning rule, the output neuron learns to spike only when the fixed pattern s_{fixed} is produced by the input neurons. With the weights w_i set randomly from normal distribution, i.e., $w_i \sim \mathcal{N}(0.5, 0.05)$ Figure 1-4 (top plot) shows the output spiking for the first 50 msecs. However after about 2000

msec, Figure 1-4 (middle plot) shows the output neuron starts to spike selectively, though it incorrectly spikes at times when the pattern is not present. Finally, after about 3000 msec, Figure 1-4 (bottom plot) shows that the output neuron spikes only when the pattern is present.

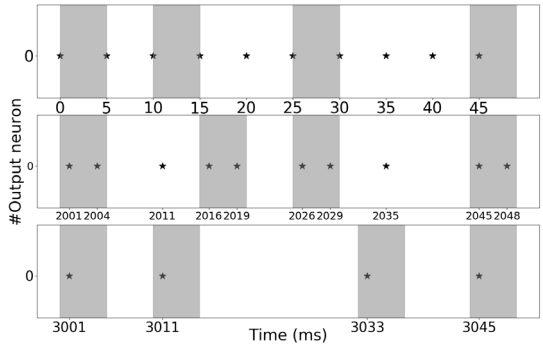

Figure 1-4: The grey box indicates the fixed pattern s_{fixed} is present in the input neurons s_i.

1.2 Convolution Operation

In this work spiking convolutional neural networks (SCNN) are used for feature extraction. A short explanation of convolution is now presented. Figure 1-5 shows a convolution operation on an input image.

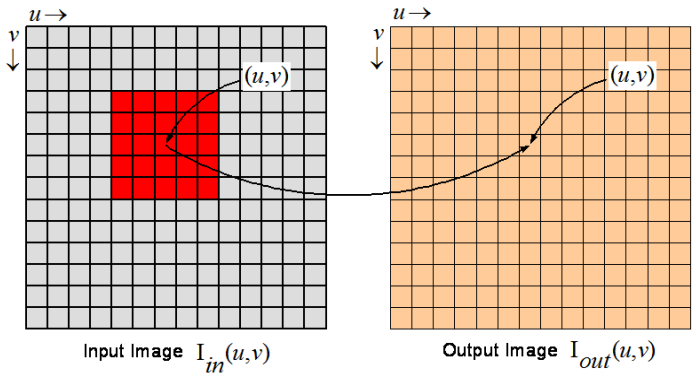

Figure 1-5: Convolution operation.

Let

$$W_C(i,j), \quad 0 \leq i,j \leq 4$$

denote a 5×5 convolution weight kernel (filter) indicated by the red square Figure 1-5 above. With the kernel centered on the location (u, v) of the input image $\mathbf{I}_{in}(u, v)$ ($0 \leq u, v \leq 14$) the value $\mathbf{I}_{out}(u, v)$ ($0 \leq u, v \leq 14$) of the output image at (u, v) is given by

$$\mathbf{I}_{out}(u, v) = \sum_{j=-2}^{j=2} \sum_{i=-2}^{i=2} \mathbf{I}_{in}(u + i, v + j) W_C(i, j).$$

Note that the shape of the output image is same as the input image, such convolutions are called same mode convolutions.

Convolution networks are used to detect features in images. To explain, consider the convolution kernel $W_{C1}(i, j, 1)$ as shown in Figure 1-6. This kernel is used to find vertical lines of spikes at any location of the spiking input image. For example, at the location (u, v) at time τ, the kernel is convolved with the spiking image to give

$$\sum_{j=-2}^{2} \sum_{i=-2}^{2} s_{in}(u + i, v + j, \tau) W_{C1}(i, j, 1).$$

If there is a vertical line of spikes in the spiking image that matches up with the kernel, then this result will be a maximum (maximum correlation of the kernel with the image). The accumulated membrane potential for the neuron at (u, v) of map1 of the Conv1 layer is given by

$$V_m(u, v, t, 1) = \sum_{\tau=0}^{t} \left(\sum_{j=-2}^{2} \sum_{i=-2}^{2} s_{in}(u + i, v + j, \tau) W_{C1}(i, j, 1) \right).$$

The neuron at (u, v) of map 1 of the Conv1 layer then spikes at time t if

$$V_m^{(1)}(u, v, t) \geq \gamma_{C1}$$

where γ_{C1} is the threshold. If the neuron at (u, v) in map 1 of Conv1 spikes, then a vertical line of spikes have been detected in the spiking image centered at (u, v).

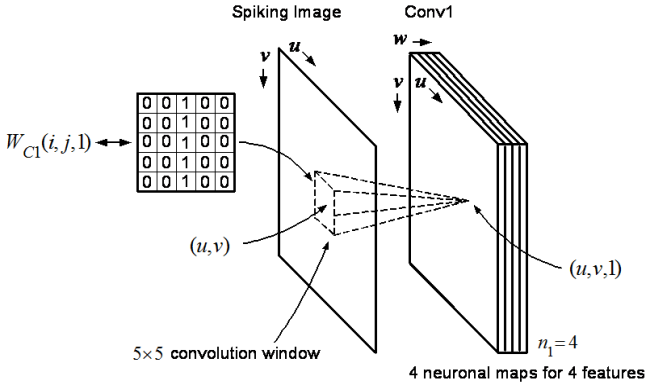

Figure 1-6: Feature detection.

Figure 1-7 shows that map 2 (second feature map) of Conv1 is used to detect a line of spikes at 45 degrees. The third feature map (map 3) is used to detect a line of spikes at 135 degrees and the fourth feature map (map 4) is used to detect a horizontal line of spikes. A typical SCNN has multiple layers. Each layer will have multiple feature maps.

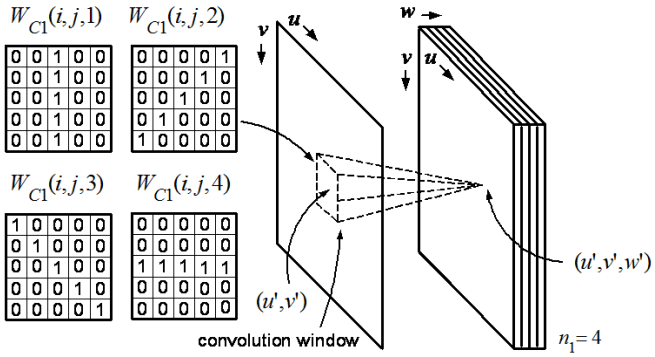

Figure 1-7: Feature detection.

CHAPTER TWO: LITERATURE SURVEY

In 1951 Hubel and Wiesel [30] showed that a cat's neurons in the primary visual cortex are tuned to simple features and the inner regions of the cortex combined these simple features to represent complex features. The neocognitron model was proposed in 1980 by Fukushima to explain this behavior [17]. This model didn't require a "teacher" (unsupervised) to learn the inherent features in the input, akin to the brain. The neocognitron model is a forerunner to the spiking convolutional neural networks considered in this work. These convolutional layers are arranged in layers to extract features in the input data. The terminology "deep" CNNs refers to a network with many such layers. However, the deep CNNs used in industry (Google, Facebook, etc.) are fundamentally different in that they are trained using supervision (back propagation of a cost function). Here our interest is to return to the neocognitron model using spiking convolutional layers in which all but the output layer is trained without supervision.

2.1 Unsupervised Networks

A network equipped with STDP [55] and lateral inhibition was shown to develop orientation selectivity similar to the visual frontal cortex in a cat's brain [13] [101]. STDP was shown to facilitate approximate Bayesian computation in the visual cortex using expectation-maximization [65]. STDP is used for feature extraction in multi-layer spiking CNNs. It has been shown that deeper layers combine the features learned in the earlier layers in order to represent advanced features, but at the same time sparsity of the network spiking activity is maintained [15] [35] [34] [58] [71] [87] [90] [89] [99]. In [14] a fully connected network trained using

unsupervised STDP and homeostasis achieved a 95.6% classification accuracy on the MNIST data set.

2.2 Reward Modulated STDP

Mozafari et al. [61] [63] proposed reward modulated STDP (R-STDP) to avoid using a support vector machine (SVM) as a classifier. It has been shown that the STDP learning rule can find spiking patterns embedded in noise [57]. That is, after unsupervised training, the output neuron spikes if the spiking pattern is input to it. A problem with this unsupervised STDP approach is that as this training proceeds the output neuron will spike when just the first few milliseconds of the pattern have been presented. (For example, the pattern in Figure 1-3 is 5 msecs long and the output starts to spike when only (say) the first 2 msecs of the pattern have been presented to it though it should only spike after the full 5 msec pattern has been presented. Mozafari et al. showed in [63] that R-STDP helps to alleviate this problem.

When unsupervised training methods are used, the features learned in the last layer are used as input to an SVM classifier [34][35] or a simple two or three layer back propagation classifier [86]. In contrast, R-STDP uses a reward or punishment signal (depending upon if the prediction is correct or not) to update the weights in the final layer of a multi-layer (deep) network. Spiking convolutional networks are successful in extracting features [63][34][35]. Because R-STDP is a supervised learning rule, the extracted features (reconstructed weights) more closely resemble the object they detect and thus can more easily differentiate between a digit “1” and a digit "7" compared to STDP. That is, reward modulated STDP seems to compensate for the inability of the STDP to differentiate between features that closely resemble each other [16] [49] [61] [84]. It is also reported in [61] that R-STDP is more computationally efficient. However, R-STDP is prone to over fitting, which is alleviated to some degree by scaling the rewards and punishments, e.g., receiving higher punishment for a false positive and a lower reward for a true positive [61]

[63]. In more detail, the reward modulated STDP learning rule is:

If a reward signal is generated then the weights are updated according to

$$\begin{cases} \Delta w_{ij} = +\frac{N_{miss}}{N} a_r^+ w_{ij}(1 - w_{ij}) & \text{if } t_j - t_i \leq 0 \\ \Delta w_{ij} = -\frac{N_{miss}}{N} a_r^- w_{ij}(1 - w_{ij}) & \text{if } t_j - t_i > 0. \end{cases}$$

If a punishment signal is generated then the weights are updated according to

$$\begin{cases} \Delta w_{ij} = -\frac{N_{hit}}{N} a_p^+ w_{ij}(1 - w_{ij}) & \text{if } t_j - t_i \leq 0 \\ \Delta w_{ij} = +\frac{N_{hit}}{N} a_p^- w_{ij}(1 - w_{ij}) & \text{if } t_j - t_i > 0. \end{cases}$$

Here t_j and t_i are the pre- and post-synaptic times, respectively. For every N input images, N_{miss} and N_{hit} are a number of misclassified and correctly classified samples respectively. Note that $N_{miss} + N_{hit} = N$, if the decision of the network is based on the maximum potential of the network, if the decision of the network is based on the early spike $N_{miss} + N_{hit} \leq N$ because there might be not be any spikes for some inputs.

2.3 Spiking Networks with Backpropagation

In [47] a two layer unsupervised spiking CNN was used for feature extraction. The output of these layers were input to a type of softmax cost function for classification with the error back propagated through all layers. They were able to obtain a classification accuracy 99.1% on the MNIST data set. A similar approach with comparable accuracy was carried by [88]. Other methods such as computing the weights on conventional (non spiking) CNNs trained using the back propagation algorithm and then converting them to work on spiking networks have been shown to achieve an accuracy of 99.4% on MNIST data set and 91.35% on CIFAR10 data set [78]. An approximate back propagation algorithm for spiking neural networks was proposed in [3] [48]. In [32] a spiking CNN with 15C5-P2-40C5-P2-300-10 layers using error back propagation through all the layers reported an accuracy of

99.49% on the MNIST data set. The authors in [32] also classified the N-MNIST data set using a fully connected three-layer network with 800 neurons in the hidden layer and reported an accuracy of 98.84%.

Another approach to back propagation in spiking networks is the *random back* propagation approach. Firstly, the standard back propagation equations in (non-spiking) neural networks are now summarized [66]. The gradient of a quadratic cost $C = \sum_{i=1}^{n_{0ut}} (y - a^L)^2$ gives the error from the last layer as

$$\delta^L = \frac{\partial C}{\partial a^L} \sigma'(z^L). \tag{2.1}$$

a^L is the activation of the neurons in the output layer, σ is the activation function and z is the net input to the output layer. This error on the last layer is back propagated according to

$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{2.2}$$

where W^{l+1} are the weights connecting the l^{th} and $(l+1)^{th}$ layer. The weights and biases are updated as follows:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.3}$$

$$\frac{\partial C}{\partial W_{jk}^l} = a_k^{l-1} \delta_j^l \tag{2.4}$$

In equation (2.2), the weight matrix W^{l+1} connecting the l^{th} and $(l+1)^{th}$ layer is the same as the weight matrix used in forward propagation to calculate the activations a^{l+1} of $(l+1)^{th}$ layer. This is bothersome to the neuroscience community as it is not biologically plausible [50] [22] [76]. This is referred to as the *weight transport problem*. Lillicrap et al. [52] showed that the back propagation algorithm works well even if W^{l+1} in equation (2.2) is replaced with another fixed *random* matrix $(W')^{l+1}$. This eliminates the requirement of weight symmetry, i.e., the same weights for forward and backward propagations. A neuromorphic hardware specific adaptation of random error back propagation that solves the weight transport problem was introduced by [64] and was shown to achieve an error rate of 1.96%

for the MNIST data set. The cost function in [64] is defined as

$$L_{sp} = 0.5 \sum_i (v_i^p(t) - v_i^l(t))^2 \tag{2.5}$$

where $e_i(t)$ is the error of the i^{th} output neuron and v^p and v^l are the firing rates of the prediction neuron and the label neuron.

$$\frac{\partial L_{sp}}{\partial W_{ij}} = -\sum_i e_i(t) \frac{\partial v_i^p(t)}{\partial W_{ij}}. \tag{2.6}$$

In equation (2.6) $\frac{\partial v_i^p(t)}{\partial W_{ij}}$ was approximated as

$$\frac{\partial v_i^p(t)}{\partial W_{ij}} \propto \begin{cases} 1, & \text{if } s_j^h(t) = 1 \text{ and } b_{\min} < I_i(t) < b_{\max} \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

where $I_i(t)$ is the current entering into i^{th} post-synaptic neuron and $s_j^h(t) = 1$ indicates the presence of a pre-synaptic spike. For more details see [64]. The weight update for the last layer is then

$$\Delta W_{ij}^E \propto \begin{cases} -e_i(t), & \text{if } s_j^h(t) = 1 \text{ and } b_{\min} < \ I_i(t) < b_{\max} \\ 0, & \text{otherwise.} \end{cases} \tag{2.8}$$

The weight update for hidden layers is

$$\Delta w_{ij}^C \propto \begin{cases} -\sum_k g_{ik} e_k^E(t), & \text{if } s_j^C(t) = 1 \text{ and } b_{\min} < \ I_i(t) < b_{\max} \\ 0, & \text{otherwise.} \end{cases} \tag{2.9}$$

where $e_k^E(t)$ denotes the error term of the k^{th} neuron in the output layer and g_{ik} is a fixed random number as suggested by the random back propagation algorithm. In the work to be reported below, random back propagation was not used. Specifically, when back propagation is used below, it is only between the penultimate and output

layer making random back propagation unnecessary.

2.4 Spike Encoding

Spikes are either rate coded or latency coded [21] [38] [75] [6]. Rate coding refers to the information encoded by the number of spikes per second (more spikes per time carries more information). In this case the spike rate is determined by the mean rate of a Poisson process. Latency encoding refers to the information encoded in the time of arrival of a spike (earlier spikes carry more information). The raster plot of Figure 1-3 shows that spatiotemporal information is provided by the input spikes to the output neuron. That is, which input neuron is spiking (spatio) and the time a neuron spikes (temporal) is received by the output neuron. The spiking networks use this spatiotemporal information to extract features (e.g., detect the pattern in Figure 1-3) in the input data [23] [60].

2.5 Realtime Spikes

Image sensors (silicon retinas) such as ATIS [73] and eDVS [10] [51] provide (latency encoded) spikes as their output. These sensors detect changes in pixel intensities. If the pixel value at location (u, v) increases then an ON-center spike is produced while if the pixel value decreased an OFF-center spike is produced. Finally, if the pixel value does not change, no spike is produced. The spike data from an image sensor is packed using an address event representation (AER [31]) protocol and can be accessed using serial communication ports. A recorded version of spikes from eDVS data set was introduced in [53] and a similar data set of MNIST images recorded with ATIS data set was introduced in [67].

CHAPTER THREE: BACKGROUND

3.1 Spiking Images

We have considered the standard 27×27 grey-scale MNIST images[1] [46] and the spiking N-MNIST data files [67] for our experiments. In the case of the MNIST images we needed to convert them to spikes. This was done by first using both an on-center and an off-center Difference of Gaussian (DoG) convolution filter $\Gamma_{\sigma_1,\sigma_2}(i,j)$ for edge detection given by

$$K_{\sigma_1,\sigma_2}(i,j) = \begin{cases} \frac{1}{2\pi\sigma_1^2} e^{-\frac{i^2+j^2}{2\sigma_1^2}} - \frac{1}{2\pi\sigma_2^2} e^{-\frac{i^2+j^2}{2\sigma_2^2}} & \text{for } -3 \leq i \leq 3, -3 \leq j \leq 3 \\ 0 & \text{otherwise} \end{cases}$$

where $\sigma_1 = 1, \sigma_2 = 2$ for the on-center and $\sigma_1 = 2, \sigma_2 = 1$ for the off-center.

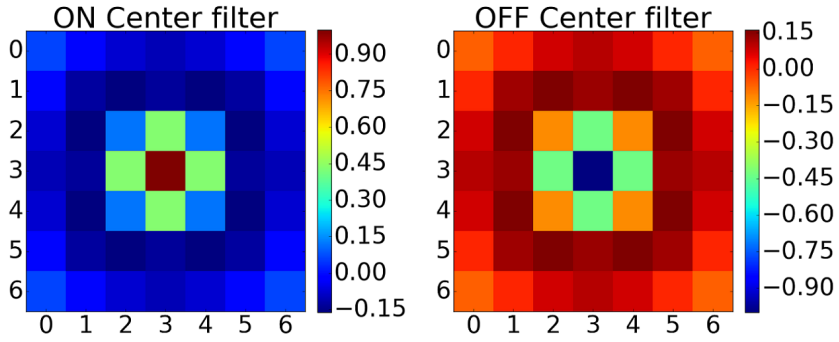

Figure 3-1: On center filter has higher values in the center whereas the off center filter has lower values in the center. Color code indicates the filter values.

With the input image $\mathbf{I}_{in}(u,v) \in \mathbb{R}^{27\times 27}$, the output of each of the two DoG

[1] We removed the outer most pixels in the data set [46] giving 27×27 images.

filters is computed using the *same* mode convolution

$$\Gamma_{\sigma_1,\sigma_2}(u,v) = \sum_{j=-3}^{j=3}\sum_{i=-3}^{i=3} \mathbf{I}_{in}(u+i, v+j)K_{\sigma_1,\sigma_2}(i,j) \text{ for } 0 \leq u \leq 26, 0 \leq v \leq 26.$$

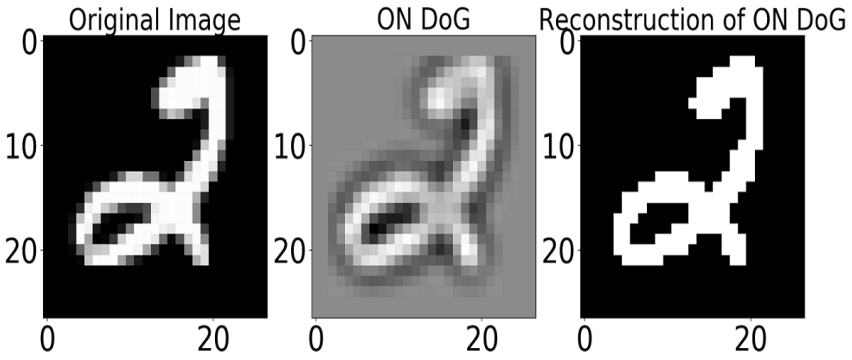

Figure 3-2: Left: Original grey-scale image. Center: Output of the ON DoG filter. Right: Accumulation of spikes (white indicates a spike, black indicates no spike).

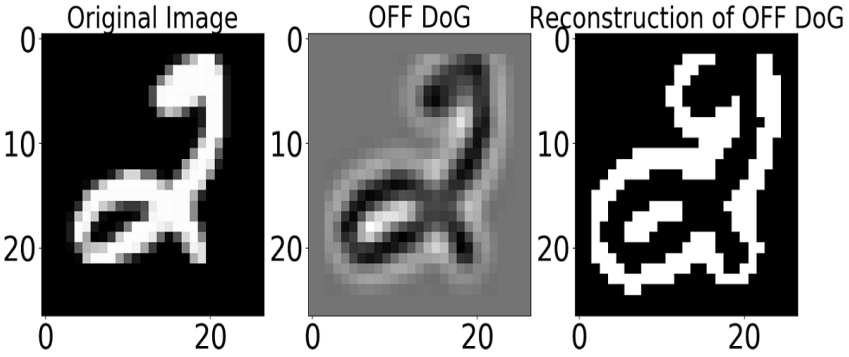

Figure 3-3: Left: Original grey-scale image. Center: Output of the OFF DoG filter. Right: Accumulation of spikes (white indicates a spike, black indicates no spike).

Then these two resulting "images" were then converted to an on and an off spiking image as follows: At each location (u, v) of the output image $\Gamma_{\sigma_1,\sigma_2}(u, v)$ a unit spike $s_{(u,v)}$ is produced if and only if [33]

$$\Gamma_{\sigma_1,\sigma_2}(u,v) > \gamma_{DoG} = 50.$$

The spike signal $s_{(u,v)}(t)$ is temporally coded (rank order coding [13]) by having

it delayed "leaving" the Difference of Gaussian image $\Gamma_{\sigma_1,\sigma_2}(u,v)$ by the amount

$$\tau_{(u,v)} = \frac{1}{\Gamma_{\sigma_1,\sigma_2}(u,v)} \text{ in milliseconds.}$$

That is, the more $\Gamma_{\sigma_1,\sigma_2}(u,v)$ exceeds the threshold γ_{DoG} the sooner it leaves $\Gamma_{\sigma_1,\sigma_2}(u,v)$ or equivalently, the value of $\Gamma_{\sigma_1,\sigma_2}(u,v)$ is encoded in the value $\tau_{(u,v)}$.

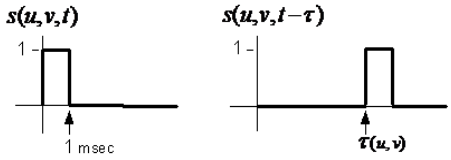

Figure 3-4: Spike signal

For all experiments the arrival times of the spikes were sorted in ascending order and then (approximately) equally divided into 10 bins (10 times in Figure 3-5). The raster plot shows which neurons (pixels of $\Gamma_{\sigma_1,\sigma_2}(u,v)$) spiked to make up bin 1 (time 0), bin 2 (time 1), etc. Figure 3-5 shows an example for ON center cell spikes. In all the experiments each image is encoded into 10 msec (10 bins) and there is a 2 msec silent period between every image.

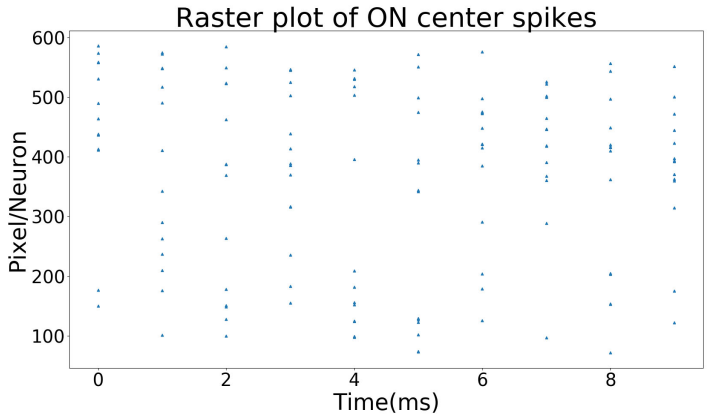

Figure 3-5: Rasterplot of spikes for an on center cell. Blue dots in the plot indicates the presence of a spike for a particular neuron and bin (timestep).

3.2 Network Description

We have a similar network as in [35][34] as illustrated in Figure 3-6. We let $s_{L1}(t,k,u,v)$ denote the spike signal at time t emanating from the (u,v) neuron of spiking image k where $k=0$ (ON center) or $k=1$ (OFF center). The L2 layers consists of 30 maps with each map having its own convolution kernel (weights) of the form

$$W_{C1}(w,k,i,j) \in \mathbb{R}^{2\times 5\times 5} \text{ for } w=0,1,2,...,29.$$

The "membrane potential" of the (u,v) neuron of map w ($w=0,1,2,...,29$) of L2 at time t is given by the *valid* mode convolution

$$V_{L2}(t,w,u,v) = \sum_{\tau=0}^{t}\left(\sum_{k=0}^{1}\sum_{i=0}^{4}\sum_{i=0}^{4} s_{L1}(\tau,k,u+i,v+j)W_{C1}(w,k,i,j)\right)$$
$$\text{for } (0,0) \le (u,v) \le (22,22)$$

If at time t the potential

$$V_{L2}(t,w,u,v) > \gamma = 15$$

then the neuron at (w,u,v) emits a unit spike.

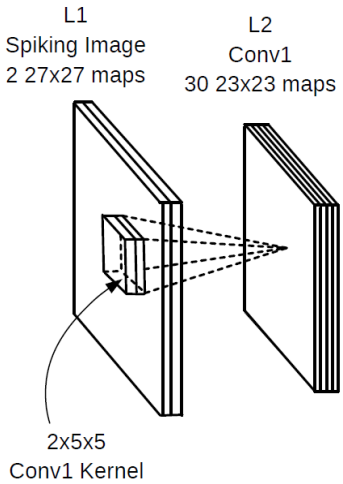

Figure 3-6: Demonstration of convolution with a 3D kernel.

3.2.1 Convolution Layers and STDP

At any time t, *all* of the potentials $V_{L2}(t, w, u, v)$ for $(0, 0) \leq (u, v) \leq (22, 22)$ and $w = 0, 1, 2, ..., 29$ are computed (in theory this can all be done in parallel) with the result that neurons in different locations within a map and in different maps may have spiked. In particular, at the location (u, v) there can be multiple spikes (up to 30) produced by different maps. The desire is to have different maps learn different features of an image. To enforce this learning, *lateral inhibition* and *STDP competition* are used [35].

Lateral Inhibition

To explain lateral inhibition suppose at the location (u, v) there were potentials $V_{L2}(t, w, u, v)$ in different maps (w goes from 0 to 29) at time t that exceeded the threshold γ. Then the neuron in the map with the highest potential $V_{L2}(t, w, u, v)$ at (u, v) inhibits the neurons in all the other maps at the location (u, v) from spiking for the current image (even if the potentials in the other maps exceeded the threshold). Figure 3-7 (left) shows the accumulated spikes (from an MNIST image of "5") from all 30 maps of Layer $L2$ at each location (u, v) *without* lateral inhibition. For example, at location (19,14) in Figure 3-7 (left) the color code is yellow indicating in excess of 20 spikes, i.e., more than 20 of the maps produced a spike at that location.

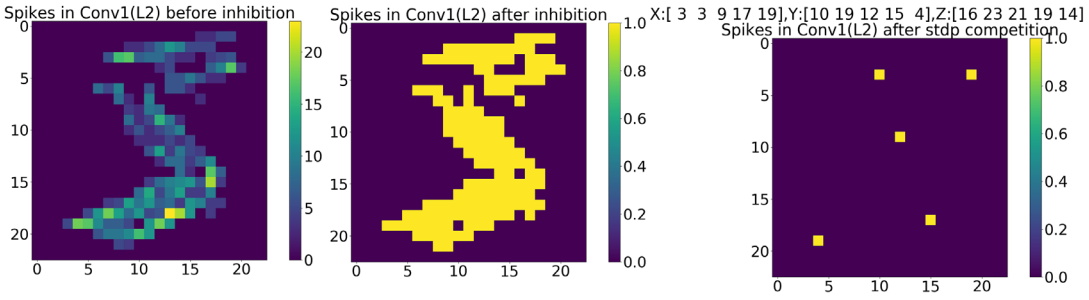

Figure 3-7: Left: MNIST digit "5" input. Accumulation of spikes from all 30 maps and 12 time steps in L2 *without* lateral inhibition. Center: Accumulation of spikes from all 30 maps and all 12 time steps in L2 *with* lateral inhibition. Right: Accumulation of spikes across all maps and 12 time steps with both lateral inhibition and STDP competition imposed for a single image.

Figure 3-7 (center) shows the accumulation of spikes from all 30 maps, but now *with* lateral inhibition imposed. Note that at each location there is at most one spike indicated by the color code. Also, as explained next, only a few of these spikes will actually result in the update of any of the 30 kernels (weights) of layer L2.

STDP Competition

After lateral inhibition we consider each of the maps in layer L2 that had one or more neurons with their potential V exceeding γ. Let these maps be $w_{k1}, w_{k2}, ..., w_{km}$ where[2] $0 \leq k_1 < k_2 < \cdots < k_m \leq 29$. Then in each map w_{ki} we locate the neuron in that map that has the maximum potential value. Let

$$(u_{k1}, v_{k1}), (u_{k2}, v_{k2}), ..., (u_{km}, v_{km}) \tag{3.1}$$

be the location of these maximum potential neurons in each map. Then the neuron (u_{ki}, v_{ki}) inhibits all other neurons in its map w_{ki} from spiking for the remainder of the time steps of the current spiking image. Further, these m neurons can inhibit each other depending on their relative location as we now explain. Suppose the neuron (u_{ki}, v_{ki}) of map w_{ki} has the highest potential of the m neurons in (3.1). Then, in an 11×11 area centered about (u_{ki}, v_{ki}), this neuron inhibits all neurons of all the other maps in the same 11×11 area. Next, suppose the neuron (u_{kj}, v_{kj}) of map w_{kj} has the second highest potential of the remaining $m-1$ neurons. If the location (u_{kj}, v_{kj}) of this neuron was within the 11×11 area centered on neuron (u_{ki}, v_{ki}) of map w_{ki}, then it is inhibited. Otherwise, this neuron at (u_{kj}, v_{kj}) inhibits all neurons of all the other maps in a 11×11 area centered on it. This process is continued for the remaining $m - 2$ neurons. In summary, there can be no more than one neuron that spikes in the same 11×11 area of all the maps[3]. The right side of Figure 3-7 shows the spike accumulation after both lateral inhibition and STDP competition

[2]The other maps did not have any neurons whose membrane potential crossed the threshold and therefore did not spike.

[3]The use of the number 11 for the 11×11 inhibition area of neurons was suggested by Dr. Kheradpisheh [33].

have been imposed. It is also shown that there is at most one spike from all the maps in any 11×11 area. For this particular input image (the number 5), these five spikes are from maps 14, 16, 19, 21, and 23 at locations (19, 4), (3,10), (17, 15), (9,12) and (3,19), respectively and will result in updates for these 5 map kernels (weights). Lateral Inhibition and STDP inhibition enforce sparse spike activity and, as a consequence, the network tends to spike sparsely. This lateral inhibition and STDP competition resulted in an average of only *5.8 spikes per image* from the $30 \times 22 \times 22$ neurons in $L2$ during training with EMNIST and MNIST datasets.

Spike Timing Dependent Plasticity (STDP)

Only those maps that produced a spike (with lateral inhibition and STDP competition imposed) have their weights (convolution kernels) updated using spike timing dependent plasticity. Let w_{ij} be the weight connecting the j^{th} pre-synaptic neuron in the L1 layer to i^{th} post-synaptic neuron in the L2 layer. If the i^{th} post-synaptic neuron spikes at time t_i with the pre-synaptic neuron spiking at time t_j then the weight w_{ij} is updated according to the simplified STDP rule [13]

$$w_{ij} \longleftarrow w_{ij} + \Delta w_{ij}, \text{ where } \Delta w_{ij} = \begin{cases} +a^{+} w_{ij}(1 - w_{ij}) \text{ if } t_i > t_j \\ -a^{-} w_{ij}(1 - w_{ij}) \text{ otherwise.} \end{cases}$$

The parameters $a^{+} > 0$ and $a^{-} > 0$ are referred to as learning rate constants. a^{+} is initialized to 0.004 and a^{-} is initialized to 0.003 and are increased by a factor of 2 after every 1000 spiking images. STDP is shown to detect a hidden pattern in the incoming spike data [57]. In all of our experiments we used the above simplified STDP model as in [35] (simplified STDP refers to the weight update not depending on the exact time difference between pre-synaptic and post-synaptic spikes). If the pre-synaptic neuron spikes before post-synaptic neuron then the synapse is strengthened, if the pre-synaptic neuron doesn't spike before post-synaptic neuron then it is assumed that the pre-synaptic neuron will spike later and the synapse is weakened.

Figure 3-8 is a plot of the weights (convolution kernels) for each of the 30 maps. Following [35], each column corresponds to a map and each row presents the weights after every 500 images. For example, $W_{C1}(29, k, i, j)$ for $k = 0, 1$ and $(0, 0) \leq (i, j) \leq (26, 26)$ are the weights for the ON (green) and OFF (red) filters[4] for the 30^{th} map (right-most column of Figure 3-8). It turned out that there were approximately 17 spikes per image in this layer (L2). At the end of the training most of the synapses will be saturated either at 0 or 1.

Figure 3-8: Plot of the weights of 30 maps of L2. The ON (green) 5×5 filter and the OFF (red) 5×5 filter are superimposed on top of each other.

Homeostasis

Homeostasis refers to the convolution kernels (weights) for all maps being updated approximately the same number of times during training. With homeostasis each kernel gets approximately the same number of opportunities to learn its unique feature. Some maps tend to update their weights more than others and, if this continues, these maps can take over the learning. That is, only the features (weights of the convolution filter) of those maps that get updated often will be of value with

[4] That is, the ON (green) and Off (red) weight are superimposed on the same plot.

the rest of the maps not learning any useful feature (as their weights are not updated). Homeostasis was enforced by simply decreasing the weights of a map by $w_{ij} \rightarrow w_{ij} - a^{-} w_{ij}(1 - w_{ij})$ if it tries to update more than twice for every 5 of input images.

3.2.2 Pooling Layers

A pooling layer is a way to down sample the spikes from the previous convolution layer to reduce the computational effort.

Max Pooling

After the synapses (convolution kernels or weights) from L1 to L2 have been learned (unsupervised STDP learning is over[5]), they are fixed, but lateral inhibition continues to be enforced in L2. Spikes from the maps of the convolution layer L2 are now passed on to layer L3 using max pooling. First of all, we ignored the last row and last column of each of the 23×23 maps of L2 so that they may be considered to be 22×22. Next, consider the first map of the convolution layer L2. This map is divided into non-overlapping 2×2 area of neurons. In each of these 2×2 sets of neurons, at most one spike is allowed through. If there is more than one spike coming from the 2×2 area, then one compares the membrane potentials of the spikes and passes the one with the highest membrane potential. Each 2×2 set of neurons in the first map is then a single neuron in the first map of the L3 layer. Thus each map of L3 has 11×11 (down sampled) neurons. This process is repeated for all the maps of L2 to obtain the corresponding maps of L3. Lateral inhibition is not applied in a pooling layer. There is no learning done in the pooling layer, it is just a way to decrease the amount of data to reduce the computational effort.

After training the L2 convolution layer, we then passed 60,000 MNIST digits through the network and recorded the spikes from the L3 pooling layer. This is shown in Figure 3-9. For example, in the upper left-hand corner of Figure 3-9 is

[5] And therefore STDP competition is no longer enforced.

shown the number of spikes coming out of the first map of the pooling layer L3 for each of the 10 MNIST digits. It shows that the digit “3” produced over 100,000 spikes when the 60,000 MNIST digits were passed through the network while the digit “1” produced almost no spikes. That is, the spikes coming from digit “1” do not correlate with the convolution kernel (see the inset) to produce a spike. On the other hand, the digit "3" almost certainly causes a spike in the first map of the L3 pooling layer. In the bar graphs of Figure 3-9 the red bars are the five MNIST digits that produced the most spikes in the L3 pooling layer while the blue bars are the five MNIST digits that produced the least.

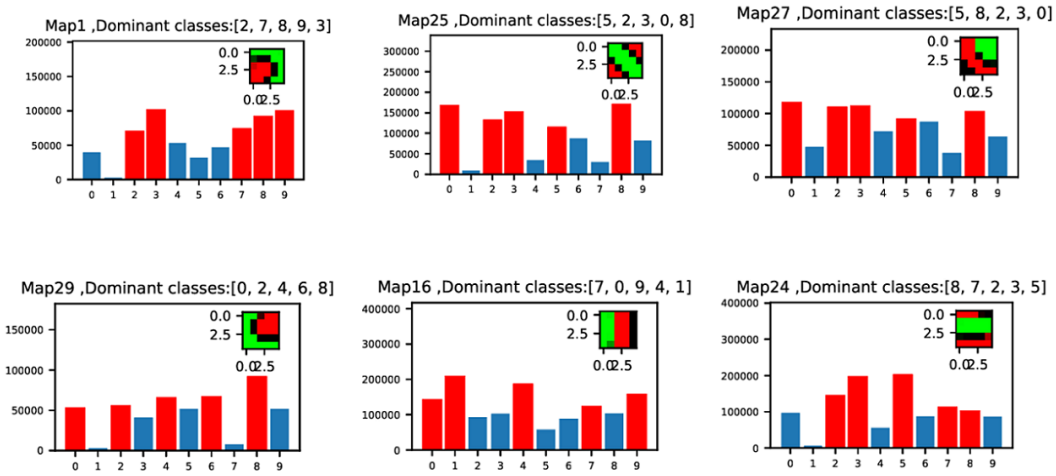

Figure 3-9: Spikes per map per digit. Headings for each of the sub-plots indicate the dominant (most spiking) digit for respective features.

Figure 3-10 shows a convolution kernel between the L3 pooling layer and the L4 convolution layer. We chose to have 500 maps in L4 which means that for $w = 0, 1, 2, ..., 499$ we have

$$W_{C2}(w, k, i, j) \in \mathbb{R}^{30\times5\times5} \text{ for } 0 \leq k \leq 29 \text{ and } (0, 0) \leq (i, j) \leq (4, 4).$$

The spikes from the L3 pooling layer are then used to train the weights (convolutional kernels) W_{C2} in the same manner as W_{C1}.

In some of our experiments we simply did a type of global pooling to go to the output layer L5. Specifically, at each time step, we convolve the spikes from L3 to

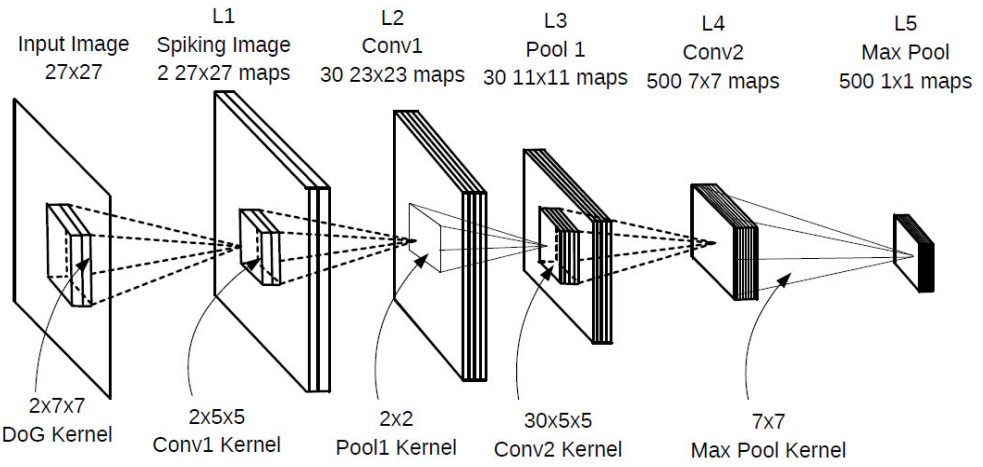

Figure 3-10: Network showing two convolution layers and a final global pooling layer.

compute the potential for each of the $500 \times 7 \times 7$ neurons of L4. The maximum potential for each map in L4 was then found and stored (This is a vector in $\mathbb{R}^{500}$). The potentials in L4 were then reset to 0 and the process repeated for each of the remaining time steps of the current image. This procedure results in ten $\mathbb{R}^{500}$ vectors for each image. The sum of these vectors then encodes the current image in L5, i.e., as a single vector in $\mathbb{R}^{500}$. The motivation to take the maximum potential of each map at each time step is because all the neurons in a given map of L4 are looking for the *same* feature in the current image. Unsupervised STDP training is done in the convolution layers with both STDP competition and lateral inhibition applied to the maps of the convolution layer doing training. Once a convolution layer is trained, its weights are fixed and the spikes are passed through it with only lateral inhibition imposed.

CHAPTER FOUR: CLASSIFICATION OF THE MNIST DATA SET

In the following subsections we considered two different network architectures along with different classifiers for the MNIST data set.

4.1 Classification with Two Convolution/Pool Layers

In this first experiment the architecture shown in Figure 3-10 was used. Max pooled "membrane potentials", i.e., the L5 layer of Figure 3-10, was used to transform each 27×27 $(= 729)$ training image into a new "image" in R^{500}. Using these images along with their labels, a support vector machine [29] was then used to find the hyperplanes that optimally[1] separate the training digits into 10 classes. With $W \in \mathbb{R}^{45\times500}$ the SVM weights, the quantity $\lambda W^T W$ was added to the SVM Lagrangian for regularization. Both linear and radial basis function (RBF) kernels were used in the SVM. We used 20,000 MNIST images for the (unsupervised) training of the two convolution/pool layers (Layers L2-L5). Then we used 50,000 images to train the SVM with another 10,000 images used for validation (to determine the choice of λ). The SVM gives the hyperplanes that optimally separate the 10 classes of digits. Table 4.1 shows classification accuracies when 500 maps were used in L4. The first two rows of Table 4.2 give the test accuracy on 10,000 MNIST test images. In particular, note a 98.01 % accuracy for the RBF SVM and a 97.8 % accuracy for a Linear SVM. Using a similar network with linear SVM, Kheradpisheh et al. [35] reported an accuracy of 98.3%.

[1] It is optimal in the sense that a Lagrangian was minimized.

Table 4.1: Classification accuracies on MNIST data set with various classifiers when number of maps in L4 is 500.

Classifier	Test Acc	Valid Acc	Training Time	λ	η	Epochs
RBF SVM	97.92 %	97.98 %	8 minutes	1/3.6	-	-
Linear SVM	97.27 %	97.30 %	4 minutes	1/0.012	-	-
2 Layer FCN (backprop)	96.90 %	97.02 %	15 minutes	1.0	$\frac{0.1}{(1.007)^{\#Epoch}}$	30
3 layer FCN (backprop)	97.8 %	97.91 %	50 minutes	6.0	$\frac{0.1}{(1.007)^{\#Epoch}}$	30

For comparison purposes with SVM, we also considered putting the L5 neurons (i.e., vectors in $\mathbb{R}^{500}$) into both a conventional two and three layer fully connected network (FCN). Using a two layer FCN (see Figure 4-1) with sigmoidal outputs, a cross-entropy cost function, and a learning rate $\eta = 0.1/(1.001)^{\#Epoch}$ we obtained 97.97 % classification accuracy. Similarly with a three layer FCN (see Figure 4-2) with the same conditions an accuracy of 98.01 % was obtained.

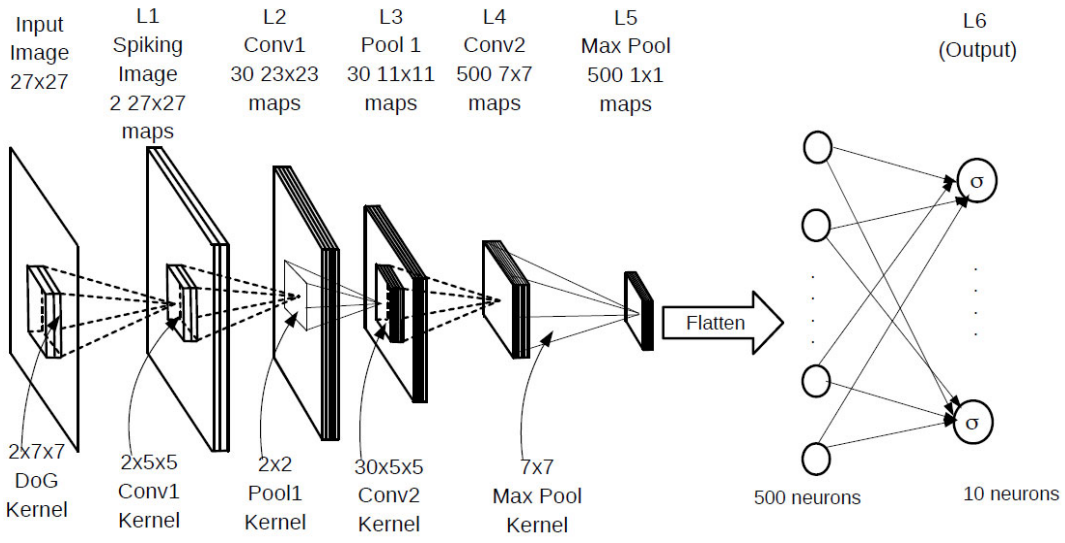

Figure 4-1: Network with two fully connected layers as a classifier.

Separability of the MNIST Set

With $\lambda = 1/1000$ the 50,000 training and 10,000 validation images converted to $\mathbb{R}^{500}$ "images" turn out to be completely separable into the 10 digit classes! However, the accuracy on the 10,000 test images drops to 97.01%. The original 60,000 MNIST (training & validation) images in R^{784} are not separable by a linear SVM

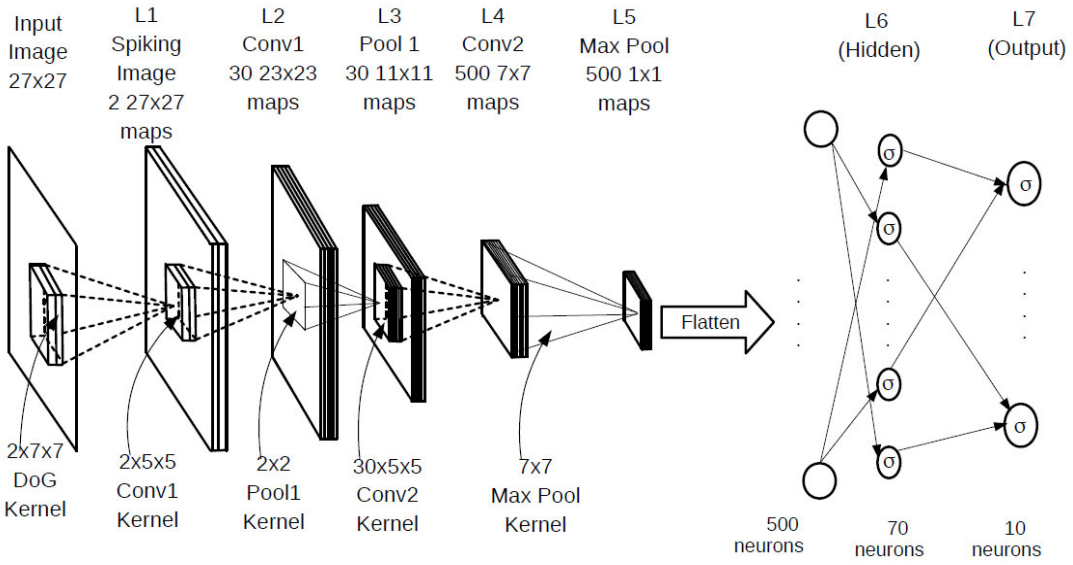

Figure 4-2: Network with three fully connected layers as a classifier.

(The SVM code was run for 16 hours with $\lambda = 1/1000$ without achieving separability).

Increasing the Number of Output Maps

If the number of maps in the L4 layer are increased to 1000 with the L5 1×1 maps correspondingly increased to 1000, then there is a slight increase in test accuracy as shown in Table 4.2. With $\lambda = 1$ the 50,000 training and 10,000 validation images converted to $\mathbb{R}^{1000}$ "images" also turn out to be completely separable into the 10 digit classes. However, with $\lambda = 1$ the test accuracy decreases to 97.61.

Table 4.2: Classification accuracies on MNIST data set with various classifiers when number of maps in L4 is 1000.

Classifier	Test Acc	Valid Acc	Training Time	λ	η	Epochs
RBF SVM	98.01 %	98.20 %	8 minutes	1/3.6	-	-
Linear SVM	97.80 %	98.02 %	4 minutes	1/0.012	-	-
2 Layer FCN (backprop)	97.71 %	98.74 %	15 minutes	1.0	$\frac{0.1}{(1.007)^{\#Epoch}}$	30
3 layer FCN (backprop)	98.01 %	98.10 %	50 minutes	6.0	$\frac{0.1}{(1.007)^{\#Epoch}}$	30

4.2 Classification with a Single Convolution/Pool Layer

The architecture shown in Figure 4-3 has a single convolutional/pooling layer with $30 \times 11 \times 11 = 3630$ pooled neurons in L3. Further, each neuron in L3 simply sums the spikes coming into it from the previous layer (L2). The L4 (output) neurons are fully connected (with trainable weights) to L3 neurons. This final layer of weights are then trained using backprop only on this output layer, i.e., only backprop to L3. (See Lee at al. [47] where the error is back propagated through all the layers and reported an accuracy of 99.3%). Inhibition settings are same as in the above experiment.

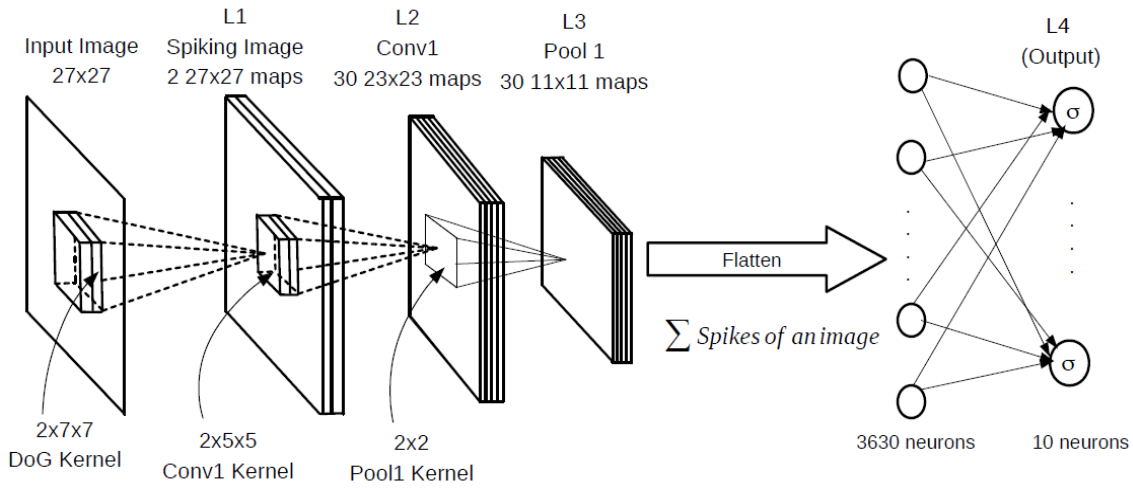

Figure 4-3: Deep spiking convolutional network architecture for classification of the MNIST data set.

The first row of Table 4.3 shows a 98.4% test accuracy using back propagation on the output layer (2 Layer FCN). The second and third rows give the classification accuracy using an SVM trained on the L4 neurons (their spike counts). The feature extraction that takes place in the L2 layer (and passed through the pooling layer) results in greater than 98% accuracy with a two layer conventional FCNN output classifier. A conventional FC two layer NN (i.e., no hidden layer) with the 28×28

images of the MNIST data set as input has only been reported to achieve 88% accuracy and 91.6% with preprocessed data [44]. This result strengthens our view that the unsupervised STDP can transform the MNIST classes into linearly separable classes. Note that the increase in linear separability was also observed when the MNIST classes were transformed to a lower dimension ($\mathbb{R}^{500}$) when compared to original MNIST dimensions ($\mathbb{R}^{784}$, see Chapter 4.1). We also counted the spikes in network with two convolution/pool layers (see Figure 3-10) but found that the accuracy decreased (see Table 4.2) This decrease may be due to reduced number of spikes in the output neurons compared to have only one convolution/pool layer.

Table 4.3: Classification accuracies on MNIST data set with various classifiers when a single convolution/pool layer is used.

Classifier	**Test Acc**	**Valid Acc**	**Training Time**	λ	η	**Epochs**
2 Layer FCN	98.4%	98.5%	10mins	$1/10$	$0.1/(1.007)^{\#Epoch}$	20
RBF SVM	98.8%	98.87%	150 minutes	$1/3.6$	-	-
Linear SVM	98.41%	98.31%	100 minutes	$1/0.012$	-	-

In this chapter, we showed that the original MNIST dataset $\mathbb{R}^{50000\times784}$ is not linearly separable. However when MNIST dataset is transformed to $\mathbb{R}^{50000\times500}$ by passing it through an unsupervisedly trained SNN we showed that the MNIST data becomes linearly separable.

CHAPTER FIVE: REWARD MODULATED STDP

Reward modulated STDP is a way to use the accumulated spikes at the output to do the final classification (in contrast to SVM and a two layer backprop mentioned above). Figure 5-1 shows the network architecture where the reward modulated STDP is carried out between the (flattened) L5 layer and the ten output neurons of the L6 layer. The weights between the fully connected neurons of Layer 5 and Layer 6 are then trained as follows: For any input image the spikes through the network arrive between $t = 0$ and $t = 11$ time steps. The final ($t = 11$) membrane potential of the k^{th} output neuron for $k = 1, 2, ..., 10$ is then

$$V_k = \sum_{t=0}^{11} \sum_{j=1}^{12000} w_{kj} s_{L5}(t, j).$$

Denote by N_{hit} and N_{miss} the number of correctly classified and incorrectly classified images for every N (e.g., $N = 100, 500, 1500,$ etc.) input images so $N_{miss} + N_{hit} = N$. If the k^{th} output potential V_k is maximum (i.e., $V_k > V_j$ for $j \neq k$) and the input image has label k then the weights going into the k^{th} output neuron are rewarded in the sense that

$$w_{kj} \longleftarrow w_{kj} + \Delta w_{kj}, \tag{5.1}$$

$$\text{where } \Delta w_{kj} = \begin{cases} +\dfrac{N_{miss}}{N} a_r^+ w_{kj}(1 - w_{kj}) & \text{if at least one pre-synaptic spike from j to k.} \\ -\dfrac{N_{miss}}{N} a_r^- w_{kj}(1 - w_{kj}) & \text{otherwise.} \end{cases} \tag{5.2}$$

If V_k is the maximum potential, but the label of the image is $j \neq k$, then the weights going into output neuron k are punished in the sense that

$$w_{kj} \longleftarrow w_{kj} + \Delta w_{kj}, \tag{5.3}$$

$$\text{where } \Delta w_{kj} = \begin{cases} -\dfrac{N_{hit}}{N} a_p^+ w_{kj}(1 - w_{kj}) & \text{if at least one pre-synaptic spike from j to k.} \\ +\dfrac{N_{hit}}{N} a_p^- w_{kj}(1 - w_{kj}) & \text{otherwise.} \end{cases} \tag{5.4}$$

Note that only the weights of those neurons connected to the output neuron with the maximum potential are updated. The term "modulated" in reward modulated STDP refers to the factors $\frac{N_{miss}}{N}$ and $\frac{N_{hit}}{N}$ which multiply (modulate) the learning rule. Equation (5.1) refers to the case where the kth output neuron also has the high membrane potential of the ten outputs. If N_{miss}/N is small then the network accuracy is performing well in terms of accuracy and the change in weights is small (as the weights are thought to already have learned to correctly classify). On the other hand, equation (5.3) refers to the case where the kth output has the highest membrane potential, but the label is $j \neq k$. Then, if N_{miss}/N is small, it follows that N_{hit}/N is large the weights of the neurons going into the kth neuron have their values changed by a relatively large amount to (hopefully) correct the misclassification.

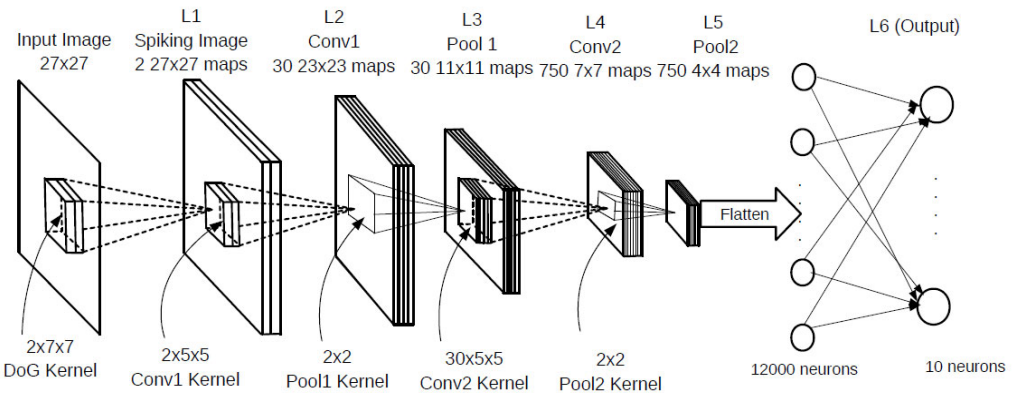

Figure 5-1: Network with 750 maps in L4.

In this experiment with R-STDP, only 20,000 MNIST digits were used for training, 10,000 digits for validation (used to choose the number of training epochs), and the 40,000 remaining digits were used for testing. The R-STDP synaptic weights between L5 and L6 were initialized using the normal distribution $\mathcal{N}(0.8, 0.01)$. Table 5.1 shows that a test accuracy of only 90.1% was obtained.

Table 5.1: Classification accuracy on MNIST data set with R-STDP when one neuron per class is used.

Maps in L4	Valid acc %	Test Acc %	Epochs
750	91.2	90.1	150

For comparison, we replaced the R-STDP classifier (from L5 to L6) with a simple 2 layer neural network (from L5 to L6) which used error back propagation. These weights for back propagation were initialized from the normal distribution $\mathcal{N}(0, 1/\sqrt{12000})$ as in [66]. Tables 5.2 and 5.3 show difference in performance between R-STDP and a simple two layer backprop which ran for only 20 epochs.

Table 5.2: Classification accuracy on MNIST data set with single layer backprop.

Classifier	Test Acc	Valid Acc	λ	η	Epochs
2 Layer FCN	97.5%	97.6%	1.0	$0.1/(1.007)^{\#Epoch}$	20

Mozafari et al. [63][61] got around this poor performance by having 250 neurons in the output layer and assigning 25 output neurons per class. They reported a 97.2 % test accuracy while training on 60,000 images and testing on 10,000 images. We also considered multiple neurons per class in the output layer. As Table 5.3 shows, we considered 300 output neurons (30 per class) and also used dropout. $P_{drop} = 0.4$ means that $0.4(300) = 120$ output neurons were prevented from updating their weights for the particular training image. For each input image a different set of 120 randomly neurons were chosen to not have their weights updated. Table 5.3 shows that the best performance of 95.91 % test accuracy was obtained with $P_{drop} = 0.4$.

Table 5.3: Classification accuracy on MNIST data set with R-STDP when more than one neuron per class is used.

Maps in L4	#Output Neurons	P_{drop}	Valid acc %	Test acc %	Epochs
750	300	0.3	95.81	95.84	400
750	300	0.4	96.01	95.91	400
750	300	0.5	95.76	95.63	400

5.1 R-STDP as a Classification Criteria

We experimented with R-STDP learning rule applied to L5-L6 synapses of the network in the Figure 5-1 by two different kinds of weight initialization and also varying initialization of parameters like $\frac{N_{miss}}{N}$, $\frac{N_{hit}}{N}$ and N.

5.1.1 Backprop Initialized Weights for R-STDP

As given in Table 5.3 using an R-STDP as a classifier was not able to achieve an accuracy 97.2% obtained by a two layer FCN. In particular, perhaps the weight initialization plays a role in that the R-STDP rule can get stuck in a local minimum. To study this in more detail the network in Figure 5-1 was initialized with a set of weights that are known to give a high accuracy. To explain, the final weights used in the 2 Layer FCN reported in Table 5.2 were used as a starting point. As these weights are both positive and negative, they were shifted to be all positive. This was done by first finding the minimum value $w_{\min}$ (< 0) of these weights and simply adding $-w_{\min} > 0$ to them so that they are all positive. Then this new set of weights were re-scaled to be between 0 and 1 by dividing them all by their maximum value (positive). These shifted and scaled weights were then used to initialize the weights of the R-STDP classifier. The parameters $a_r^+, a_r^-, a_p^+, a_p^-$ were initialized to be 0.004, 0.003, 0.0005, 0.004 respectively. With the network in Figure 5-1 initialized by these weights, the validation images were fed through the network and the neuron number with the maximum potential is the predicted output. The validation accuracy was found to be 97.1%.

With weights of the fully connected layer of Figure 5-1 initialized as just de-

scribed, the R-STDP rule was used to train the network further for various number of epochs and two different ways of updating $\frac{N_{miss}}{N}$ and $\frac{N_{hit}}{N}$.

Batch Update of $\frac{N_{miss}}{N}$ and $\frac{N_{hit}}{N}$

The first set of experiments were done with the $\frac{N_{miss}}{N}$ and $\frac{N_{hit}}{N}$ ratios updated after every *batch* of N images for $N = 100, 500, 1500, 2500$. As the weights of the fully connected layer of Figure 5-1 with the backprop trained values, we expect $\frac{N_{miss}}{N}$ to be a low fraction or equivalently $\frac{N_{hit}}{N}$ to be high. Consequently, they were initialized as $\frac{N_{miss}}{N} = 0.1$, $\frac{N_{hit}}{N} = 0.9$. With these initialization, Table 5.4 shows that accuracy on the validation set did not decrease significantly for N not too large e.g., $N < 2500$). In general, using larger values of N (value of N depends on the initialization of N_{miss}/N and N_{hit}/N) the accuracy goes down significantly. For example, for the cases where $N_{miss}/N = 0.035$ and $N_{hit}/N = 0.965$ the accuracy didn't significantly decrease until the batch size was $N = 3500$. In the case with $N_{miss}/N = 0.0$ and $N_{hit}/N = 1.0$ the accuracy didn't decrease at all. This is because the best performing weights for validation accuracy were used, but these same weights also gave 100% accuracy on the training data.

Table 5.5 shows the classification accuracy with "poor" initialization $N_{miss}/N = 0.9$ and $N_{hit}/N = 0.1$. If the weights had been randomly initialized then the initialization $N_{miss}/N = 0.9$ and $N_{hit}/N = 0.1$ would be appropriate. However, Table 5.5 shows that R-STDP isn't able to recover from this poor initialization.

Update of $\frac{N_{miss}}{N}$ and $\frac{N_{hit}}{N}$ After Each Image

Next, N_{miss}/N and N_{hit}/N were updated after every image using the most recent N images. Even with N_{miss}/N and N_{hit}/N initialized incorrectly, the validation accuracies in Table 5.6 did not decrease significantly. Though the accuracy still goes down slightly, the table indicates that updating N_{miss}/N and N_{hit}/N after every image mitigates this problem.

Still updating N_{miss}/N and N_{hit}/N after each image, it was found that R-STDP

Table 5.4: Demonstration of sensitivity of R-STDP to N value with correct initialization of hit and miss ratios.

$\frac{N_{miss}}{N}$	$\frac{N_{hit}}{N}$	N	Acc. at start	Acc. at end
0.1	0.9	100	97.1%	96.91%
0.1	0.9	500	97.1%	96.96%
0.1	0.9	1500	97.1%	96.82%
0.1	0.9	2500	97.1%	90.76%
0.035	0.965	2500	97.1%	96.69%
0.035	0.965	3000	97.1%	96.58%
0.035	0.965	3500	97.1%	91.05%
0.035	0.965	4000	97.1%	90.98%
0.0	1.0	100	97.1%	96.93%
0.0	1.0	500	97.1%	96.93%
0.0	1.0	1500	97.1%	96.94%
0.0	1.0	2500	97.1%	96.94%
0.0	1.0	3000	97.1%	96.94%
0.0	1.0	3500	97.1%	96.94%
0.0	1.0	4000	97.1%	96.93%

Table 5.5: Demonstration of sensitivity of R-STDP to N value with incorrect initialization of hit and miss ratios.

$\frac{N_{miss}}{N}$	$\frac{N_{hit}}{N}$	N	Acc. at start	Acc. at end
0.9	0.1	100	97.1%	91.52%
0.9	0.1	500	97.1%	90.67%
0.9	0.1	1500	97.1%	90.47%
0.9	0.1	2500	97.1%	90.45%

accuracy was very sensitive to the initialized weights. Specifically the L5-L6 R-STDP weights were initialized using the backprop trained weights (as explained above) by doing the backprop for just 10 epochs (instead of 20) and $\lambda = 10.0$ (regularization parameter) which gave 99.6% training and 96.8% validation accuracies. Table 5.7 gives the validation accuracies using R-STDP for 100 epochs. Surprisingly, even with a good initialization of the weights and the ratios N_{miss}/N and N_{hit}/N, the validation accuracy suffers.

For the same cases as Table 5.7 the R-STDP algorithm was run for 1000 epochs with the training and validation accuracies versus epoch plotted in Figure 5-2. No-

Table 5.6: Demonstration of sensitivity of R-STDP.

$\frac{N_{miss}}{N}$	$\frac{N_{hit}}{N}$	N	Acc. at start	Acc. at end
0.9	0.1	100	97.1%	96.93%
0.9	0.1	500	97.1%	96.94%
0.9	0.1	1500	97.1%	96.93%
0.9	0.1	2500	97.1%	96.94%

Table 5.7: Demonstration of sensitivity of R-STDP for weight initialization.

$\frac{N_{miss}}{N}$	$\frac{N_{hit}}{N}$	N	Acc. at start	Acc. at end
0.0	1.0	100	96.8%	90.75%
0.0	1.0	4000	96.8%	90.67%

tice that the validation accuracy drops to ~90%. It seems that R-STDP is not a valid cost function as far as accuracy is concerned[1]. Interestingly, as shown next, training with R-STDP with randomly initialized weights, the validation accuracy only goes up to ~90% (see Figure 5-3).

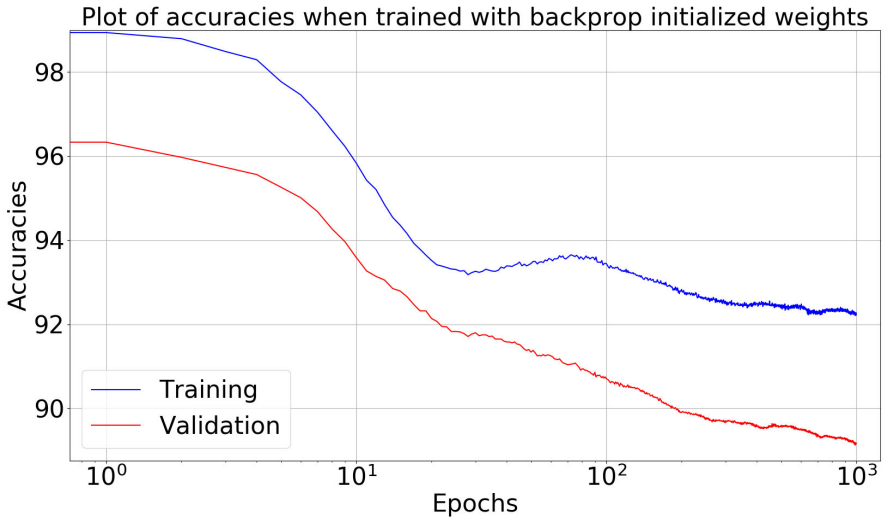

Figure 5-2: Plot of accuracies versus epochs when the weights were initialized with backprop trained weights.

[1] At least using one output neuron per class.

5.1.2 Randomly Initialized Weights for R-STDP

In the set of experiments with R-STDP the weights were *randomly* initialized from the normal distribution $\mathcal{N}(0.8, 0.01)$ and the $N_{miss}/N, N_{hit}/N, N$ parameters initialized with the values given in Table 5.8. Validation accuracies are shown at the end of 100 epochs N_{miss}/N and $N_{hit}/N, N$ were updated after every image.

Table 5.8: Demonstration of sensitivity of R-STDP.

$\frac{N_{miss}}{N}$	$\frac{N_{hit}}{N}$	N	Acc. at start	Acc. at end
0.9	0.1	100	10.3	90.22
0.9	0.1	500	10.1	90.13
0.9	0.1	1500	10.2	90.12
0.9	0.1	2500	10.6	90.16

For these same cases as Table 5.8, the R-STDP algorithm was run for 1000 epochs with the training and validation accuracies versus epochs plotted in Figure 5-3. The validation accuracy only goes up to ~90%.

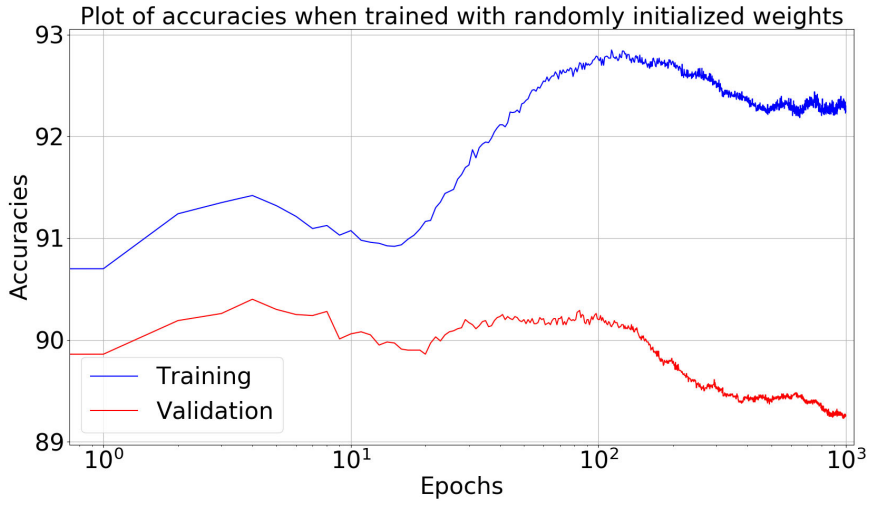

Figure 5-3: Plot of accuracies versus epochs when the weights were randomly initialized.

In this chapter we showed that a simple linear neural network (without a hidden layer) trained with error backpropagation performs better than R-STDP.

CHAPTER SIX: CLASSIFICATION OF THE N-MNIST DATA SET

6.1 Transfer Learning

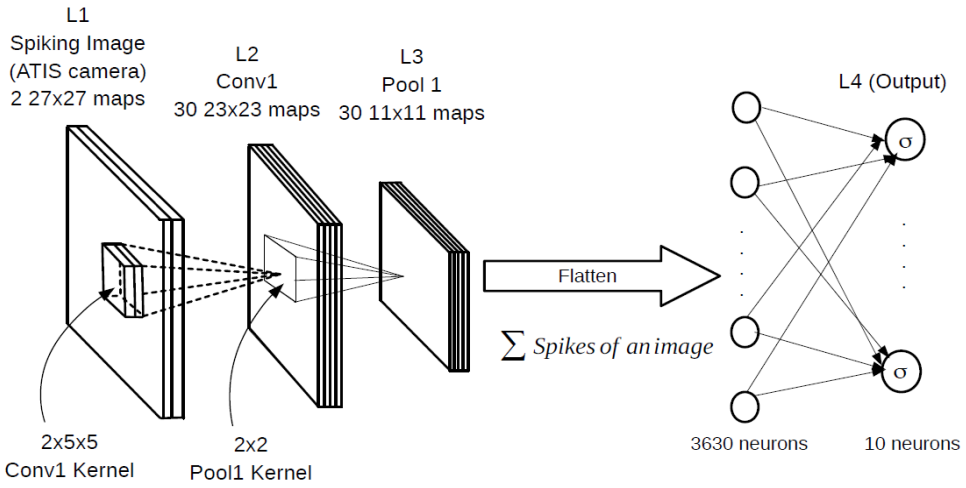

Figure 6-1: Network for N-MNIST classification.

In the above experiments, we artificially constructed spiking images using a DoG filter on the standard MNIST data set as in [35][34]. However the ATIS (silicon retina) camera [73] works by producing spikes. We also considered classification directly on recorded output from the ATIS camera given in the N-MNIST data set [67]. A silicon retina detects change in pixel intensity and thus the MNIST digits are recorded with camera moving slightly (saccades). Figure 6-2 shows the raw accumulated spikes of the N-MNIST data set as given in [67].

Figure 6-3 is the same as Figure 6-2, but corrected for saccades (camera motion) using the algorithm given in [67]. Figure 6-1 shows the network we used for

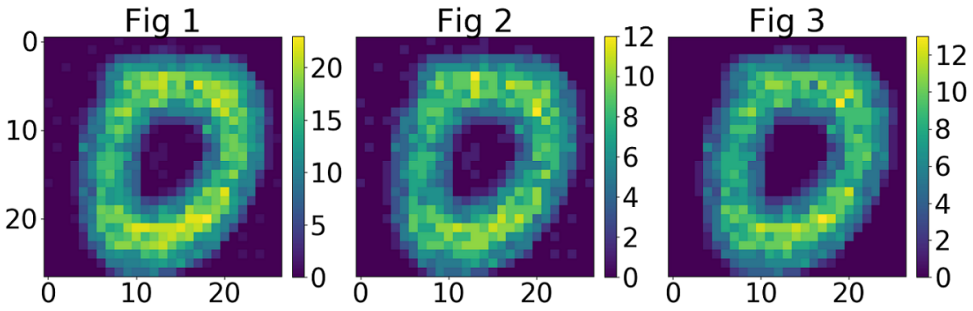

Figure 6-2: Left: Accumulated ON and OFF center spikes. Center: Accumulate ON center spikes. Right: Accumulated OFF center spikes.

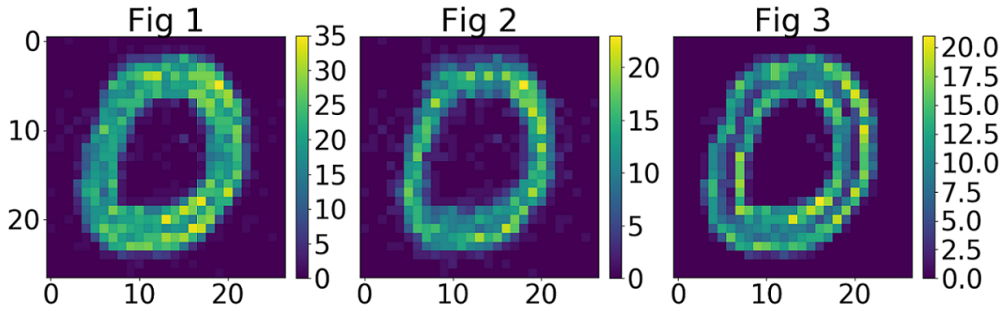

Figure 6-3: Left: Accumulated ON and OFF center spikes. Center: Accumulate ON center spikes. Right: Accumulated OFF center spikes.

classification of the N-MNIST data. We first hard wired the weights W_{C1} of the convolution kernel from L1 to L2 of Figure 6-1 to the values already trained above in subsection 4.2 (see Figure 4-3). Only the weights from L4 to L5 were trained for classification by simply back propagating the errors from L5 to L4. This result in given in the first row of Table 6.1. We also trained an SVM on the L4 neuron outputs with the results given in row 2 (RBF) and row 3 (linear) of Table 6.1. All the results in Table 6.1 were done on the raw spiking inputs from [67] (i.e., not corrected for saccade) with training done on 50,000 (spiking) images, validation & testing done on 10,000 images each.

Table 6.1: Classification accuracies of N-MNIST data set with one convolution/pool layers for transfer learning.

Classifier	Test Acc	Valid Acc	Training Time	λ	η	Epochs
2 Layer FCN	97.45%	97.62%	5 minutes	$\frac{1}{10.0}$	$\frac{0.1}{1.007^{\#Epoch}}$	20
RBF SVM	98.32%	98.40%	200 minutes	$\frac{1}{3.6}$	-	-
Linear SVM	97.64%	97.71%	100 minutes	$\frac{1}{0.012}$	-	-

6.2 Training with N-MNIST Spikes

In Table 6.2 we show the results for the case where the weights W_{C1} of the convolution kernel from L1 to L2 of Figure 6-1 were trained (unsupervised) using the N-MNIST data set. In this instance we used N-MNIST data corrected for saccades since this gave better result than the uncorrected data. All the results in Table 6.2 were produced by training on 50,000 (spiking) images with validation & testing done using 10,000 images.

Table 6.2: Classification accuracies of N-MNIST data set with one convolution/pool layers when trained with N-MNIST spikes.

Classifier	**Test Acc**	**Valid Acc**	**Training Time**	λ	η	**Epochs**
1 Layer FCN	97.21%	97.46%	5 minutes	$\frac{1}{10.0}$	$\frac{0.1}{1.007^{\#Epoch}}$	20
RBF SVM	98.16%	98.2%	150 minutes	$\frac{1}{3.6}$	-	-
Linear SVM	97.38%	97.44%	100 minutes	$\frac{1}{0.012}$	-	-

We also added an extra convolution layer, but found that the classification accuracy decreased. Jin et al [32] reported an accuracy of 98.84% by using a modification of error back propagation (all layers) algorithm. Stromatias et al. [86] reported an accuracy of 97.23% accuracy by using artificially generated features for the kernels of the first convolutional layer and training a 3 layer fully connected neural network classifier on spikes collected at the first pooling layer.

In this chapter we used the N-MNIST dataset to train the SNN. We also performed transfer learning on a network that was trained using synthetically generated spikes from the MNIST dataset.

CHAPTER SEVEN: FEATURE RECONSTRUCTION AND OVER TRAINING

7.1 Feature Reconstruction

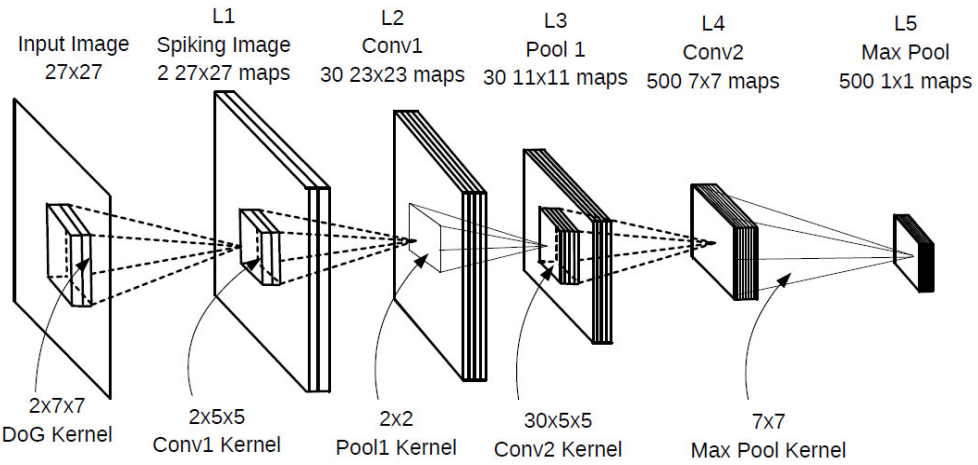

Figure 7-1: Network showing two convolution layers and a final global pooling layer.

We have already presented in Figure 3-8 a reconstruction of the convolution kernels (weights) from Layer L1 to Layer 2 into features. Each of the 30 maps of L2 has a convolution kernel in $\mathbb{R}^{2\times5\times5}$ associated with it which maps L1 to L2 using convolution. We now want to reconstruct (visualize) the features learned by the second convolution layer. Each of the 500 maps of L4 (see Figure 7-1) has a convolutional kernel associated with it which maps L3 to L4, i.e., for $w = 0, 1, 2, ..., 499$. These kernels have the form

$$W_{C2}(w,k,i,j) \in \mathbb{R}^{30\times5\times5} \text{ for } 0 \le k \le 29 \text{ and } (0,0) \le (i,j) \le (4,4).$$

So $W_{C2} \in \mathbb{R}^{500\times30\times5\times5}$, a 5×5 area of pooled layer L3 receives spikes from 10×10 area of neurons in L2. For $w = 0, 1, 2, ..., 499$, the kernels $W_{C2}(w, k, i, j) \in \mathbb{R}^{30\times5\times5}$ are reconstructed to be features

$$F_{P1}(w, k, i, j) \in \mathbb{R}^{30\times10\times10} \text{ for } 0 \leq k \leq 29 \text{ and } (0, 0) \leq (i, j) \leq (9, 9)$$

connecting L2 to L4, so $F_{P1} \in \mathbb{R}^{500\times30\times10\times10}$. How is this done? F_{P1} is initialized with all zeros. Consider the 1^{st} kernel $W_{C2}(0, k, i, j) \in \mathbb{R}^{30\times5\times5}$ and for the k^{th} 5×5 slice of $W_{C2}(0, k, i, j) \in \mathbb{R}^{5\times5}$ the value of the (i, j) element is mapped to the $(2i, 2j)$ element of the k^{th} 10×10 slice of $F_{P1}(0, k, i, j) \in \mathbb{R}^{10\times10}$. All other values of the k^{th} 10×10 slice in F_{P1} are set to zero. This is repeated for all $k = 0, 1, ..., 29$ and for $w = 0, 1, ..., 499$. Now recall that there are 30 kernels in W_{C1}. Specifically, for $z = 0, 1, 2, ..., 29$.

$$W_{C1}(z, k, i, j) \in \mathbb{R}^{2\times5\times5} \text{ for } 0 \leq k \leq 1 \text{ and } (0, 0) \leq (i, j) \leq (4, 4).$$

$k = 0$ is for ON center kernels and $k = 1$ is for OFF center kernels so $W_{C1} \in \mathbb{R}^{30\times2\times5\times5}$. Note that 27×27 neurons in L1 map to 23×23 (27-5+1×27-5+1) neurons in L2 when using a valid mode convolution, conversely a 10×10 area of neurons in L2 receive spikes from a 14×14 area of neurons in L1.So W_{C1} kernels map spikes from 14×14 area of neurons in L1 to a 10×10 area of layer of L2. Thus the feature $F_{P1}(0, k, i, j) \in \mathbb{R}^{30\times10\times10}$ must be reconstructed to be a feature in $\mathbb{R}^{2\times14\times14}$ that corresponds to the input layer L1. That is, for $w = 0, 1, ..., 499$

$$F_{L1}(w, k, i, j) \in \mathbb{R}^{2\times14\times14} \text{ for } 0 \leq k \leq 1 \text{ and } (0, 0) \leq (i, j) \leq (14, 14).$$

So $F_{L1} \in \mathbb{R}^{500\times2\times14\times14}$ (Each neuron in L4 has a field of view of $2 \times 14 \times 14$ neurons in L1). How is this done?

Let the 5×5 matrix on the left-hand side of Figure 7-2 denote an ON center kernel $W_{C1}(z, 0, i, j) \in \mathbb{R}^{5\times5}$ for some $z = 0, 1, ..., 29$. In particular, let it be the second kernel so $z = 1$, $W_{C1}(1, 0, i, j) \in \mathbb{R}^{5\times5}$. Now the 1^{st} feature de-

noted by $F_{P1}(0,k,i,j) \in \mathbb{R}^{30\times10\times10}$ can be visualized as being made up of 10×10 slices for $k = 0, 1, ..., 29$. To go with the second kernel $W_{C1}(1,0,i,j) \in \mathbb{R}^{5\times5}$ we take the second slice ($k = 1$) of the feature $F_{P1}(0,k,i,j) \in \mathbb{R}^{30\times10\times10}$ denoted as $F_{P1}(0,1,i,j) \in \mathbb{R}^{10\times10}$ which we take to be the 10×10 matrix on the right-hand side of Figure 7-2. In practice these slices are sparse and we show the particular slice in Figure 7-2 to have only two non zero elements, the $(1,1)$ and the $(5,5)$ elements. To carry out the reconstruction at L1 we compute $w_{11}^{(1)} \times W_{C1}(1,0,i,j) \in$

5×5

0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	0

10×10

$w_{11}^{(1)}$	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	$w_{55}^{(1)}$	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figure 7-2: Left: Second ON 5×5 kernel (out of 30 kernels), $W_{C1}(1,0,i,j) \in \mathbb{R}^{5\times5}$. Right: Second 10×10 slice (out of 30 slices) of 1^{st} feature (out of 500 features) of pool 1 features, $F_{P1}(0,1,i,j) \in \mathbb{R}^{10\times10}$.

$\mathbb{R}^{5\times5}$ and center it on $w_{11}^{(1)}$ of $F_{P1}(0,1,i,j) \in \mathbb{R}^{10\times10}$ as indicated in Figure 7-3. We then repeat this process for all non zero elements of $F_{P1}(0,1,i,j) \in \mathbb{R}^{10\times10}$ which in this example is just $w_{55}^{(1)}$. Filling in with zeros we end up with the 14×14 matrix shown in Figure 7-4. Similarly, to reconstruct the third 14×14 matrix we use the third kernel $W_{C1}(2,0,i,j) \in \mathbb{R}^{5\times5}$ ($z = 2$) taken to be the 5×5 matrix on the left-side of Figure 7-5 and the third slice ($k = 2$) of the feature $F_{P1}(0,k,i,j) \in \mathbb{R}^{30\times10\times10}$ denoted as $F_{P1}(0,2,i,j) \in \mathbb{R}^{10\times10}$ which we take to be the 10×10 matrix on the right-hand side of Figure 7-5. Here the only non zero components are $w_{11}^{(2)}$ and $w_{51}^{(2)}$. We compute $w_{11}^{(2)} \times W_{C1}(2,0,i,j) \in \mathbb{R}^{5\times5}$ and center it on $w_{11}^{(1)}$ of $F_{P1}(0,2,i,j) \in \mathbb{R}^{10\times10}$ as indicated in Figure 7-6. We then compute $w_{51}^{(2)} \times W_{C1}(2,0,i,j) \in \mathbb{R}^{5\times5}$ and center it on $w_{51}^{(2)}$ of $F_{P1}(0,2,i,j) \in \mathbb{R}^{10\times10}$. In non zero overlapping elements of the 14×14 matrix the components are just added

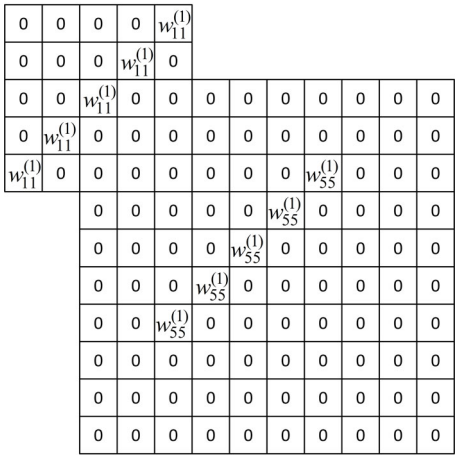

Figure 7-3: Reconstruction at Conv1 (L2). Figure shows 1^{st} feature of 500 feature maps and 2^{nd} slice of 30 slices, $F_{L1}(0, 1, i, j) \in \mathbb{R}^{14\times14}$.

together as shown in Figures 7-6 and 7-7. Finally, 30 of these 14×14 matrices shown in Figures 7-4 and 7-7 are added up ($\sum_{k=0}^{29} F_{L1}(0, k, i, j) \in \mathbb{R}^{14\times14}$) to reconstruct the 1^{st} ON center feature of the 500 features learned by neurons of L4, this procedure is repeated for the OFF center features as well. In other words, a particular neuron of L4 spikes when it detects its corresponding ($2 \times 14 \times 14$) ON and OFF center feature in the original image. Figure 7-8 shows 150 of the 500 reconstructed features from the 500 convolution kernels of the second convolution from L3 to L4. Each feature is 14×14 neurons (pixels) of the original spiking image with ON (green) and OFF (red) features superimposed on top of each other.

14×14

0	0	0	0	$w_{11}^{(1)}$	0	0	0	0	0	0	0	0	0
0	0	0	$w_{11}^{(1)}$	0	0	0	0	0	0	0	0	0	0
0	0	$w_{11}^{(1)}$	0	0	0	0	0	0	0	0	0	0	0
0	$w_{11}^{(1)}$	0	0	0	0	0	0	0	0	0	0	0	0
$w_{11}^{(1)}$	0	0	0	0	0	0	0	$w_{55}^{(1)}$	0	0	0	0	0
0	0	0	0	0	0	0	$w_{55}^{(1)}$	0	0	0	0	0	0
0	0	0	0	0	0	$w_{55}^{(1)}$	0	0	0	0	0	0	0
0	0	0	0	0	$w_{55}^{(1)}$	0	0	0	0	0	0	0	0
0	0	0	0	$w_{55}^{(1)}$	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-4: Reconstruction at Conv1 (L2), $F_{L1}(0, 1, i, j) \in \mathbb{R}^{14\times14}$.

5×5

0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0

10×10

$w_{11}^{(2)}$	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
$w_{51}^{(2)}$	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figure 7-5: Left: Third ON 5×5 kernel (out of 30 kernels), $W_{C1}(2, 0, i, j) \in \mathbb{R}^{5\times5}$. Right: Third 10×10 slice (out of 30 slices) of 1^{st} feature (out of 500 features) of pool 1 features, $F_{P1}(0, 2, i, j) \in \mathbb{R}^{10\times10}$.

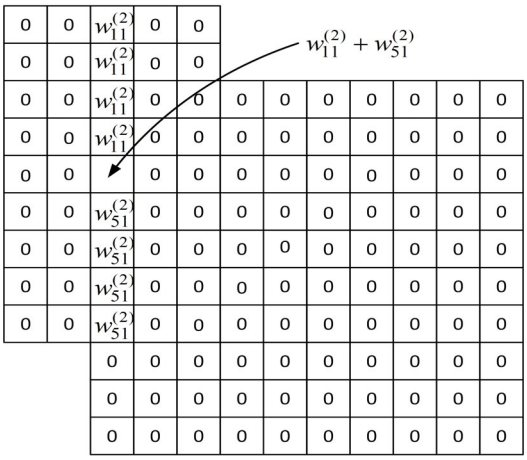

Figure 7-6: Reconstruction at Conv1 (L2), $F_{L1}(0, 2, i, j) \in \mathbb{R}^{14\times14}$.

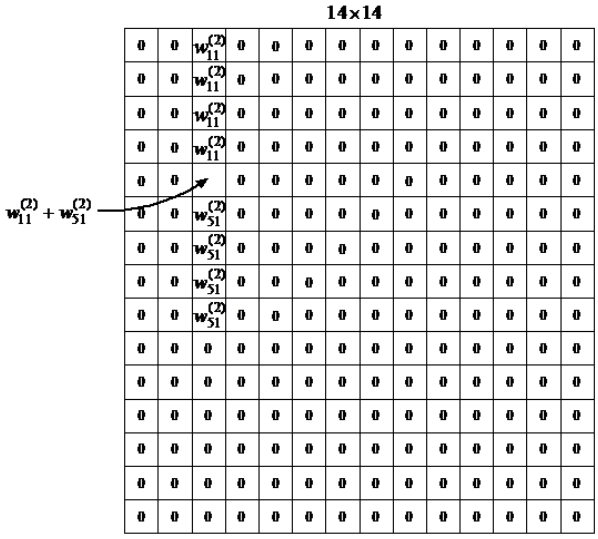

Figure 7-7: Reconstruction at Conv1 (L2), $F_{L1}(0, 2, i, j) \in \mathbb{R}^{14\times14}$.

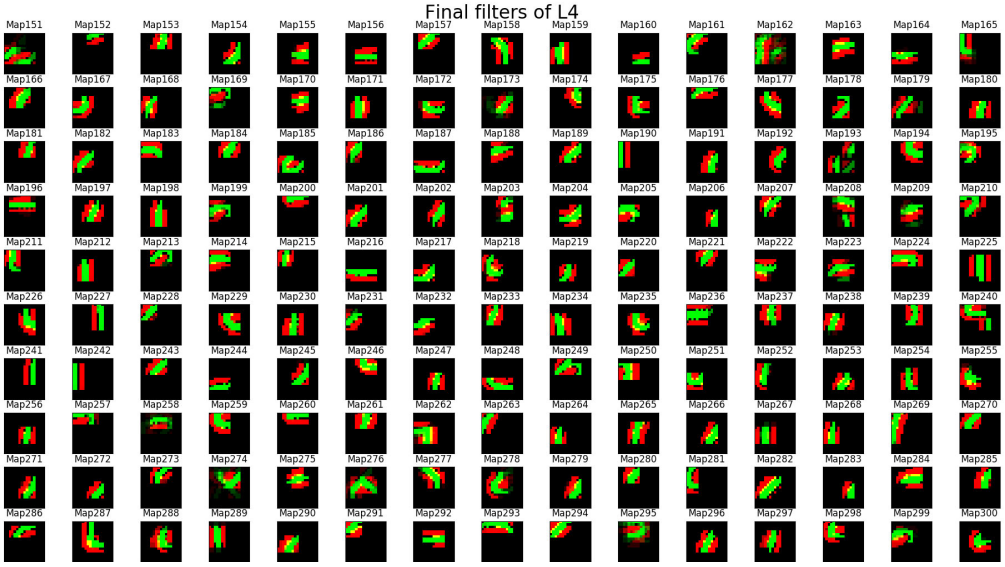

Figure 7-8: Weights of 150-300 maps of L4 that is trained by in coming spikes without lateral inhibition in L3, STDP competition region in L4 set to $\mathbb{R}^{500\times3\times3}$ and with homeostasis signal applied in L4, notice that the reconstructed features are quite complex and they could well represent a digit or a major section of a digit, note that all neurons of a map in a layer will have shared weights. In this experiment number of maps is L4 was set to 500. Notice that the reconstructed features are not as complex looking as in Figure A-1

7.2 Effect of Over Training the Convolution Kernels

The first row of Figure 7-9 shows the reconstruction of the features from the convolution kernels of the L3 to L4 layer after training with just 20,000 images. In contrast, the second row of Figure 7-9 shows the reconstruction of the features from the convolution kernels of the L3 to L4 layer after training with 60,000 MNIST images for 4 epochs. This shows that more training results in individual kernel weights (w_{ij}) saturating to 1 or 0 (i.e., the reconstructions in the second row are sharper), but the features become less complex. Figure 7-9 shows that we need a

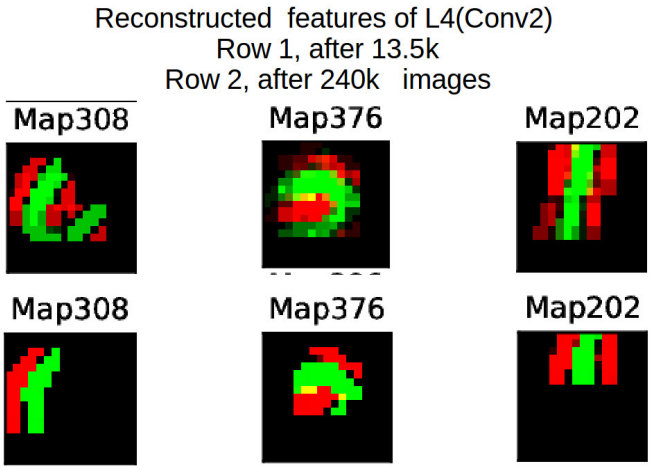

Figure 7-9: Reduction in the complexity of learned features because of over training. First row of this figure shows reconstruction of L3→L4 synapses after training for 15.5k images and second row shows the reconstruction of L3→L4 synapses after training for 240k images (4 epochs)

mechanism to stop training. To this end, we looked at the difference in weights during training. Consider

$$W_{C2}^{(n)} = \{w^{(n)}(z,i,j,k)\} \in \mathbb{R}^{500\times30\times5\times5}$$

where $W_{C2}^{(n)}$ is kernel W_{C2} after the n^{th} training is image has passed. The L3L4 (red) plot of Figure 7-10 is a plot of

$$\frac{\sum_{z=0}^{499}\sum_{i=0}^{29}\sum_{j=0}^{4}\sum_{k=0}^{4}\left(w^{(n*150)}(z,i,j,k)-w^{((n+1)*150)}(z,i,j,k)\right)}{375000} \text{ for } n=0,1,...,130$$

where $375000 = 500 \times 30 \times 5 \times 5$. Similarly the L1L2 (blue) plot was done for $W_{C1}^{(n)} = \{w^{(n)}(z,i,j,k)\} \in \mathbb{R}^{30\times2\times5\times5}$.

For the L3L4 the weights dramatically change between $n = 80$ and $n = 100$. Multiple experiments indicated that over training of W_{C2} kernels starts after $n = 100$. If the network was trained further, we found that the final classification accuracy drops by by $\sim$2%. Kheradpisheh et al. [35] proposed a convergence factor

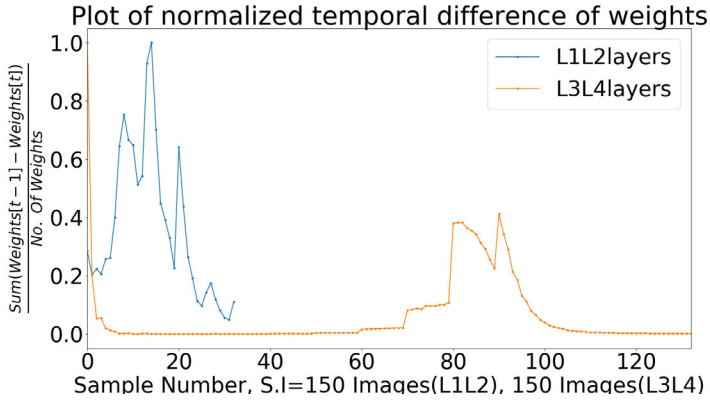

Figure 7-10: Plot shows the difference of successive samples of synapses. If the difference approaches zero it means that weights are not changing hence features learnt by a neuron also remain the same. Notice the sudden jump in difference between 80-100 samples.

given by

$$\frac{\sum_{z=0}^{499}\sum_{i=0}^{29}\sum_{j=0}^{4}\sum_{k=0}^{4}\left(w^{(n*150)}(z,i,j,k)(1-w^{(n*150)}(z,i,j,k))\right)}{375000} \text{ for } n=0,1,...,130.$$

The convergence plot is shown in Figure 7-11. The training was stopped when the convergence factor is between 0.01 and 0.02. We found that using this criteria there was a bit of over training resulting in 1%-2% decrease in testing accuracy.

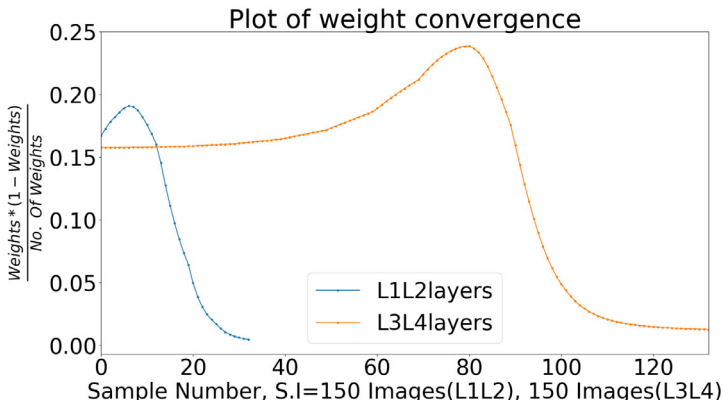

Figure 7-11: Plot shows the fashion of convergence for the synapses. Note that the convergence factor dips sharply between the samples 80-100.

In this chapter, we discussed the over training problem that arises when using unsupervised STDP. We also showed that over training results in reduction in complexity of the features learned in deeper layers. We also proposed a heuristic method to prevent over training.

CHAPTER EIGHT: SURROGATE GRADIENTS AND STDP

In this chapter we shall discuss how to combine STDP based unsupervised feature classification with stochastic gradient descent for the classification layers of an SNN. We planned to use R-STDP as a classification criterion for the extracted binary spike features, but we decided against it owing to its slow convergence (see Chapter 5.1). Stochastic gradient descent (SGD) via backpropagation is the primary choice for state-of-the-art classification, regression, and generative learning [95]. A cost function is assigned to the last layer of the network and the synapses are updated to minimize the cost. In our network, backpropagation is used only in the classification layers (L3-L4-L5) of the network which has a single hidden layer L4. Let $\delta^l, a^l = \sigma(z^l), b^l, W^l, z^l = w^l z^{l-1} + b^l$ denote the error vector, the activation vector, the bias vector, the weights and the net input to the activation function for the l^{th} layer, respectively [66]. With σ the activation function and$C$ denotes the output cost. For convenience we shall re-state the backpropagation equations from Chapter 2.

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{8.1}$$

where δ^L denotes the error vector on the last layer and the error vector for the hidden layers are given recursively by

$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{8.2}$$

Updates to biases and weights of layer l are calculated using

$$\frac{\partial C}{\partial b^l} = \delta^l \tag{8.3}$$

$$\frac{\partial C}{\partial W^l} = \delta^l a^{(l-1)T} \tag{8.4}$$

C denotes the cost in the final layer. We used a softmax activation with a cross entropy cost function for the last layer so that equation (8.1) becomes

$$\delta^L = -(y - a^L), \tag{8.5}$$

where a^L and y are softmax activation of the output layer and the one hot label vector, respectively. For the remainder of the chapter we refer to gradients obtained using Equations (8.1)-(8.5) as *true gradients* with $\sigma(z)$ an ReLU activation function.

8.1 Binary Activations and Surrogate Gradients

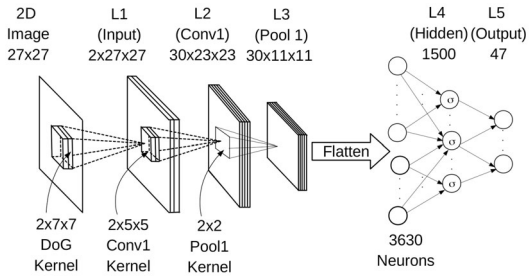

Figure 8-1: Layers $L1 - L3$ are the feature extraction layers and layer $L3 - L5$ are the feature classification layers.

8.1.1 Weight Initialization

The weights of the $L2$ layer are initialized from the normal distribution $\mathcal{N}(0.8, 0.04)$. The weights of layers $L4$ & $L5$ layers are initialized from the normal distribution $\mathcal{N}(0, 0.01)$, but truncated to restrict them between ± 0.02. A softmax activation is used for the classification layer $L5$ with its inputs converted to integers using the floor function. A look-up table containing predefined values of the exponential

function e^x can be used to calculate softmax activation in a hardware implementation. The activation functions employed in layer $L4$ (denoted by σ in Figure 8-1) are discussed below (in Section 8.1).

In order to significantly reduce the number of high precision multiplications the activation functions of the L4 layer are made binary. That is, if the net input to a neuron is greater than zero the output is one. Otherwise the output is zero. Consequentially this activation function is not differentiable (the gradient doesn't exist). Here we give two different possible functions that we used to replace the true gradient, i.e., to be its surrogate [11].

8.1.2 Surrogate Gradient 1

The activation function of a neuron in layer $L4$ is defined by

$$a^l = \sigma(z^l) \triangleq \begin{cases} 0, & z < 0 \\ z, & 0 \leq z < \tau \leq 1 \\ \tau, & z \geq \tau. \end{cases} \tag{8.6}$$

Figure 8-2 is a plot of this activation function which is a ReLU that saturates for some $0 < \tau < 1$.

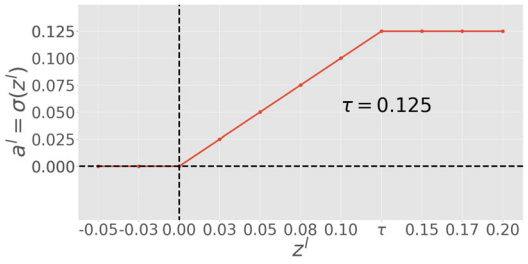

Figure 8-2: Activation function $a^l = \sigma(z^l)$ for neurons in layer $L4$.

The activation is required to be binary so its definition is modified to be ($\lceil \cdot \rceil$

denotes the ceiling function)

$$a^l = \lceil \sigma(z^l) \rceil \triangleq \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \tag{8.7}$$

For this activation (8.7) we define its surrogate gradient to be

$$\sigma'(z^l) \triangleq \begin{cases} 1, & 0 \leq z < \tau \leq 1 \\ 0, & \text{otherwise.} \end{cases} \tag{8.8}$$

which is the derivative of Equation 8.6 and is shown in Figure 8-3.

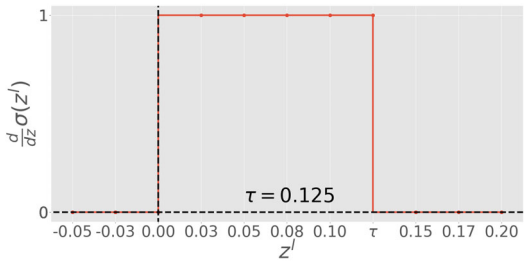

Figure 8-3: Surrogate gradient of activation function defined in equation (8.6).

Simulations were performed by setting τ to $0.25, 0.125, 0.05$ and it was found that 0.125 maximized the validation accuracy. As equation (8.7) is not differentiable the derivative of $\sigma(z)$ is taken to be equation (8.8). For convenience, we denote an activation value of 1 as spike and an activation value of 0 as no spike.

8.1.3 Surrogate Gradient 2

We also considered a second activation given by

$$a^l = \sigma(z^l) \triangleq \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \tag{8.9}$$

and define its surrogate gradient to be

$$\sigma'(z^l) \triangleq \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \tag{8.10}$$

Note that $\sigma'(z) = \sigma(z)$ and is binary so that $\sigma'(z^l) = a^l$ in the hidden layer. Equation (8.2) then becomes

$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot a^l \tag{8.11}$$

where a^l determines if a neuron spikes in the l^{th} layer. That is, a^l determines if a neuron in the l^{th} layer is to receive error information from the $l+1$ layer. Substituting Equation (8.11) in Equation (8.4) gives

$$\frac{\partial C}{\partial W^l} = \left((w^{l+1})^T \delta^{l+1} \odot a^l\right) a^{(l-1)T} \tag{8.12}$$

This shows that a neuron in $l-1$ layer gets to update its synapse with a neuron in l^{th} layer if both neurons have spiked, i.e., for $\partial C / \partial W^l_{pq}$ to be a non-zero both a^l_p and a^{l-1}_q have to be non-zero.

8.2 MNIST

The MNIST digits were passed through the network in Figure 8-1 and encoded into spike vectors as described in Chapter 3.2.2. Note that the extracted features are *binary* valued. Table 8.1 shows that surrogate gradient 1 yields a test accuracy 0.74% higher or 74 more correct classifications compared to surrogate gradient 2 with $10,000$ test images. Figure 8-4 shows the classification accuracy per class using the surrogate gradient 1. For results reported in Table 8.1 a dropout (50%) mechanism was used in the hidden layer for regularization, the number of neurons in layer L4 were set to 900, and mini-batch size was set to 5 and η for the actual and true gradients was set to 0.0125 and 0.01, respectively. These results were obtained

by averaging over five experiments with the classification layers of the network (in Figure 8-1) trained for 30 epochs each time. For accuracies reported using the true gradient a quadratic cost function with a ReLU activation function for layers L4, L5 was used whereas for accuracies reported using the surrogate gradients a cross-entropy cost function with softmax approximation (see Section 8.1.1) for layer L5 and binary activation function for layers L3, L4 was used.

Table 8.1: MNIST results. True gradients refers to Equations (8.1)-(8.5).

Gradient Type	Mean Test Acc.	Max. Test Acc.
True Gradient	98.58%	98.66%
Surrogate Gradient 1	98.49%	98.54%
Surrogate Gradient 2	97.75%	97.77%

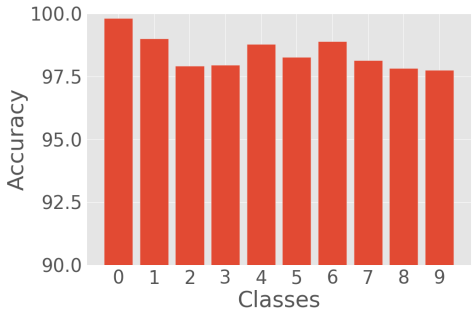

Figure 8-4: Classification accuracy per class with surrogate gradient 1.

8.3 Extended MNIST

The EMNIST dataset has 47 classes containing handwritten upper & lower case letters of the English alphabet in addition to the digits. This dataset is divided into 102,648 training images, 10,151 validation images, and 18,800 test images [9]. The mini-batch size was set to 5 and a dropout of 50% was used in the hidden layer (L4). The number of neurons in layer L4 was 1500. The number of epochs was set to 35 and all the experiments were averaged over 5 trials.

8.3.1 Why Use Unsupervised STDP Based Feature Extraction?

In this section *binary* valued features vectors (i.e., vector with 0s and 1s) were collected in layer $L3$ as described in Chapter 3.2.2. Classification was performed using an ANN with binary activation for the hidden layer $L4$ neurons and an approximated softmax output explained in Section 8.1.1. The synapses of L2 layer (Conv1) were fixed with random weights and the binary spike features collected in layer L3 were classified using surrogate gradient 1 resulting in 80.43% maximum test accuracy. Similarly, binary spike features collected from layer L3 with unsupervised trained weights in layer L2 (Conv2) were classified using surrogate gradient 1 and resulted in a maximum test accuracy of 85.6% or ≈ 972 more correct classifications when compared to random weights in L2. Results averaged over five trials are given in Table 8.2. Figures 8-6 and 8-5 show the confusion matrices for the network with random synapses in L2 and STDP trained synapses in L2. When the layer L2 was trained with STDP, Figure 8-5 shows that there is frequent misclassification between the classes {f} and {F}, the classes {0} and {O}, the classes {q} and {9}, the classes {1}, {I} and {L}, the classes{S} and {5}, and the classes {2} and {Z}. Misclassifications for this case are explainable in the sense that one might expect humans to make such errors. For example, in 6^{th} element of the 3^{rd} row off Figure 8-7 the network predicted a lower case "f", while the label was an upper case "F". In contrast, when layer L2 was not trained, Figure 8-6 shows that the network frequently misclassified the classes {H} and {0}, the classes {E} and {1}, the classes {A} and {1}, the classes {Z} and {7}, and the classes {h} and {L}. One would not expect humans to make such mistakes.

Table 8.2: EMNIST accuracy with random and trained L2 layer.

Gradient Type	Mean Test Acc.	Max Test Acc.	L2 Synapses
Surrogate Gradient 1	80.21%	80.43%	Random
Surrogate Gradient 1	85.35%	85.60%	STDP trained

We performed experiments to study the classification accuracy in the presence of noise in the spiking input images (L1). To explain, suppose a particular image

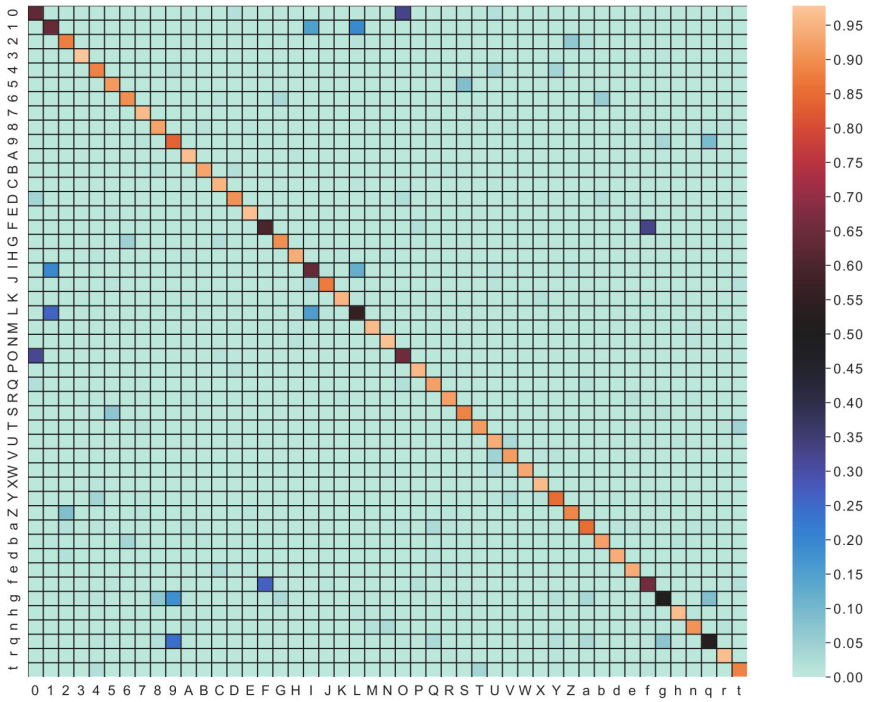

Figure 8-5: Confusion matrix of predictions with EMNIST dataset when the synapses in layer L2 were learned in an unsupervised fashion using STDP.

resulted in 100 spikes in L1. Then by 10% noise we mean that 5 of the randomly chosen neurons that spiked were set to zero, while 5 randomly chosen non-spiking neurons were forced to spike. Figure 8-8 shows the result of this input noise on the final classification accuracy. As shown in Figure 8-8, the network can withstand $\approx 40\%$ this input noise before the classification accuracy decreases to that of the case where the L2 layer synapses were set randomly.

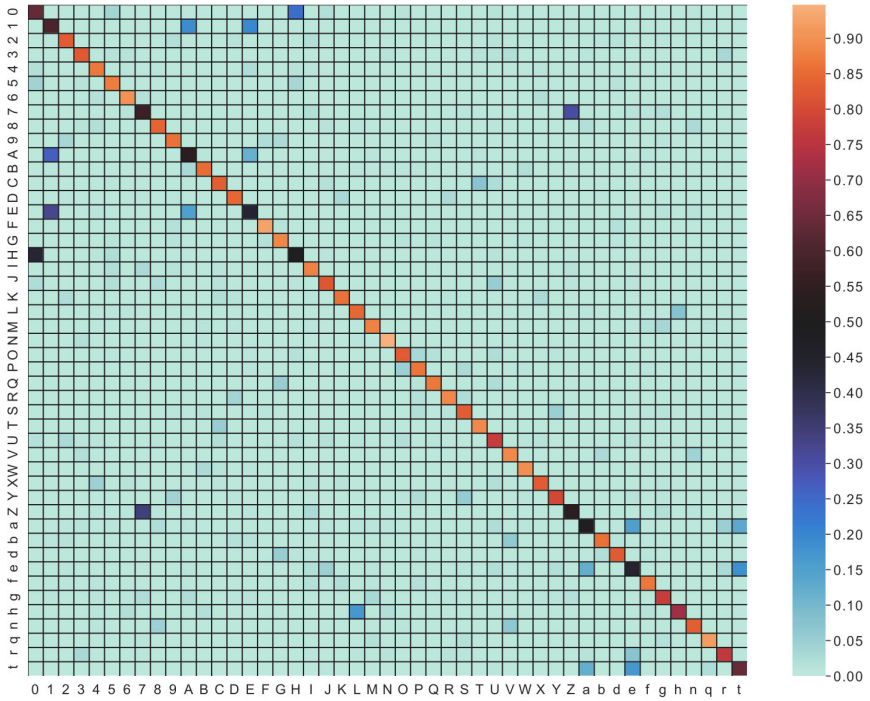

Figure 8-6: Confusion matrix of predictions with EMNIST dataset when the weights (synapses) in layer L2 were random.

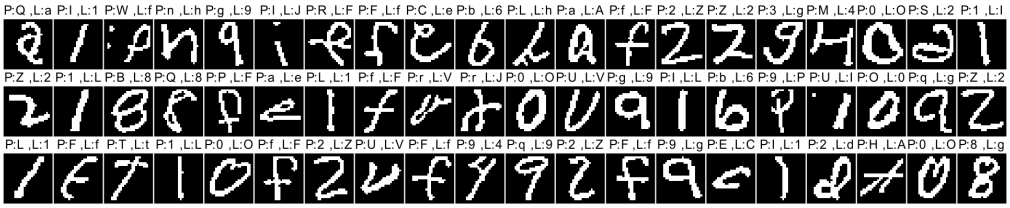

Figure 8-7: Frequently misclassified classes in the EMNIST dataset. *P* and *L* denote predicted class and actual label, respectively.

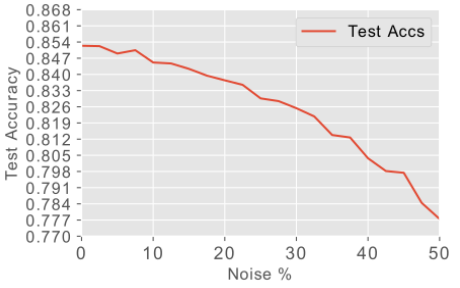

Figure 8-8: Effect of input noise on the final classification accuracy.

8.3.2 Effect of Gradient Approximation on Classification

Table 8.3: EMNIST results. True gradient refers to Equations (8.1)-(8.5).

Gradient Type	Mean Test Acc.	Max. Test Acc.	Cond. Max. Test Acc.	η	Activation
True Gradient	85.47%	85.7 %	94.49 %	0.05	ReLU
Surrogate Gradient 1	85.35 %	85.60 %	94.1 %	0.02	Binary
Surrogate Gradient 2	84.24 %	84.47 %	93.72 %	0.02	Binary

Table 8.3 shows that the true gradients results in best classification accuracy and surrogate gradient 1 outperforms gradient surrogate 2 by 1.0% (188 more correct classifications with 18800 test images).

8.3.3 Conditioning on Upper Case, Lower Case, and Digits

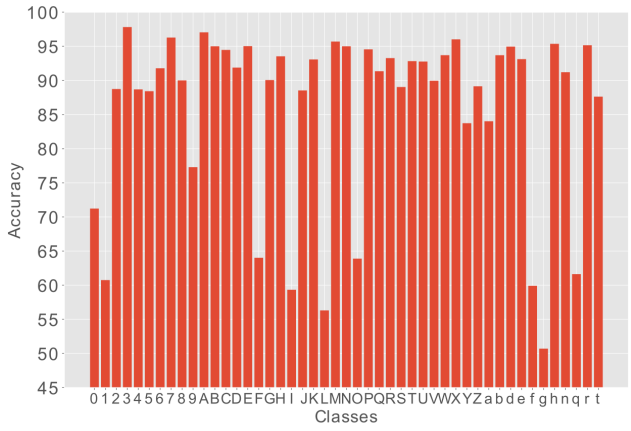

Figure 8-9: Classification accuracy per class with surrogate gradient 1.

Figure 8-9 shows the accuracy per class when surrogate gradient 1 is used for classification. With handwritten data even a human classifier may not be able to tell the difference between, for example, the upper case letter "O" and the digit "0". To study this we also ran the classifier conditioned on (given that) the image under test was an either an upper case letter, a lower case letter, or a digit. No retraining was done for this section. Table 8.3 shows the dramatic increase in accuracy under this conditioning. The accuracy per class using this conditioning is given in Figure 8-10. It is seen that the classes I, L, g, q have the least recognition rate, but still well above their accuracies given previously in Figure 8-9 where conditioning was not used. In

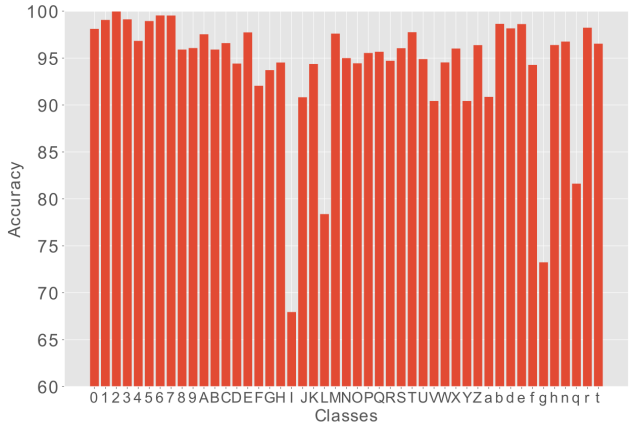

Figure 8-10: Classification accuracy per class of EMNIST dataset with surrogate gradient 1 after conditioning.

more detail we found that about 13% of the letters “q” were misclassified as the letter "g", about 4% of letters “q” were misclassified as the letter “a”, while about 83% of letters "q" were correctly classified. About 20% of letters “g” were misclassified as the letter “q” while about 73% of letters “g” were correctly classified. Similarly, we found that about 27% of letters of upper case “I” (eye) were misclassified as the upper case letter “L" while 68% of upper case “I” were correctly classified. As a final observation about 20% of upper case letters “L” were misclassified as an upper case “I” (eye) while about 78% of upper case letters “L” were correctly classified. Figure 8-11 shows the confusion matrix for the conditioned case.

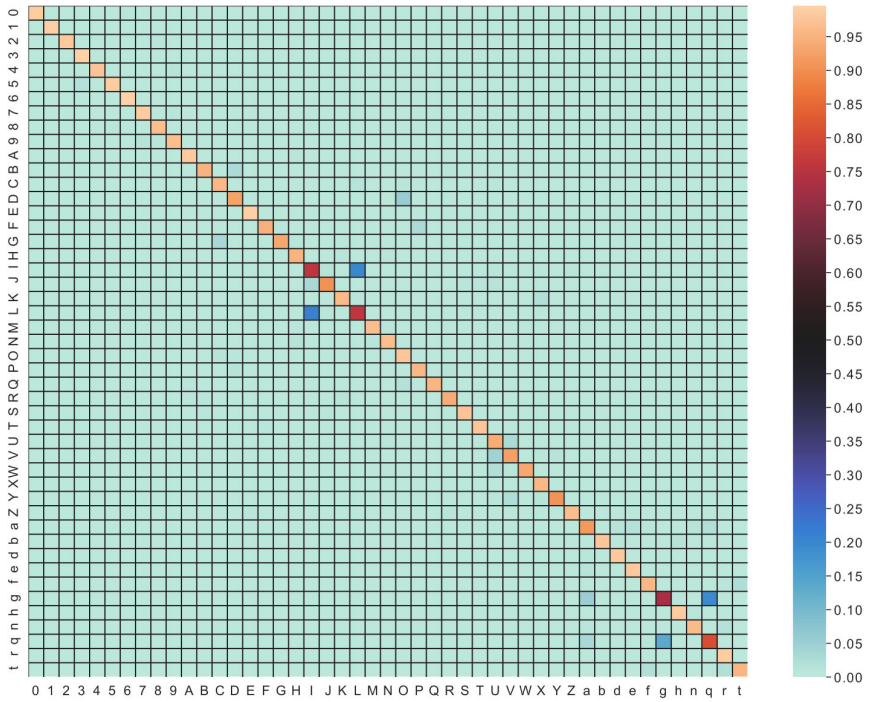

Figure 8-11: Confusion matrix of predictions with EMNIST dataset when the inputs are conditioned on Upper Case, Lower Case and Digits.

8.3.4 Computational Advantage of Binary Activations

In the feedforward paths L1 through L4 the matrix-vector multiplication operations can all be avoided in a hardware implementation as these layers all have binary activations. For example, executing the multiplication of a set of (floating point) weights times a set of spikes (binary activations) is simply.

$$\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ & \vdots & \\ w_{n1} & w_{n2} & w_{n3} \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} w_{12} \\ w_{22} \\ w_{32} \\ \vdots \\ w_{n2} \end{pmatrix} + \begin{pmatrix} w_{13} \\ w_{23} \\ w_{33} \\ \vdots \\ w_{n3} \end{pmatrix}. \tag{8.13}$$

That is, multiplication is replaced by addition. This technique avoids the need for dedicated multiplier hardware and allows the feasibility of in memory com-

puting [97][98]. Another advantage is found in backpropagation computations. Specifically, as the surrogate gradient $\sigma'(z^l)$ is binary, the error vector δ^l for the hidden layer can be obtained without having to do a majority of the row-column multiplications for example,

$$\left(\underbrace{\begin{pmatrix} \mathrm{w}_{11} & \mathrm{w}_{12} & \mathrm{w}_{13} \\ \mathrm{w}_{21} & \mathrm{w}_{22} & \mathrm{w}_{23} \end{pmatrix}}_{(w^{l+1})^T} \times \underbrace{\begin{pmatrix} 1 \\ 2.5 \\ 3.1 \end{pmatrix}}_{\delta^{l+1}} \right) \odot \underbrace{\begin{pmatrix} 0 \\ 1 \end{pmatrix}}_{\sigma'(z^l)} = \begin{pmatrix} 0 \\ \mathrm{w}_{21} + 2.5\mathrm{w}_{22} + 3.1\mathrm{w}_{23} \end{pmatrix}. \tag{8.14}$$

That is, in equation (8.14) the row-column multiplications of the first row are avoided as the result will zero due to the element-wise (Hadamard product) vector multiplication. All the weight updates, $\partial C/\partial W^l$ can be obtained without explicitly calculating the vector outer product $\delta^l a^{(l-1)T}$ as the activations of $L3$ and $L4$ layers are binarized. For example,

$$\underbrace{\begin{pmatrix} \mathrm{a} \\ \mathrm{b} \\ \mathrm{c} \end{pmatrix}}_{\delta^l} \times \underbrace{\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}}_{a^{(l-1)T}} = \begin{pmatrix} 0 & a & 0 \\ 0 & b & 0 \\ 0 & c & 0 \end{pmatrix}. \tag{8.15}$$

That is, the matrix on the right side of Equation (8.15) is found by simply transcribing δ^l into its columns as specified by $a^{(l-1)T}$.

8.3.5 Number of High-Precision Multiplications

Table 8.4: Comparison of multiplications for a DNN and an SNN in Figure 8-1.

Architecture	L2	L4	L5	Total
DNN	2.84×10^{12}	3.92×10^{13}	5.06×10^{11}	$\approx4.25\times10^{13}$
Proposed SNN	5.22×10^{7}	1.87×10^{10}	0	$\approx1.87\times10^{10}$

The majority of computations in a DNN are high-precision multiplications of the weights with the activations during both the forward inference as well as the

backpropagation of the error. Energy consumption of the network is hardware architecture dependent, but in order to provide an estimate about the energy savings in our SNN we compare the number of high precision multiplications between a DNN and our SNN [77]. It requires $m \times n \times p$ high precision multiplications in order to multiply an $m \times n$ matrix by an $n \times p$ matrix in a fully connected network. Convolution (in valid mode) of an $I \times I$ image with an $F \times F$ filter requires $(I-F+1)\times(I-F+1)\times F\times F$ multiplications. As we employ temporally encoded spikes with binary activations used in the classification layer, the forward path can be implemented with no multiplications (See Equation (8.13) and Equation (8.15)). Further, orders of magnitude less multiplications are required for backpropagation as we explain next. Table 8.4 below compares the number of high precision multiplications required for a DNN with our approach. In a neuromorphic system the input spikes are typically provided by a silicon retina (eDVS [10]) so we assume that the images are available in spike form. We begin by estimating the number of multiplications in the L4 layer for our SNN. Figure 8-12 shows the average number of neurons in the L4 layer (1500 total neurons) for each epoch that have a non-zero activation. The number of multiplications required to calculate the error in layer L4 according to Equation (8.14) is as follows: In the earliest epochs, the number of multiplications during the training is approximately 1.45×10^9 computed from

$$20500 \text{ m-batches} \times 5 \frac{\text{images}}{\text{m-batch}} \times 47 \text{ classes} \times 300 \text{ non-zero activations.}$$

In the latter epochs the number of non-zero activations decreases to 100 making the number of multiplications approximately $20500 \times 5 \times 47 \times 100 = 4.5 \times 10^8$. Summing over the 35 epochs results in approximately 1.87×10^{10} multiplications. To compute the number of multiplications in the L2 layer of our SNN, note that during the unsupervised training of L2 (Conv1), the lateral inhibition and STDP competition result in sparse neuronal activity in that there are only 5.8 weight updates (winner spikes) per spiking input image (see Section 3.2.1). L2 was trained (unsupervised) on 6000 spiking input images. The number of multiplications required is

approximately 5.22×10^7 computed from

$$5.8 \text{ avg updates} \times 2 \times 5 \times 5 \times 30 \text{ L2 synapses} \times 6000 \text{ images}.$$

Due to the binary activation of L4, layer L5 of our network can be implemented in a custom hardware without any multiplications. Based on this quantitative analysis our approach makes a suitable candidate for low power implementations as it uses approximately $3-4$ orders of magnitude less multiplications compared to a standard DNN.

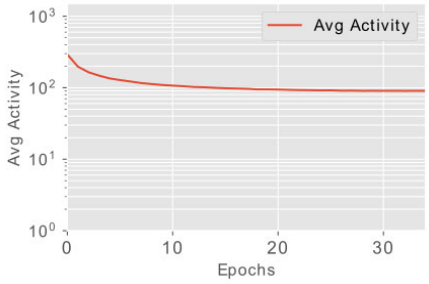

Figure 8-12: Number of neurons with non-zero activations in layer L4 as the training in classification sections of the network in Figure 8-1 progresses.

8.4 SPYKEFLOW

The PYNN software tool with NEURON [93] [26] was considered as a simulation tool. However these tools are designed for neuroscientists with neuron models much more complex than needed in our case. The software tool NENGO [5], developed for bio-inspired machine learning, uses a more complex neuronal model than required here. Motivated by the simple spiking models in Kheradpisheh et al.'s work [35], we developed a software tool called SPYKEFLOW. SPYKEFLOW[1] primarily uses NUMPY to do the calculations of lateral inhibition, STDP updates, neuron spike accumulation, etc. However, SPYKEFLOW also uses TENSORFLOW for computationally intensive calculations such as convolution and pooling. Therefore, the users will have the ability to use a GPU, if one is available. Detailed

[1] `https://github.com/ruthvik92/SpykeFlow`

instructions to use the software are provided in [94]. Following [35] our package supports instantaneous (non leaky integrate and fire) neurons, latency encoding, and inhibition mechanisms to be able to simply extract meaningful features from the input images. The feature extraction in our SNNs is done unsupervised using STDP, which requires monitoring the weight updates (synapse changes) in the spiking network. The SPYKEFLOW software provides the capability to monitor spike activity, weight evolution (updates), feature extraction (spikes per map per label), and synapse convergence [91] [92]. Similar to SPYKEFLOW, Mozafari et al. released the software tool SPYKETORCH in [62], which is based on the PYTORCH deep learning tool.

8.5 Comparison with Other Works

A comparison of our work with recent publications that employ the EMNIST dataset is provided in Table 8.5. Rate encoded spiking networks require hundreds of time-steps of simulation for a single input image resulting in very high spike counts. In contrast, latency encoded inputs to an SNN equipped with first spike based feature extraction results in very few spikes, in turn this requires fewer synapse updates implying lower power consumption.

Table 8.5: Comparison of EMNIST classification results.

Learning method	Neuron model	Input Encoding	Max. Test Acc.
Supervised DNN [81]	ReLU	-	90.59 %
Supervised SNN[32]	LIF	Rate	85.57 %
This work	Instantaneous	Latency	85.60 %

In this dissertation, neurons are essentially used as coincidence detectors with latency encoded input spikes and first spike based feature extraction to transform the inputs to spike feature vectors that contain robust object category information as observed in biology [56]. These spike features were then classified using the proposed backpropagation with surrogate gradients to demonstrate up to 85.60% accuracy with the EMNIST dataset. This was achieved by employing backpropagation only in the classification layers of the network which are decoupled from

the feature extraction layers. The accuracy achieved here is quite comparable to the 85.57% accuracy reported in [32] which used rate encoded (Poisson) input spikes in a network with one hidden layer comprised of 800 neurons and with backpropagation performed in all the layers. Furthermore, [32] uses complex leaky integrate-and-fire (LIF) neurons as opposed to our simple instantaneous summation neurons that act as coincidence detectors. Using a conventional deep convolution network, Shawon et al. [81] report an accuracy of 90.59% on the balanced EMNIST (see the survey paper [4]). The deep network in [81] consisted of 6 convolution layers, a hidden layer with 64 neurons, followed by a classification layer. Though our accuracy is lower than DNNs, we have proposed an energy-efficient solution using bio-inspired unsupervised techniques. This energy efficiency can be realized by implementing the proposed architecture using a Neuromorphic ASIC or FPGA. We also demonstrated an accuracy of 94.49% when the classifier was given the information that an input image was either a letter (upper or lower case) or a digit. As discussed in the above sections, this conditioning was considered due to the indistinguishability of some samples between a few of the classes e.g., betwen {0} and {O} in Figure 8-7.

In this chapter we introduced binary activations to reduce the number of floating point multiplications. We also showed that the STDP trained network can be resistent to the presence of stray spikes.

CHAPTER NINE: CATASTROPHIC FORGETTING

Catastrophic forgetting is a problematic issue in (non spiking) deep convolutional neural networks. In the context of the MNIST data set this refers to training the network to learn the digits 0,1,2,3,4 and, after this is done, training on the digits 5,6,7,8,9 is carried out. The catastrophic part refers to the problem that the network is no longer able to classify the first set of digits 0,1,2,3,4.

9.1 Catastrophic Forgetting in Non-Spiking Networks

In this section we shall examine a conventional (non-spiking) convolutional neural network whose weights were trained using backpropagation algorithm. In more detail, Figure 9-1 shows a conventional neural network with one convolution layer & one pool layer followed by a fully connected softmax output.

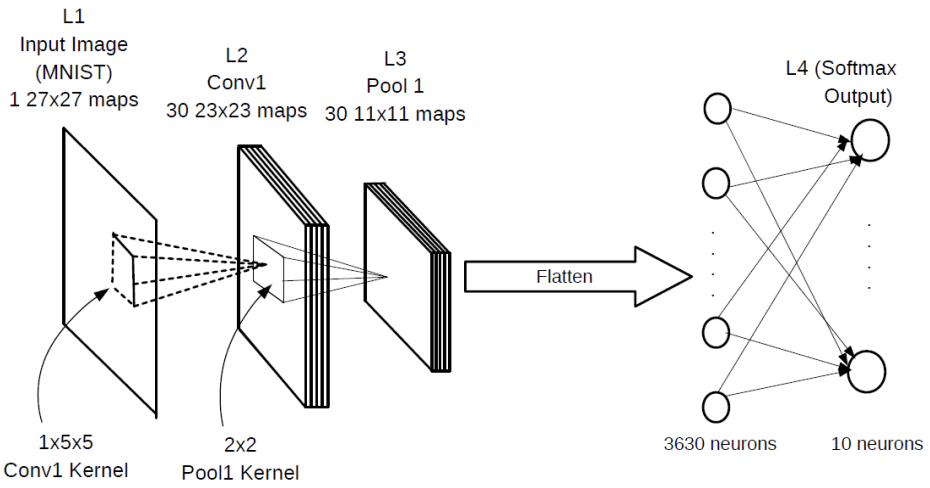

Figure 9-1: Network architecture for catastrophic forgetting.

This network has 10 outputs, but was first trained only on the digits 0,1,2,3,4

back propagating the error (computed from all 10 outputs) to the input (convolution) layer. This training used approximately 2000 digits per class and was done for 75 epochs. Before training the network on the digits 5,6,7,8,9 we initialized the weights and biases of the convolution and fully connected layer with the saved weights of the previous training. For the training with the digits 5,6,7,8,9 we *fixed* the weights and biases of the convolution layer with their initial values. The network was then trained, but only the weights of the fully connected layer were updated. (I.e., the error was only back propagated from the 10 output neurons to the previous layer (flattened pooled neurons). This training also used approximately 2000 digits per class and was done for 75 epochs. While the network was being trained on the second set of digits, we computed the validation accuracy on all 10 digits at the end of each epoch. These accuracies are plotted in Figure 9-2. The solid red line in Figure 9-2 are the accuracies versus epoch on the first set of digits {0,1,2,3,4} while the solid blue line gives the accuracies on the second set of digits {5,6,7,8,9} versus epochs. Figure 9-3 is a zoomed in picture of Figure 9-2 for better resolutions of the accuracies above 90%. These plots also show the validation accuracy results when the second set of training data is modified to include a fraction of the data from the first set of training digits {0,1,2,3,4}. For example, the dashed red line is the validation accuracy on the first set of digits when the network was trained with 2000 digits per class from {5,6,7,8,9} *along with* 200 (10%) digits per class from{0,1,2,3,4}. The blue dashed line is the validation accuracy of the second set of digits after each epoch. Similarly this was done with 15%, 25%, 27.5%, and 30% of the first set of digits included in the training set of the second set of digits. The solid red line shows that after training with the second set of digits for a single epoch the validation accuracy on first set goes down to 10% (random accuracy). The solid blue line shows a validation accuracy of over 97% on the second set of digits after the first epoch. Thus the network has now learned the second set of digits, but has catastrophically forgotten the first set of digits shown by solid red line.

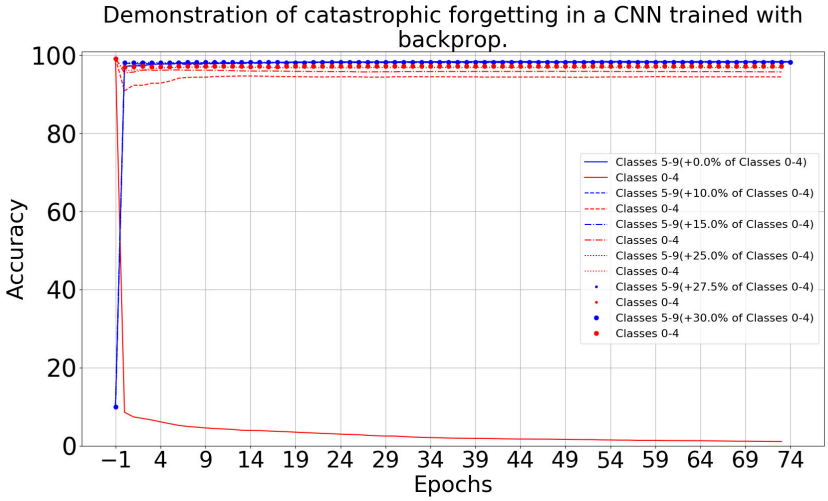

Figure 9-2: Catastrophic forgetting in a convolutional network while revising a fraction of the previously trained classes. Note that epoch -1 indicates that the network was tested for validation accuracy before training of the classes 5-9 started. Brackets in the legend shows the fraction of previously trained classes that were used to revise the weights from the previous classes.

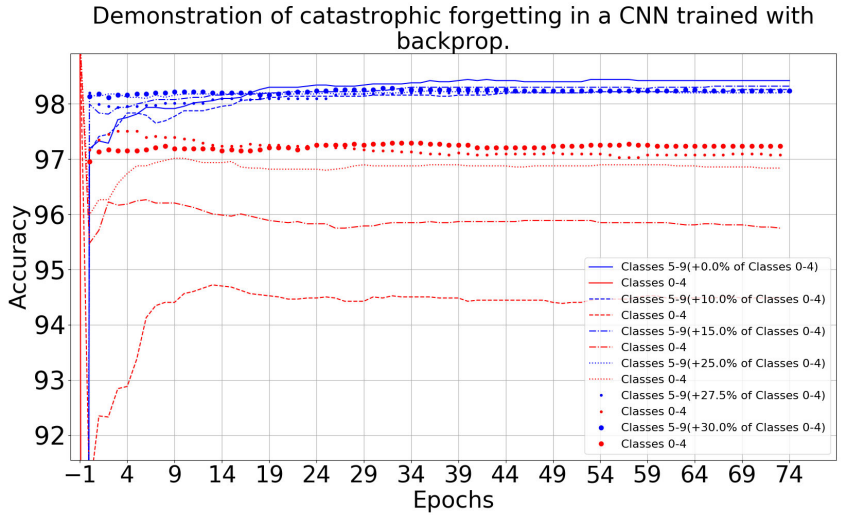

Figure 9-3: Zoomed upper portion of the Figure 9-2

9.2 Forgetting in Spiking Networks

For comparison we tested forgetting in our spiking network of Section 4.2 (see Figure 4-3). The network was first trained only on the digits {0,1,2,3,4} with unsupervised STDP on the convolution layer and back propagating the error (computed from all 10 outputs) just to the previous (flattened pool) layer. This training used approximately 2000 digits per class and was done for 75 epochs. Then, before training the network on the set of digits {5,6,7,8,9}, we initialized the weights of the convolution and fully connected layer with the saved weights of the previous training. For the training with the digits {5,6,7,8,9} we *fixed* the weights of the convolution layer with their initial values. The network was then trained, but only the weights of the fully connected layer were updated. I.e., the error was only back propagated from the 10 output neurons to the previous flattened layer. This training also used approximately 2000 digits per class and was done for 75 epochs. While the network was being trained on the second set of digits, we computed the validation accuracy on all 10 digits at the end of each epochs. These accuracies are shown in Figure 9-4. The solid red line in Figure 9-4 are the accuracies versus epochs on the first set of digits {0,1,2,3,4} while the solid blue line gives the accuracies on the second set of digits {5,6,7,8,9} versus epochs. Figure 9-5 is a zoomed in picture of Figure 9-4 for better resolutions of the accuracies above 90%. These plots also show the validation accuracy results when the second set of training data modified to include a fraction of data from the first set of training digits {0,1,2,3,4}. For example, the dashed red line is the validation accuracy on the first set of digits when the network was trained with 2000 digits per class of {5,6,7,8,9} *along with* 200 (10%) digits per class of {0,1,2,3,4}. The blue dashed line is the validation accuracy of the second set of digits after each epoch. Similarly this was done with 15%, 25%, 27.5%, and 30% of the first set of digits included in the training set of the second set of digits. The solid red line shows that after training with the second set of digits for a single epoch the validation accuracy on first set goes down to 77% (compared to the 10% accuracy of a non-spiking CNN). The solid blue line shows a validation accuracy

of about 95% on the second set of digits after the first epoch. Thus the network has now learned the second set of digits but has not catastrophically forgotten the first set of digits shown by solid red line.

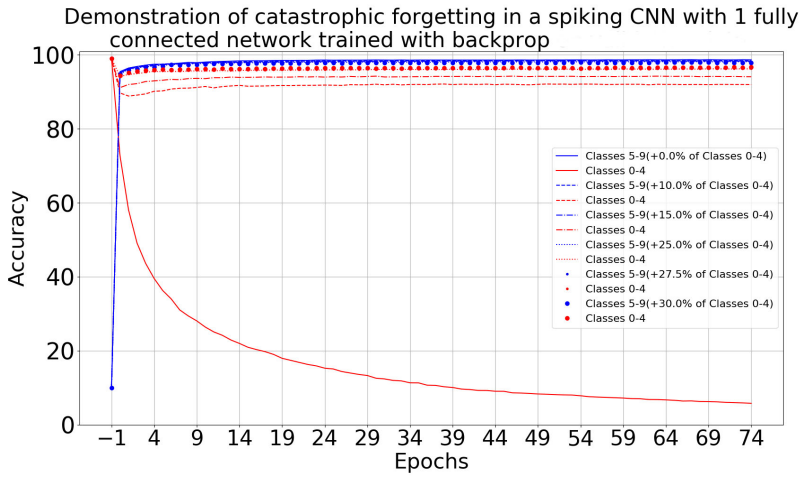

Figure 9-4: Catastrophic forgetting in a spiking convolutional network while revising a fraction of the previously trained classes. Note that epoch -1 indicates that the network was tested for validation accuracy before training of the classes 5-9 started. Brackets in the legend shows the fraction of previously trained classes that were used to revise the weights from the previous classes.

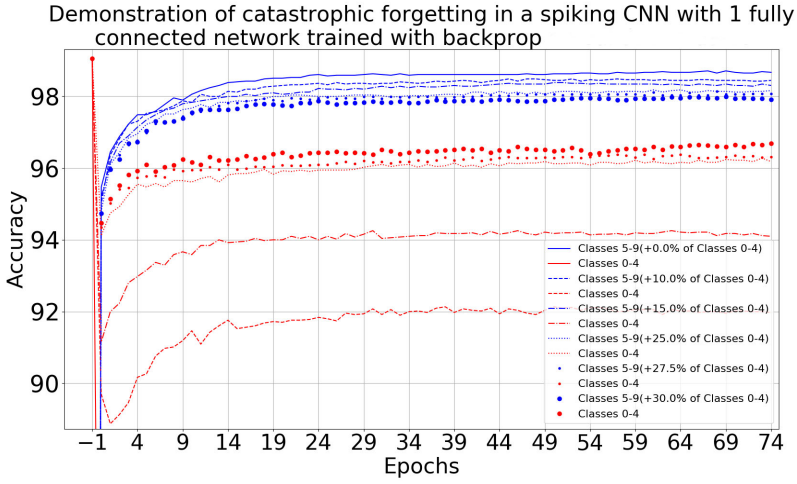

Figure 9-5: Zoomed upper portion of the Figure 9-4

As another approach we first trained on the set {0,1,2,3,4} exactly as just described above. However, we then took a different approach to training on the set {5,6,7,8,9}. Specifically we trained on 500 random digits chosen from {5,6,7,8,9}

(approximately 50 from each class) and then computed the validation accuracy on all ten digits. We repeated this for every additional 250 images with the results shown in Figure 9-6. Interestingly this shows that if we stop after training on 1000 digits from {5,6,7,8,9} we retain a validation accuracy of 91.1% and 90.71% test accuracy on all 10 digits.

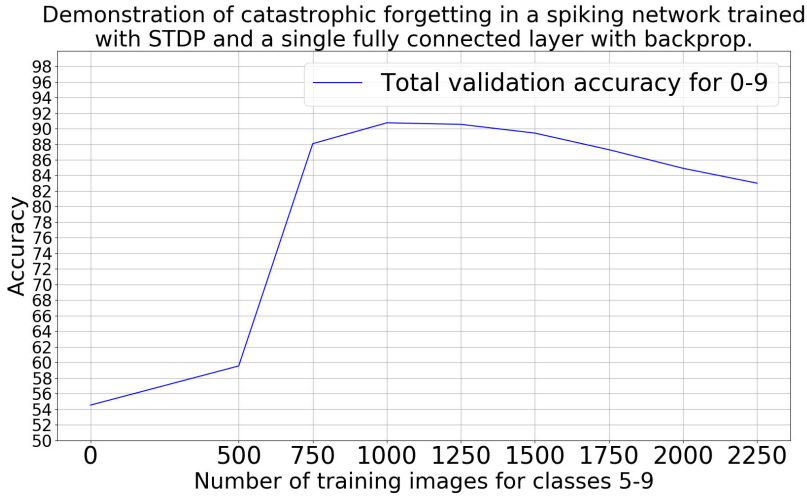

Figure 9-6: Note that as the number of training images for the classes 5-9 increases the total accuracy drops.

Table 9.1: Demonstration of forgetting in a spiking convolution network.

# images (classes 5-9)	# images (classes 0-4)	Validation	Test	Epochs
10,000	1000(10%)	95.235%	95.1%	75
10,000	1500(15%)	95.95%	95.9%	75
10,000	2500(25%)	96.83%	96.81%	75
10,000	2750(27.5%)	96.98%	96.92%	75
10,000	3000(30%)	97.1%	97.043%	75

Jason et al. reported an accuracy of 93.88% for completely disjoint data sets[2].

9.3 Continuous Learning in a Single-Incremental-Task Scenario with Spike Features

Typically, Spiking Neural Networks (SNNs) are trained using an unsupervised algorithm called Spike Timing Dependent Plasticity (STDP) [35]. Spike features

extracted from latency encoded convolutional variants of SNNs have been used with an SVM [35] and a linear neural network classifier [92] to achieve classification accuracies in excess of 98.5%. However, SNNs tend to achieve lower classification accuracies when compared to Artificial Neural Networks (ANNs) [69]. ANNs are trained using Stochastic Gradient Descent (SGD). The main assumption of SGD is that the mini-batches of the training data contain approximately equal number of data points with the same labels (i.e., the data is uniformly randomly distributed). This assumption does not hold for many of the machine learning systems that learn online continuously. Different kinds of continuous learning schemes have been proposed to mitigate the problem of catastrophic forgetting. Two main scenarios of continuous learning are the Multi-Task (MT) and the Single-Incremental-Task (SIT) scenarios [54]. In the MT scenario a neural network with a disjoint set of output neurons is used to train/test a corresponding set of disjoint tasks. In contrast, a neural network for the SIT scenario expands the number of neurons in the output layer to accommodate new classification tasks. The MT scenario is useful when training different classification tasks on the same network thereby allowing resource sharing. The SIT scenario is useful for online continuous learning applications. That is, the SIT scenario is more suitable for online machine learning systems and is more difficult compared to the MT scenario. This is because the SIT network has to not only mitigate catastrophic forgetting, but also learn to differentiate classes that are usually not seen together (unless the system has some kind of short term memory to be replayed later). Self-Organizing Maps (SOM) with short-term memory were used in [18] [70] to achieve an accuracy of 85% on the MNIST dataset using a SIT scenario and replaying the complete dataset. Using STDP based unsupervised learning and plasticity modulation, controlled forgetting was proposed in [2]. It was shown to achieve a 95% accuracy on MNIST dataset using the SIT scenario. Unsupervised spiking networks with predictive coding have been trained with STDP and shown to achieve an accuracy of 76% on the MNIST dataset using the MT scenario [68]. In our work here, our network classifies the data according to the AR1 method given in [54]. This uses the SIT scenario which

was inspired by synaptic intelligence for the MT scenario in [100]. In our previous work [92] we used the MNIST dataset split into two disjoint tasks to show that features extracted from a spiking convolutional network (SCN) demonstrated more immunity to catastrophic forgetting compared to their ANN counterparts. In [92], using early stopping, the first five output neurons were trained to classify the digits $\{0, 1, 2, 3, 4\}$ and then the remaining five output neurons were trained to classify the digits $\{5, 6, 7, 8, 9\}$. The network was then tested on the complete test dataset (digits 0-9) and achieved a 93% accuracy on this test data. In the work presented here we exclusively work with spike features extracted from an SCN and study the effect of continuous learning using the SIT scenario on the MNIST dataset. For this study the MNIST dataset was split into the five disjoint classification tasks $\{\{0, 1\}, \{2, 3\}, \{4, 5\}, \{6, 7\}, \{8, 9\}\}$. The feature classification is done unsupervised in the convolution layer (L2) while the classification is done in the latter layers using error backpropagation. Here we modify the synaptic intelligence regularizer calculation of [100] in order to reduce the computational load.

9.4 Network

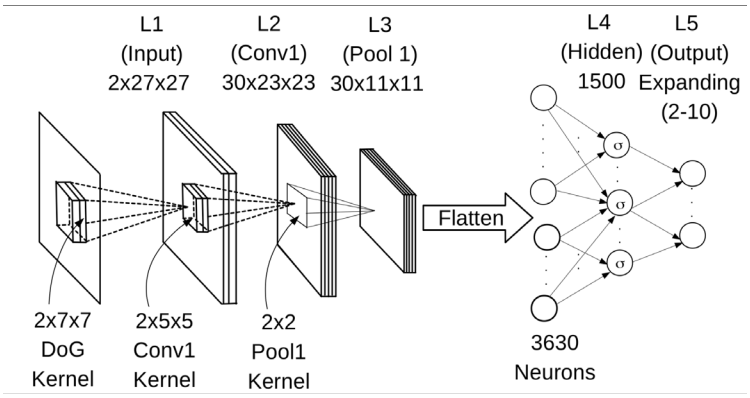

Figure 9-7: Layers L1-L3 and L3-L5 are feature extraction and feature classification layers respectively. Shown in the figure is an expanding output layer from 2-10 output neurons to accommodate the five classification tasks for the MNIST dataset. For the EMNIST dataset the same network has been modified to accommodate the ten classification tasks. EMNIST dataset with 47 classes has been divided to 10 sub tasks.

The feature extraction part of the network is same as in [91] [92]. Input images are encoded into spikes using ON and OFF center DoG filters followed by thresholding [35]. The L2 (convolution) layer consists of 30 maps and the neurons that emerge as winners after lateral inhibition and STDP competition [94] get to update their weights according to a simplified STDP [35] which was introduced in Chapter 1.1. For convenience the simplified STDP formula is given below

$$\Delta w_i = \begin{cases} -a^- w_i(1-w_i), & \text{if } t_{out} - t_{in} < 0 \\ +a^+ w_i(1-w_i), & \text{if } t_{out} - t_{in} \geq 0 \end{cases}$$

$$w_i \leftarrow w_i + \Delta w_i$$

t_{in} and t_{out} are the spike times of the pre-synaptic (input) and the post-synaptic (output) neuron, respectively. If the i^{th} input neuron spikes before the output neu-

ron spikes, the weight w_i is increased; otherwise the weight is decreased.[1] Learning refers to the change Δw_i in the (synaptic) weights with a^+ and a^- denoting the learning rate constants. These rate constants are initialized with low values $(0.004, 0.003)$ and are typically increased for every 1500 input images as learning progresses [35]. This STDP rule is considered simplified because the amount of weight change doesn't depend on the time duration between pre-synaptic and post-synaptic spikes. In this work, backpropagation is used only in the classification layers (L3-L4-L5) of the network with a single hidden layer L4.

9.5 Continuous Learning

By continuous learning we mean that the network in Figure 9-7 will start with two output neurons in L5 and be trained to classify the digits $\{0, 1\}$. During this training the error is backpropagated from layer L5 only as far as L3. After this training is complete two new neurons will be appended to the L5 layer and then trained to classify the digits $\{2, 3\}$. This is continued in the same manner for the three remaining classes $\{\{4, 5\}, \{6, 7\}, \{8, 9\}\}$. We proceed in the rest of this section to give the details of this training by specifying the cost function along with the (cost per synapse) weight regularizer. The neural network in this work has a softmax output layer which is the likelihood of the input image belonging to a particular class. Let $\mathbf{X} \in R^{3630}$ denote the (flattened) spike features in L3 and $\theta \in R^{1500\times 3630}$ denote the weights from L3 to the hidden layer L4. For task 1 there are two output neurons and we let C_1 denote the cross-entropy cost computed with the softmax outputs of these two neurons. For task 2 there are now four output neurons and we let C_2 denote the cross-entropy cost computed with the softmax outputs of these four neurons. The costs C_3, C_4, C_5 are defined in a similar manner. The L4 and the L5 weights are updated using SGD on mini-batches. $C_1^{(m)}$ denotes the cost of the m^{th} input mini-batch for the task $\{0, 1\}$. $C_2^{(m)}, ..., C_5^{(m)}$ are similarly defined. During training for task 1 the weights $\theta \in R^{1500\times 3630}$ are updated as usual according to

[1] The input neuron is assumed to have spiked *after* the output neuron spiked.

$$\Delta\theta = -\eta \frac{\partial C_1^{(m)}}{\partial \theta}. \tag{9.1}$$

After training is completed for task 1, we need to know the importance of each of the weights θ_{rs} for $r = 1, ..., 1500$ and $s = 1, ..., 3630$ in terms of classifying the images of task 1. This is necessary because when we proceed to train on task 2 these "important" weights should not be allowed to change significantly. That is the network must be forced to use the other weights for the training of task 2. Accordingly, we next define a *cost per synapse* regularizer during the training of task 2 to help prevent changes to the so called important weights of the task 1. The change in the cost per each synapse $\Delta C_{1,rs}^{(m)}$ is defined as

$$\Delta C_{1,rs}^{(m)} \triangleq \frac{\partial C_1^{(m)}}{\partial \theta_{rs}} \Delta\theta_{rs} = -\eta \left(\frac{\partial C_1^{(m)}}{\partial \theta_{rs}} \right)^2 \tag{9.2}$$

with

$$\Delta C_1^{(m)} \triangleq \left\{ \Delta C_{1,rs}^{(m)} \right\}_{\substack{r=1,...,1500 \\ s=1,...,3630}} \in \mathbb{R}^{1500 \times 3630} \tag{9.3}$$

For each task there are M mini-batches with P images per mini-batch for a total of $N = MP$ input images for each task. The average change in cost for θ_{rs} is given by

$$f_{1,rs} \triangleq \frac{1}{M} \sum_{m=1}^{M} \Delta C_{1,rs}^{(m)} = -\eta \frac{1}{M} \sum_{m=1}^{M} \left(\frac{\partial C_1^{(m)}}{\partial \theta_{rs}} \right)^2 \tag{9.4}$$

with

$$f_1 \triangleq \{ f_{1,rs} \}_{\substack{r=1,...,1500 \\ s=1,...,3630}} \in \mathbb{R}^{1500 \times 3630} \tag{9.5}$$

A softmax output layer with a cross-entropy cost function and one-hot encoded labels is the same as the log-likelihood cost function [66]. The MNIST label l with $l \in \{0, 1, 2, ..., 9\}$ corresponds to the $k^{th} (= l + 1)$ output neuron with $k \in \{1, 2, ..., 10\}$, respectively. Let $\mathbf{X}^{(m)} = \{(X^{(im)}, l_i), i = 1, ..., P\}$ denote the images and corresponding labels in the m^{th} mini-batch. In Equation (9.4) the average cost $C_1^{(m)}$ for mini-batch m is

$$C_1^{(m)} = \frac{1}{P}\sum_{i=1}^{P} C_1(\mathbf{X}^{(im)}) = -\frac{1}{P}\sum_{i=1}^{P}\sum_{k=1}^{n_o=2} y_k \ln a_k^{L5}(\mathbf{X}^{(im)}) = -\frac{1}{P}\sum_{(\mathbf{X}^{(im)},l_i)\in\mathbf{X}^m} \ln a_{l_i+1}^{L5}(\mathbf{X}^{(im)}) \tag{9.6}$$

as $y_{l+1} = 1$ and $y_k = 0$ for $k \neq l+1$. Here L5 indicates the last layer and $a^{L5}(\mathbf{X})$ indicates softmax output activations. Substituting Equation (9.6) in to Equation (9.4), Equation (9.5) becomes

$$f_1 = -\eta\frac{1}{M}\sum_{m=1}^{M}\left(\frac{\partial C_1^{(m)}}{\partial\theta}\right)^2 \in \mathbb{R}^{1500\times 3630} \tag{9.7}$$

In [37] the authors state that near a minimum of the cost the (r,s) component of Equation (9.7) given by $-\frac{1}{M}\sum_{m=1}^{M}\left(\frac{\partial C_1^{(m)}}{\partial\theta_{rs}}\right)^2$ is the same as

$$I_{rs}(\theta) \triangleq \frac{1}{M}\sum_{m=1}^{M}\frac{\partial^2 C_1^{(m)}}{\partial\theta_{rs}^2}, \tag{9.8}$$

with some limitations [41] and is the Fisher information [7] for the parameter θ_{rs}, $I_{rs}(\theta)$ is a measure of the "importance" of the weight θ_{rs}. A large value of $I_{rs}(\theta)$ implies that small changes in the value of θ_{rs} will lead to a large increase in average cost (classification error). When the network is to train for task 2, those weights θ_{rs} with a large $I_{rs}(\theta)$ computed from task 1 must now be constrained to only small changes so the network will continue to classify the images of task 1 correctly. That is, when training on task 2, the network must be forced to (essentially) use only those weights that had a small value of $I_{rs}(\theta)$ from task 1. So, the cost per synapse for the first task f_1 (calculated during the last epoch of training for the first task) gives the relative importance of the weights for the task1 classification problem. Let $\Delta\theta_1 \in \mathbb{R}^{1500\times 3630}$ be the change in weights during the last epoch of task 1. Further $\hat{\theta}_1$ denotes the value of the weights after training on task 1. The

second task is trained using the regularized cost function given by

$$C_2^{reg} \triangleq C_2 + \frac{\lambda}{2N} \sum_{rs} (\theta_2 - \hat{\theta}_1) \odot F_1 \odot (\theta_2 - \hat{\theta}_1) \tag{9.9}$$

with

$$F_1 \triangleq f_1 \oslash (\Delta\theta_1 \odot \Delta\theta_1 + \xi) \tag{9.10}$$

where $\odot$ and $\oslash$ represent the Hadamard product and division, respectively. ξ is a small positive number added to each element of the matrix to prevent division by zero when doing Hadamard division. Similarly, f_t is calculated during the last epoch of training task t, $\Delta\theta_t$ denotes the change in the weights during the last epoch, and finally $\hat{\theta}_t$ denotes the weights at the end of training task t. Task t is trained by adding a weight regularizing term to prevent the "important" weights from the previous tasks being changed significantly. With $f_0 \in \mathbb{R}^{1500\times3630}$ a matrix of zeros define

$$F_t \triangleq \sum_{\tau=0}^{t-1} f_\tau \oslash (\Delta\theta_\tau \odot \Delta\theta_\tau + \xi). \tag{9.11}$$

then we can write the cost function of the t^{th} task as

$$C_t^{reg} = C_t + \frac{\lambda}{2N} \sum_{rs} F_t \odot (\theta_t - \hat{\theta}_{t-1}) \odot (\theta_t - \hat{\theta}_{t-1}), \qquad \text{t=1,2,3,4,5} \tag{9.12}$$

$\hat{\theta}_{t-1}$ are the weights between L3 and L4 layers at the end of $(t-1)^{th}$ task and $N = MP$ is the number of input images. **Remark** Note that only the weights connecting L3 to L4 are subject to cost per synapse regularization. The weights connecting L4 to L5 are trained without regularization and use the AR1 method to train sequentially [54]. The parameter updates for the cost function are

$$\frac{\partial C_t^{reg}}{\partial \theta_t} = \frac{\partial C_t}{\partial \theta_t} + \frac{\lambda}{N} F_t \odot (\theta_t - \hat{\theta}_{t-1}) \tag{9.13}$$

In [100] the cost per synapse is calculated over all the training epochs (rather than just the last epoch).

9.5.1 Results with MNIST Dataset

The parameter λ in Equation (9.13) was optimized with validation data. Figure 9-8 shows the effect of λ on accuracy. Results for each λ were obtained from 10 different weight initializations. In this section the network was not presented with any

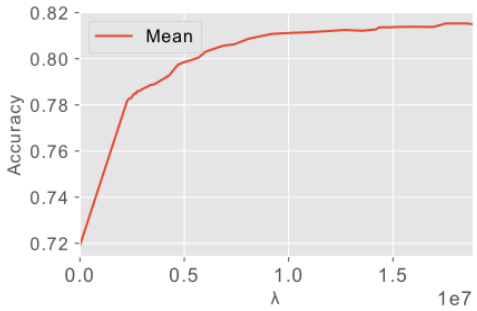

Figure 9-8: Search for λ.

of the data from the previous tasks. Figure 9-9 shows the trend of testing accuracy as the network is trained on disjoint tasks with 10 different weight initializations. The highest testing accuracy achieved for this disjointly trained tasks was 84.61%

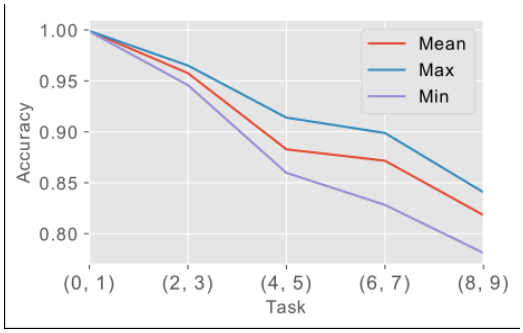

Figure 9-9: Test accuracy

and λ was set to 2.03×10^7. in Figure 9-9 'Max' in the legend indicates the weight initialization that resulted in highest test accuracy and 'Min' indicates the weight initialization that resulted in lowest test accuracy. For all of the above reported experiments the hyper-parameter $\eta = 1.0 \times 10^{-3}$, the mini-batch size $P = 10$ and the value of M was calculated based on the ratio of number of samples per task and the mini-batch size.

9.5.2 Results with EMNIST Dataset

The parameter λ in Equation (9.13) was optimized with validation data. Figure 9-10 shows the effect of λ on accuracy. Results for each λ were averaged from 10 different weight initializations. In this section the network was not presented

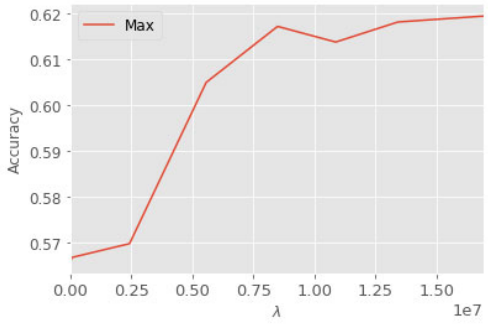

Figure 9-10: Search for λ.

with any of the data from the previous tasks. Figure 9-11 shows the trend of testing accuracy as the network is trained on disjoint tasks with 10 different weight initializations. The highest testing accuracy achieved for this disjointly trained tasks was

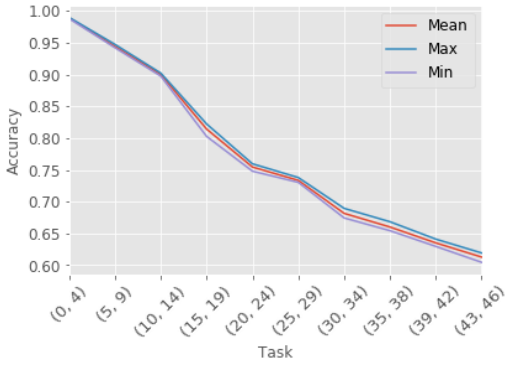

Figure 9-11: Trend of test accuracy as the learning progresses in an SIT scenario.

62.26% and λ was set to 1.65×10^7. in Figure 9-9 'Max' in the legend indicates the weight initialization that resulted in highest test accuracy and 'Min' indicates the weight initialization that resulted in lowest test accuracy. For all of the above reported experiments the hyper-parameter $\eta = 1.0 \times 10^{-3}$, the mini-batch size $P = 10$ and the value of M was calculated based on the ratio of number of samples per task and the mini-batch size.

In this chapter, we demonstrate that STDP trained CNN is resistant to catastrophic forgetting when compared to a non-spiking CNN. All the continual learning experiments in this chapter were performed using the MNIST and the EMNIST datasets.

CHAPTER TEN: MODELLING A CMOS IMAGE SENSOR USING NEURAL NETWORKS

10.1 Introduction

This chapter presents the internship work done at ON Semiconductor Inc, to model their image sensor using neural networks. Deep learning has been used in a plethora of applications like autonomous driving, cancer prediction, low power object recognition etc. [91] [92] [79]. In particular, neural networks as a regression tool have been used in applications like, time series learning [27], stock prediction [74], pose estimation in computer vision [42], cost predictions [85] etc. Traditionally, linear regression with linear or non-linear coefficients has been used for modeling where real valued outputs are required. Neural networks are iterative methods that minimize a loss function defined on the output layer of neurons. Universal approximation theorem states that a feed forward neural network with at least one hidden layer can approximate a continuous function of $\mathbb{R}^n$ [28]. Neural networks use error back-propagation with stochastic gradient descent (SGD) [45] to achieve an acceptable local minima that optimizes the output loss function.

Many industrial sensors require fine tuning of the input settings to attain a desired output. Figure 10-1 shows that the number of experiments to be conducted grows exponentially with the resolution and number of inputs to a sensor. In this work, we employ deep learning to model the relationship between inputs and outputs of a sensor that were collected at set intervals. Once a satisfactory model is achieved, it will be used to interpolate the outputs for any input combinations that are within an allowed range. Using appropriate optimization criteria we show that

one can arrive at input settings that maximize or minimize required outputs for a given sensor.

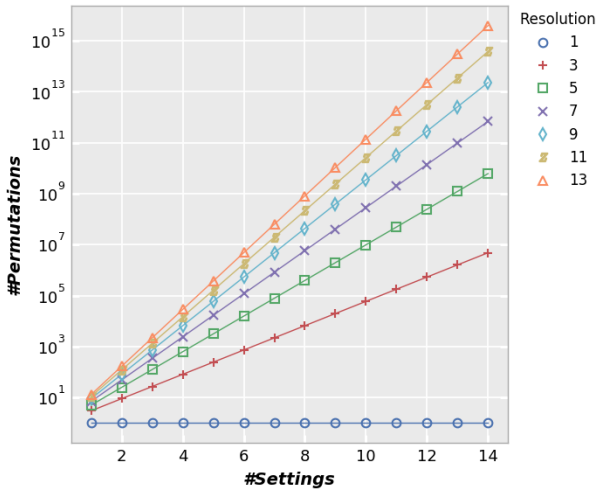

Figure 10-1: Resolution indicates number of values a particular setting can assume.

10.2 Data Visualization

Throughout the chapter, we shall use the image sensor data obtained from ON Semiconductor. Given a sensor has seven inputs and three outputs, six of the inputs are numerical and the seventh input is categorical and it can assume four possible values. Histograms of all the numerical inputs and outputs are shown in Figure 10-2. Each of the numerical inputs assumes five different values therefore we have a total of $5^5(3125)$ possible combinations. For each of the possible combinations, Input5 was swept from $0-49$. Categorical variable that assumes four unique values is not shown in Figure 10-2. Each of the input setting combinations yields a table (DataFrame) of $50 * 4(= 200)$ rows. Because there are 3125 possible setting combinations the output table contains $3125 * 200(= 625000)$ rows. Each row in Table 10.1 is applied as a setting combination to the sensor resulting in three outputs consisting of Signal, SNR and Output3. Therefore, the input to the neural network is $\in \mathbb{R}^{625000\times10}$ and the output is $\in \mathbb{R}^{625000\times3}$.

Table 10.1: Concerned sensor of this work was presented with all the combinations of Input1, Input2, Input3, Input4, Input6 values given in the table. For each of the combination, Input5 was swept from 0-49 obtaining a single Signal [AU] vs SNR [dB] curve. Note that the resolution of inputs for which outputs were recorded is 22, 8, 25, 200 and 200 respectively for Inputs 1, 2, 3, 4 and 6 respectively.

Input1	Input2	Input3	Input4	Input6
418	112	400	2850	3200
441	120	425	3050	3400
464	128	450	3250	3600
478	136	475	3450	3600
510	144	500	3650	4000

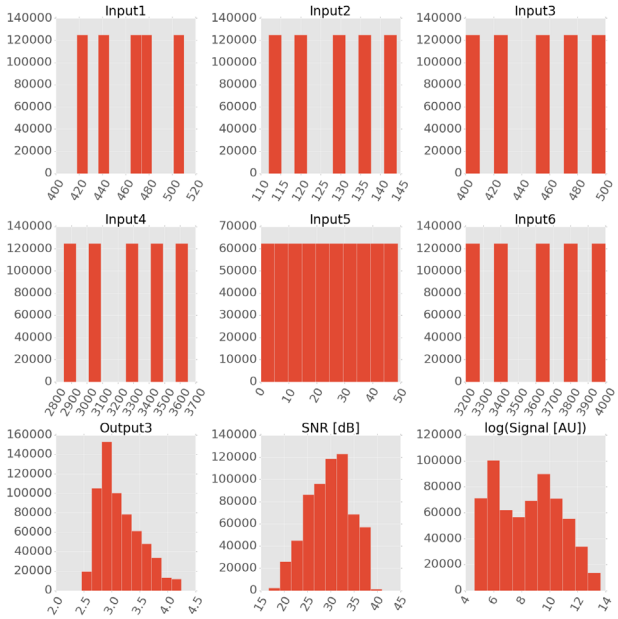

Figure 10-2: Histogram of all the inputs and outputs.

Figure 10-3 shows correlations between numerical inputs and outputs. Since the data is in higher dimensions ($\mathbb{R}^{10}$), we cannot visualize the relationship between inputs and outputs. However, we can plot the Signal vs SNR plot with at most two of the input settings varied. Signal [AU] vs SNR [dB] plot with varied Input1 and Input 2 is plotted in the Figure 10-4.

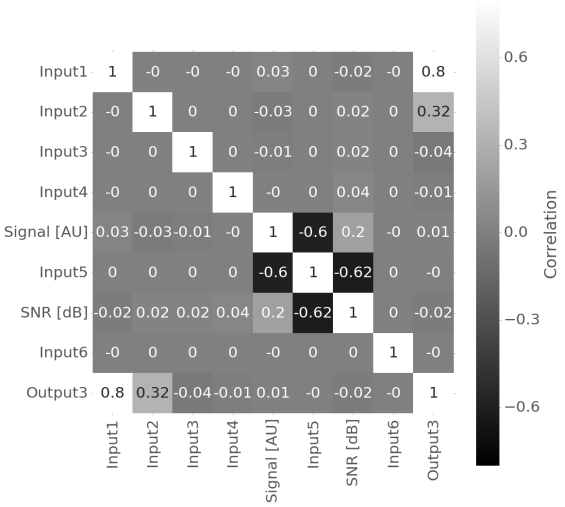

Figure 10-3: Correlation between various inputs and outputs.

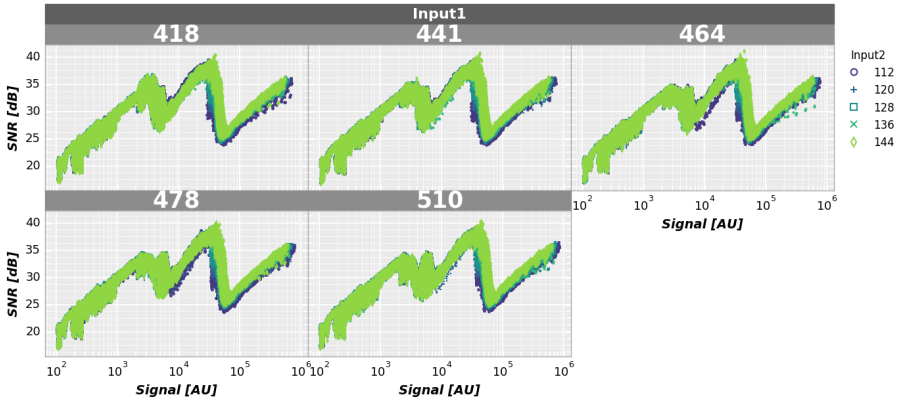

Figure 10-4: Plot of Signal [AU] vs SNR [dB] given Input1 and Input2.

10.3 Data Pre-processing

The Signal vs SNR relation of the data from the concerned sensor is approximately piecewise log linear with some non-linearities that are controlled by the inputs 1-6. This is shown in the Figure 10-4. Signal [AU] column of the dataframe was log transformed and all the inputs to the neural network were normalized by dividing the input with the maximum value that the input could assume. So, all the inputs to the neural network are in between 0 and 1 similarly, outputs were also normalized. All the data were converted to dataframes using Pandas [59]. Original and normalized sample sections of the dataframes are shown in Figures 10-5 and

10-6.

	Input1	Input2	Input3	Input4	Input5	Input6	Categ_Input1	Categ_Input2	Categ_Input3	Categ_Input4
405584	464	112	475	3050	0	3400	0	0	1	0
84221	441	144	400	3450	47	3400	0	0	1	0
338759	478	136	450	3450	32	3600	1	0	0	0
129873	510	144	425	2850	40	3200	1	0	0	0
188550	464	136	425	3250	19	3600	0	0	1	0

Figure 10-5: Un-normalized input data.

	Input1	Input2	Input3	Input4	Input5	Input6	Categ_Input1	Categ_Input2	Categ_Input3	Categ_Input4
430509	0.909804	0.777778	0.95	0.890411	0.693878	0.85	1.0	0.0	0.0	0.0
185603	0.937255	0.777778	0.85	0.890411	0.755102	0.90	0.0	0.0	1.0	0.0
58167	0.819608	0.944444	0.80	0.890411	0.346939	0.85	1.0	0.0	0.0	0.0
411725	0.937255	0.833333	0.95	0.835616	0.918367	0.90	0.0	1.0	0.0	0.0
14978	1.000000	1.000000	0.80	0.780822	0.326531	0.90	1.0	0.0	0.0	0.0

Figure 10-6: Normalized input data.

10.4 Neural Network

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1536)              16896
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 1536)              0
_________________________________________________________________
dense_2 (Dense)              (None, 768)               1180416
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 768)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               393728
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 1539
=================================================================
Total params: 1,592,579
Trainable params: 1,592,579
Non-trainable params: 0
_________________________________________________________________
```

Figure 10-7: Keras sumary of the final neural network that was used to train the data.

A neural network with three hidden layers, mean squared error cost function and a leaky ReLU activation function was chosen. Our network has 10 input and

3 output neurons which are determined by the dataset. Training was performed using Keras [8] with Tensorflow [1] back end. Network's Keras summary is given in Figure 10-7.

10.4.1 How to Choose Neural Network Parameters ?

Motivated by universal approximation theorem we started with one hidden layer with sigmoid activations and we found that using two hidden layers results in faster convergence. One of the most important hyper-parameters for neural networks is the learning rate (α). We started (α) with a value of 0.0005. Mean absolute error (MAE) was chosen as the cost function and by experimentation we found that MAE cost function and sigmoid neurons delayed the convergence. Weights were initialized with Glorot or He initialization depending on the activation functions. Glorot initialization is beneficial for reducing the hidden neuron activation's variance for sigmoid neurons [20] and He initialization helps networks with ReLU activation functions [25]. In deep neural networks it is desirable to have similar variance for activations and gradients of the neurons in hidden layers.

Glorot Initialization

Weights between two layers are initialized with a normal distribution with mean, $\mu = 0$ and standard deviation

$$\sigma = \sqrt{\frac{2}{fan_{in} + fan_{out}}}$$

where fan_{in} and fan_{out} are number of neurons in incoming and outgoing layers. If a uniform distribution is used then the weights are sampled from

$$U(-\sqrt{\frac{6}{fan_{in} + fan_{out}}}, \sqrt{\frac{6}{fan_{in} + fan_{out}}})$$

Note that our network in this work utilizes Glorot initialization with normal distribution

He Initialization

He initialization is used for ReLU units. If a normal distribution is used then the weights are sampled from a distribution with mean, $\mu = 0$ and standard deviation,

$$\sigma = \sqrt{\frac{2}{fan_{in}}}$$

If a uniform distribution is used then the weights are sampled from

$$U(-\sqrt{\frac{6}{fan_{in}}}, \sqrt{\frac{6}{fan_{in}}})$$

Usually, a ReLU activation is used for efficient error back-propagation. However, care should be taken when using ReLU activation, if a single "bad" weight update results in negative activations for majority of the neurons in a layer then the majority of the gradients will be nullified as ReLU function is zero for negative inputs. Hence, the network will not be able to backpropagate the error from the final layer. Equation 8.2 is modified accordingly. If a ReLU activation function is chosen then

$$\sigma(z) = \begin{cases} z, & \text{if } z > 0 \\ 0, & o.w \end{cases}$$

Its derivative is given by

$$\sigma'(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & o.w \end{cases}$$

$$\delta^L = \begin{cases} -(y - a^L) \odot \sigma'^L), & \text{if } z^L > 0 \\ 0, & \text{if } z^L \leq 0 \end{cases}$$

If a component of the vector z^L is negative then the corresponding component of δ^L is zeroed out resulting in no error back-propagation. Gradient vectors were normalized to have a max value of one so that a single gradient update with larger negative

values doesn't drive the net inputs of neuron to have a negative value resulting in a zero activation value. This results in no forward propagation resulting in "dead neuron phenomenon" and this issue can be mitigated by using Leaky ReLU activation function. We found that Leaky ReLU with Mean squared Error (MSE) cost function gives a faster convergence when compared to Sigmoid activations with MSE or MAE. Leaky ReLU is able to back-propagate both positive and negative components of the δ^l it is given by

$$\sigma(z) = \begin{cases} z, & \text{if } z > 0 \\ \alpha z, & o.w \end{cases}$$

Where α was set to a small value, 0.01. Derivative of Leaky ReLU is given by

$$\sigma'(z) = \begin{cases} 1, & \text{if } z > 0 \\ \alpha, & o.w \end{cases}$$

10.4.2 Modeling

Data were split into training (81%), validation (9%) and testing (10%). Our network was trained for 100 epochs and learning rate was reduced by a factor of two for every consecutive five epochs if the validation error did not decrease. Figure 10-8 shows the progress of the network in learning the dataset.

10.4.3 Prediction and Evaluation

Once the modeling was done, training data, testing data and validation data were passed through the network to obtain the predictions for the required outputs (SNR[dB], Signal [AU], Output3). Note that the model/network has not "seen" the testing data directly and validation data was "seen " indirectly in that it was used to optimize for the learning rate. Figure 10-9 shows the Actual vs Predicted plot for SNR[dB] and it is a linear plot indicating that the model was successful in recalling

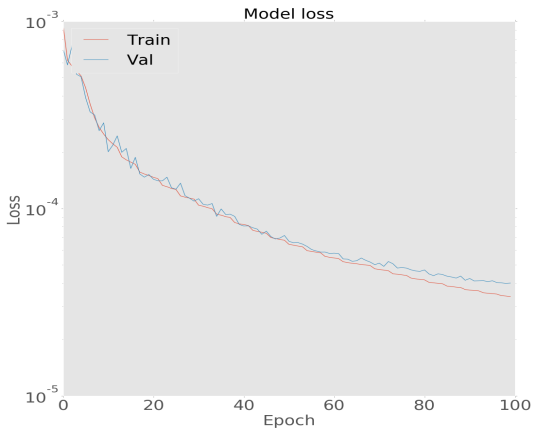

Figure 10-8: Epochs vs Loss plot of the neural network.

the SNR[dB] values.

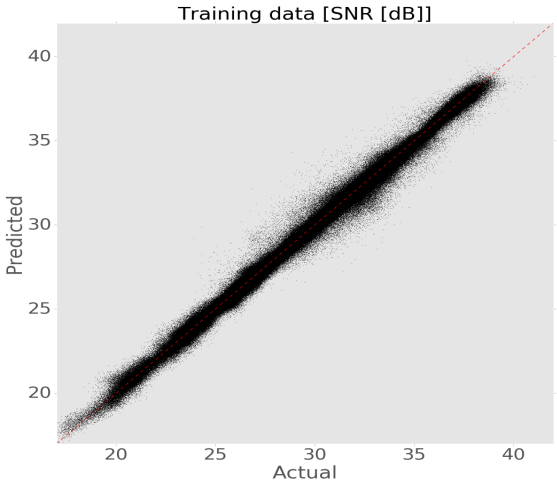

Figure 10-9: Actual vs Predicted plot for SNR [dB] in the training dataset. Goodness of fit (R^2) was found to be 0.991

Figure 10-10 shows that predicted SNR [dB] values were quite close to the actual SNR [dB] values of the testing set. Figures 10-11, 10-12, 10-13, 10-14 show Actual vs Predicted plots of Signal [AU], Output3 for training and testing datasets respectively.

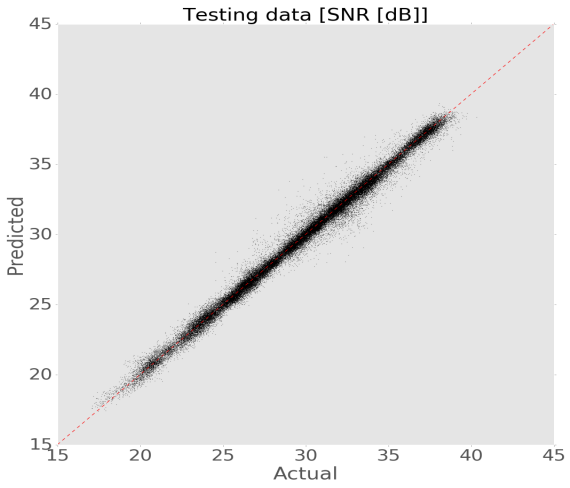

Figure 10-10: Actual vs Predicted plot for SNR [dB] in the testing dataset. Goodness of fit (R^2) was found to be 0.990

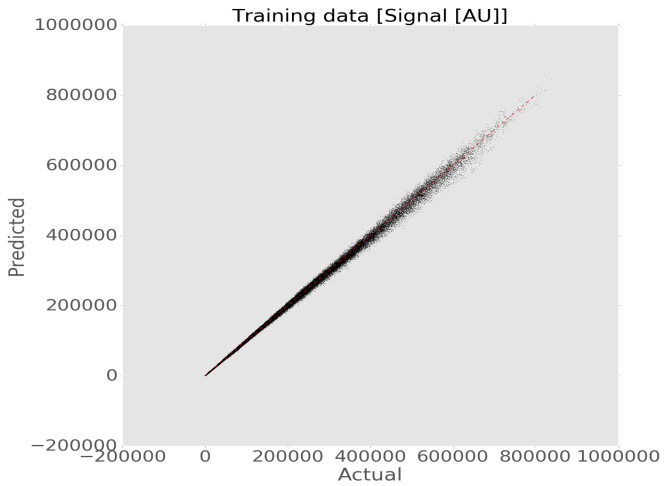

Figure 10-11: Actual vs Predicted plot for Signal [AU] in the training dataset. Goodness of fit (R^2) was found to be 0.999

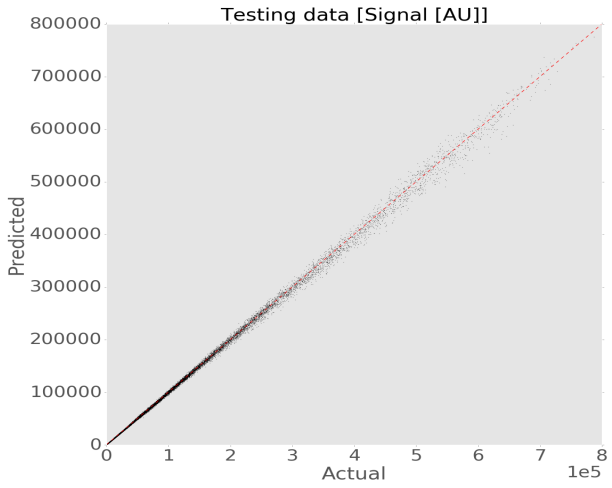

Figure 10-12: Actual vs Predicted plot for Signal [AU] in the testing dataset. Goodness of fit (R^2) was found to be 0.999

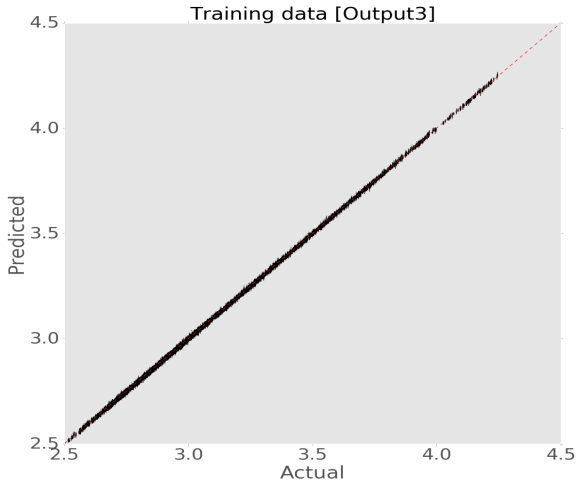

Figure 10-13: Actual vs Predicted plot for Output3 in the training dataset. Goodness of fit (R^2) was found to be 0.999

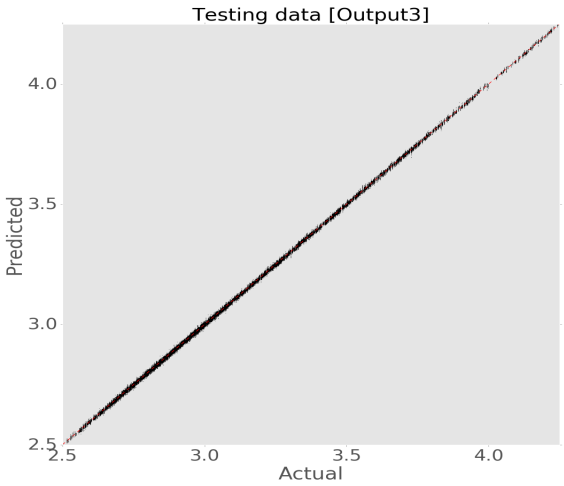

Figure 10-14: Actual vs Predicted plot for Output3 in the testing dataset. Goodness of fit (R^2) was found to be 0.999

10.5 Optimization

The goal of the optimization process is to obtain a settings combination (of Input1, Input2, Input3, Input4 and Input6) that results in a Signal [AU] vs SNR [dB] curve that is closest to the ideal and minimize the value of Output3. For the sensor under consideration, the ideal $SNR[dB] = 10\log_{10}(\sqrt{Signal[AU]})$. Each of the settings combinations (of Input1, Input2, Input3, Input4 and Input6) results in a dataframe of 200 rows because Input6 is swept from $0-49$ and the categorical variable assumes four different categories and each of these dataframes yields a single Signal [AU] vs SNR [dB] curve. Note that Signal [AU], SNR [dB] and Output3 are the outputs of the trained neural network. The trained model was used to predict Signal [AU] vs SNR [dB] plots for a large number ($\approx 12 \times 10^6$) of interpolated settings combinations within the domains of all the input settings, to that end we increased the resolution of the numerical inputs listed in Table 10.1. Similar to the original dataset, each of the interpolated input settings combinations also yields a single Signal [AU] vs SNR [dB] curve. Shown in Figure 10-15 is a Signal [AU] vs SNR [dB] curve for a randomly chosen interpolated input settings, green and blue colors indicate ideal and predicted Signal [AU] vs SNR [dB] relationships. In this case, Input1, Input2, Input3, Input4 and Input6 happened to be 418, 112, 400, 2850 and 3200 respectively and the value of Output3 is 2.9365. The green colored line indicates the fitted line of Signal [AU] with SNR [dB] for Signal [AU] values that are less than 2×10^3 .

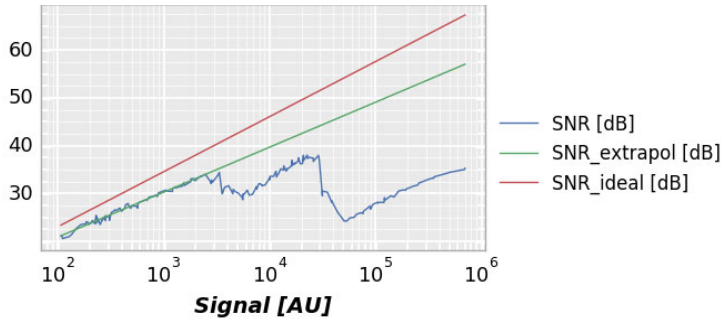

Figure 10-15: Plot of Signal [AU] vs SNR [dB].

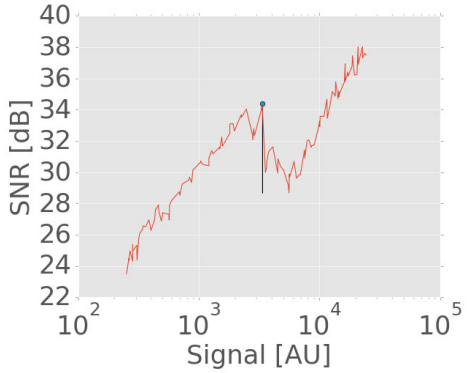

Figure 10-16: Plot of Signal [AU] vs SNR [dB] in the interval $\approx 3 \times 10^3 - 10^4$ AU from the sensor for a single settings combination. Recorded prominence (SNR[dB] drop) value for this settings combination was $\approx$ 5.77 dB. One of the methods of optimization is to choose the settings combination that produces the least SNR drop in the interval $\approx 3 \times 10^3 - 10^4$ AU.

The blue curve in Figure 10-15 shows a linear relationship until Signal [AU] reaches $\approx 3 \times 10^3$ AU. Ideally, we expect this behavior to continue for the rest of the Signal values. A sudden dip of $\approx$ 5 dB (see Figure 10-16) is noticeable when the Signal [AU] value is in the range, $\approx 3 \times 10^3 - 10^4$ AU. Since it is highly unlikely to achieve an ideal performance, we set a few criteria (heuristics) to choose a particular settings combination that could give the smallest prominence in the SNR value at the interval $\approx 3 \times 10^3 - 10^4$ AU and a Signal [AU] vs SNR [dB] curve that is closest to the ideal Signal [AU] vs SNR [dB] curve. The best interpolated input combination was filtered by applying different criterion described below. Lower values are preferred for all the criteria except for criteria 4 and 5.

- **MAE between ideal and predicted (criterion 1):** MAE was calculated for each of the settings combinations and serial numbers of each of the dataframes (a single settings combination) was ordered in an ascending order of the calculated MAEs.

- **Prominence of SNR dip (criterion 2):** Serial numbers of each of the dataframes (a single settings combination) was ordered in an ascending order of the calculated prominence values at $\approx 3 \times 10^3 - 10^4$ AU.

- **MAE between fitted line and predicted (criterion 3):** Serial numbers of

each of the dataframes (of a single settings combination) was ordered in an ascending order of the calculated MAE between fitted green line and predicted blue curve of Figure 10-15. Green line was fitted for Signal [AU] vs SNR [dB] up to $\approx 3\times10^3 - 10^4$ AU and extrapolated for the rest of the Signal [AU] values.

- **Area under curve (criterion 4):** Serial numbers of each of the dataframes (of a single settings combination) was ordered in an ascending order of the calculated area under the curve of SNR [dB] vs Signal [AU].

- **Minimum SNR value for the second transition (criterion 5):** Serial numbers of each of the dataframes (a single settings combination) was ordered in descending order of the calculated minimum SNR [dB] for Signal [AU] greater than $\approx 10^4$ AU.

- **Least value for Output3 (criterion 6):** Serial numbers of each of the dataframes (a single settings combination) was ordered in an ascending order of the calculated Output3 value.

The first index among the intersection of all the indices obtained from the above steps gives the optimal input setting combination with a Signal [DN] vs SNR [dB] curve that meets all the above criteria. Figure 10-17 shows the optimized curve.

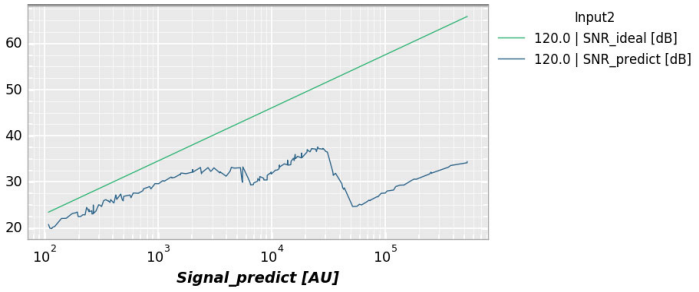

Figure 10-17: Plot of Signal [AU] vs SNR [dB] for the corresponding input settings combination that resulted in a Signal [AU] vs SNR [dB curve close to the ideal Signal [AU] vs SNR [dB]. Input1, Input2, Input3, Input4, Input6 were found to be $430, 120, 485, 2900, 3525$ respectively. Optimization was performed using all the six criteria mentioned above.

Table 10.2: Various criteria values when the predictions where optimized for Signal [AU] vs SNR [dB] curve and Output3.

criterion 1	criterion 2	criterion 3	criterion 5	criterion 6
1167.50	3.9	384.73	24.66	2.64

Table 10.2 shows the numerical values of different criteria used in the optimization process. If criterion 6 was excluded from the optimization criteria (i.e., Signal [AU] vs SNR [dB] curve not optimized for Output3) then the Signal [AU] vs SNR [dB] is shown in Figure 10-18 and corresponding values of criteria are shown in Table 10.3.

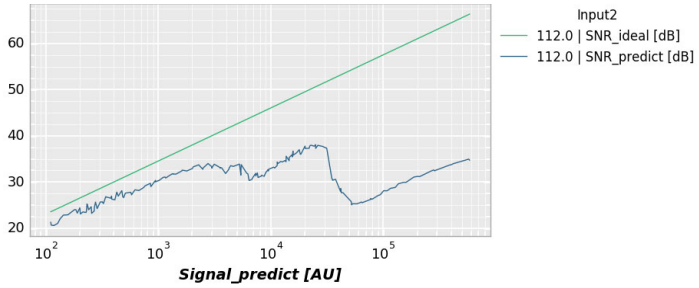

Figure 10-18: Plot of Signal [AU] vs SNR [dB] for the corresponding input settings combination that resulted in a Signal [AU] vs SNR [dB] curve close to the ideal Signal [AU] vs SNR [dB]. Input1, Input2, Input3, Input4, Input6 were found to be $426, 112, 495, 3000, 3600$ respectively. Optimization was performed using all the six criteria mentioned above.

Table 10.3: Various criteria values when the predictions where optimized only for Signal [AU] vs SNR [dB] curve.

criterion 1	criterion 2	criterion 3	criterion 5	criterion 6
1043.09	3.68	372.75	25.03	2.88

Figure 10-18 was obtained by optimizing for only Signal [AU] vs SNR [dB] curve. Hence, criterion 6 of Table 10.3 shows higher value than that of criterion 6 of Table 10.2.Many of the data pre-processing steps were parallelized using python multiprocessing and neural network training and inference was performed on NVIDIA TITAN GPUs. Effectively, we were able to cut down the time taken for characterization from ≈15 days to ≈2 days.

In this chapter, we applied modern deep learning tools and methods to reduce

the time taken to characterize an image sensor. Specifically, we used a neural network as a function approximator to model the relationship between inputs and outputs of an image sensor.

CHAPTER ELEVEN: CONCLUSION

In this work we employed spiking neural networks (SNNs) as an alternative to deep neural networks (DNNs) to study the possibility of energy efficient neural networks for classification tasks. We developed the required software tool (SPYKEFLOW) [94] to facilitate various experiments conducted in this work. We documented the effect of various hyper-parameters on SNN's learning abilities and we also explored the abilities of SNNs in continual learning tasks. We proposed surrogate gradients to classify the extracted spiking features for energy-efficient neuromorphic/in-memory devices. In the last chapter, we presented the internship work done at ON semiconductor for image sensor characterization using modern deep learning tools.

11.1 Summary

11.1.1 Chapter 4

In this chapter, we showed that MNIST training data $\in \mathbb{R}^{50000\times784}$ transformed to max-pooled neuron potentials $\in \mathbb{R}^{50000\times500}$ after passing through an SNN with two convolution and two pooling layers (2c2p) becomes linearly separable by an SVM. We also showed that original MNIST training data $\in \mathbb{R}^{50000\times784}$ is not linearly separable by an SVM. These experiments were conducted to illustrate that SNNs aid the separability of the input data.

11.1.2 Chapter 5

In this chapter, we examine Reinforced-STDP (R-STDP) as a classification criteria for spike or membrane voltage feature vectors. We conclude that a simple

linear neural network (without a hidden layer) trained with error backpropagation performed better than R-STDP.

11.1.3 Chapter 6

In this chapter, we use spikes obtained directly from a silicon retina (ATIS) to train an SNN instead of using synthetically generated spikes from MNIST images. We also show the results for transfer learning experiments conducted with a network trained on synthetically generated spikes and tested on spikes from ATIS.

11.1.4 Chapter 7

In this chapter, we discuss the over training problem that arises when using unsupervised-STDP. We show that over training results in reduction in complexity of the features learned in deeper layers and it can result in loss of classification accuracy. We also presented a heuristic method to prevent over training.

11.1.5 Chapter 8

In this chapter, we introduce binary activations for the classification sections of an SNNs. Subsequently, we introduce two different methods to calculate surrogate gradients for neurons with non-differentiable activation functions. We also showcase that binary activations and surrogate gradients help in significantly reducing the number of high-precision floating point multiplications. For example, all the calculations in a matrix (floating point)-vector (binary) multiplication can be performed by simply choosing rows/columns.

11.1.6 Chapter 9

In this chapter, we demonstrate catastrophic forgetting in a DNN with 1c1p1fcn structure. We also demonstrate that an SNN with same structure as a DNN is relatively more resistant to catastrophic forgetting. Subsequently, we also introduce

"importance per synapse" metric to immunize the classification sections of an SNN against catastrophic forgetting in a single incremental task (SIT) scenario.

11.1.7 Chapter 10

In this chapter, we applied modern deep learning tools and methods to characterize an image sensor We demonstrate that a neural network can be used as a function approximator to accelerate the characterization of an industrial sensor.

11.2 Future work

11.2.1 Time Dependent Classifier

In this work we used a binarized non-spiking DNN for classifying spike feature vectors extracted using a spiking network. This work can be extended further by having a feature classifier that can preserve the time information. This can potentially increase the classification accuracy. Further, time dependent classifier will enable the network to use labelled and unlabeled data thereby making this approach a suitable candidate for semi-supervised learning applications.

11.2.2 Hardware Implementation

Current state of the art DNNs are not hardware friendly and their power consumption rates are not suitable for edge computing. The methods presented here are a suitable candidate for implementation on hardware.

11.2.3 Learning Spike Times

In this work we used rank order coding and Difference of Gaussian (DoG) to generate spikes, this approach limits the learnable texture information in the input images which limits the problem solving ability of the rank order coding based spiking networks. Methods that can learn the spike times such as BS4NN [36]

without using DoG can potentially enable spiking neural networks to solve complex datasets such as ImageNet [40] and CIFAR100 [39]. Combining surrogate gradients and methods that learn spike times can be promising approach to solve complex datasets.

REFERENCES

[1] M. ABADI, A. AGARWAL, P. BARHAM, E. BREVDO, Z. CHEN, C. CITRO, G. S. CORRADO, A. DAVIS, J. DEAN, M. DEVIN, S. GHEMAWAT, I. GOODFELLOW, A. HARP, G. IRVING, M. ISARD, Y. JIA, R. JOZEFOWICZ, L. KAISER, M. KUDLUR, J. LEVENBERG, D. MANE, R. MONGA, S. MOORE, D. MURRAY, C. OLAH, M. SCHUSTER, J. SHLENS, B. STEINER, I. SUTSKEVER, K. TALWAR, P. TUCKER, V. VANHOUCKE, V. VASUDEVAN, F. VIEGAS, O. VINYALS, P. WARDEN, M. WATTENBERG, M. WICKE, Y. YU, AND X. ZHENG. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (2016).

[2] J. M. ALLRED AND K. ROY. Controlled Forgetting: Targeted Stimulation and Dopaminergic Plasticity Modulation for Unsupervised Lifelong Learning in Spiking Neural Networks. *Frontiers in Neuroscience* **14**, 7 (2020).

[3] N. ANWANI AND B. RAJENDRAN. NormAD - Normalized Approximate Descent based supervised learning rule for spiking neurons. In "2015 International Joint Conference on Neural Networks (IJCNN)", pp. 1–8 (July 2015).

[4] A. BALDOMINOS, Y. SAEZ, AND P. ISASI. A Survey of Handwritten Character Recognition with MNIST and EMNIST. *Applied Sciences* **9**(15) (2019).

[5] T. BEKOLAY, J. BERGSTRA, E. HUNSBERGER, T. DEWOLF, T. STEWART, D. RASMUSSEN, X. CHOO, A. VOELKER, AND C. ELIASMITH. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics* **7**, 48 (2014).

[6] D. BUTTS, C. WENG, J. JIN, C.-I. YEH, N. A LESICA, J.-M. ALONSO, AND G. STANLEY. Temporal precision in the neural code and the timescales of natural vision. *Nature* **449**, 92–5 (10 2007).

[7] J. N. CHIASSON. "Introduction to probability theory and stochastic processes". Wiley (2013).

[8] F. CHOLLET. "Deep Learning with Python". Manning Publications Co., USA, 1st ed. (2017).

[9] G. COHEN, S. AFSHAR, J. TAPSON, AND A. VAN SCHAIK. EMNIST: an extension of MNIST to handwritten letters. *arXiv e-prints* p. arXiv:1702.05373 (Feb. 2017).

[10] J. CONRADT, R. BERNER, M. COOK, AND T. DELBRUCK. An embedded AER dynamic vision sensor for low-latency pole balancing. In "2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops", pp. 780–785 (Sep. 2009).

[11] M. COURBARIAUX, I. HUBARA, D. SOUDRY, R. EL-YANIV, AND Y. BENGIO. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1 (2016).

[12] M. DAVIES, N. SRINIVASA, T.-H. LIN, G. CHINYA, P. JOSHI, A. LINES, A. WILD, AND H. WANG. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* **PP**, 1–1 (01 2018).

[13] A. DELORME, L. PERRINET, AND S. J. THORPE. Networks of integrate-and-fire neurons using Rank Order Coding B: Spike timing dependent plasticity and emergence of orientation selectivity. *Neurocomputing* **38-40**, 539 – 545 (2001). Computational Neuroscience: Trends in Research 2001.

[14] P. DIEHL AND M. COOK. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience* **9**, 99 (2015).

[15] P. FERRÉ, F. MAMALET, AND S. J. THORPE. Unsupervised Feature Learning With Winner-Takes-All Based STDP. *Frontiers in Computational Neuroscience* **12**, 24 (2018).

[16] R. FLORIAN. Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity. *Neural computation* **19**, 1468–502 (07 2007).

[17] K. FUKUSHIMA. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics* **36**(4), 193–202 (Apr 1980).

[18] A. GEPPERTH AND C. KARAOGUZ. A Bio-Inspired Incremental Learning Architecture for Applied Perceptual Problems. *Cognitive Computation* **8**(5), 924–934 (Oct 2016).

[19] R. GIRSHICK, J. DONAHUE, T. DARRELL, AND J. MALIK. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv e-prints* p. arXiv:1311.2524 (Nov 2013).

[20] X. GLOROT AND Y. BENGIO. Understanding the difficulty of training deep feedforward neural networks. In Y. W. TEH AND M. TITTERINGTON, editors, "Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics", vol. 9 of "Proceedings of Machine Learning Research", pp. 249–256, Chia Laguna Resort, Sardinia, Italy (13–15 May 2010). PMLR.

[21] T. GOLLISCH AND M. MEISTER. Rapid Neural Coding in the Retina with Relative Spike Latencies. *Science* **319**(5866), 1108–1111 (2008).

[22] S. GROSSBERG. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science* **11**(1), 23 – 63 (1987).

[23] A. GUPTA AND L. LONG. Character Recognition using Spiking Neural Networks. *IEEE International Conference on Neural Networks - Conference Proceedings* pp. 53 – 58 (09 2007).

[24] D. HASSABIS, D. KUMARAN, C. SUMMERFIELD, AND M. BOTVINICK. Neuroscience-Inspired Artificial Intelligence. *Neuron* **95**, 245–258 (07 2017).

[25] K. HE, X. ZHANG, S. REN, AND J. SUN. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification (2015).

[26] M. HINES AND T. CARNEVALE. "NEURON Simulation Environment", pp. 1–8. Springer New York, New York, NY (2013). In Encyclopedia of Computational Neuroscience, Jaeger, Dieter and Jung, Ranu, Editors.

[27] S. HOCHREITER AND J. SCHMIDHUBER. Long Short-Term Memory. *Neural Computation* **9**(8), 1735–1780 (1997).

[28] K. HORNIK, M. STINCHCOMBE, AND H. WHITE. Multilayer feedforward networks are universal approximators. *Neural Networks* **2**(5), 359 – 366 (1989).

[29] C.-W. HSU, C.-C. CHANG, AND C.-J. LIN. A Practical Guide to Support Vector Classification. (November 2003).

[30] D. H. HUBEL AND T. N. WIESEL. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology* **160**(1), 106–154 (1962).

[31] U. JARAMILLO-AVILA, H. ROSTRO-GONZALEZ, L. A. CAMUÑAS-MESA, R. DE JESUS ROMERO-TRONCOSO, AND B. LINARES-BARRANCO. An address Event Representation-Based Processing System for a Biped Robot. *International Journal of Advanced Robotic Systems* **13**(1), 39 (2016).

[32] Y. JIN, P. LI, AND W. ZHANG. Hybrid Macro/Micro Level Backpropagation for Training Deep Spiking Neural Networks. *arXiv-eprints* (05 2018).

[33] S. R. KHERADPISHEH. private communication.

[34] S. R. KHERADPISHEH, M. GANJTABESH, AND T. MASQUELIER. Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition. *Neurocomputing* **205**, 382 – 392 (2016).

[35] S. R. KHERADPISHEH, M. GANJTABESH, S. J. THORPE, AND T. MASQUELIER. STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks* **99**, 56 – 67 (2018).

[36] S. R. KHERADPISHEH AND T. MASQUELIER. S4NN: temporal backpropagation for spiking neural networks with one spike per neuron (2019).

[37] J. KIRKPATRICK, R. PASCANU, N. RABINOWITZ, J. VENESS, G. DESJARDINS, A. A. RUSU, K. MILAN, J. QUAN, T. RAMALHO, A. GRABSKA-BARWINSKA, D. HASSABIS, C. CLOPATH, D. KUMARAN, AND R. HADSELL. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* **114**(13), 3521–3526 (2017).

[38] M. KISELEV. Rate coding vs. temporal coding - is optimum between? In "2016 International Joint Conference on Neural Networks (IJCNN)", pp. 1355–1359 (July 2016).

[39] A. KRIZHEVSKY. Learning Multiple Layers of Features from Tiny Images. *University of Toronto* (05 2012).

[40] A. KRIZHEVSKY, I. SUTSKEVER, AND G. E. HINTON. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems* **25** (01 2012).

[41] F. KUNSTNER, L. BALLES, AND P. HENNIG. Limitations of the Empirical Fisher Approximation for Natural Gradient Descent (2019).

[42] S. LATHUILIÉRE, P. MESEJO, X. ALAMEDA-PINEDA, AND R. HORAUD. A comprehensive analysis of deep regression (2018).

[43] Y. LECUN, Y. BENGIO, AND G. HINTON. Deep Learning. *Nature* **521**, 436–44 (05 2015).

[44] Y. LECUN, L. BOTTOU, Y. BENGIO, AND P. HAFFNER. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**(11), 2278–2324 (Nov 1998).

[45] Y. LECUN, L. BOTTOU, Y. BENGIO, AND P. HAFFNER. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**(11), 2278–2324 (Nov 1998).

[46] Y. LECUN, C. CORTES, AND C. BURGES. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* **2** (2010).

[47] 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* **12**, 435 (08 2018).

[48] J. H. LEE, T. DELBRUCK, AND M. PFEIFFER. Training Deep Spiking Neural Networks Using Backpropagation. *Frontiers in Neuroscience* **10**, 508 (2016).

[49] R. LEGENSTEIN, D. PECEVSKI, AND W. MAASS. Theoretical Analysis of Learning with Reward-Modulated Spike-Timing-Dependent Plasticity. *arXiv e-prints* **20** (01 2007).

[50] Q. LIAO, J. Z. LEIBO, AND T. POGGIO. How Important is Weight Symmetry in Backpropagation? *arXiv e-prints* p. arXiv:1510.05067 (Oct 2015).

[51] 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* **43**(2), 566–576 (2008).

[52] T. LILLICRAP, D. COWNDEN, D. TWEED, AND C. J. AKERMAN. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications* **7**, 13276 (11 2016).

[53] Q. LIU, G. PINEDA-GARCĨA, E. STROMATIAS, T. SERRANO-GOTARREDONA, AND S. B. FURBER. Benchmarking Spike-Based Visual Recognition: A Dataset and Evaluation. *Frontiers in Neuroscience* **10**, 496 (2016).

[54] D. MALTONI AND V. LOMONACO. Continuous learning in single-incremental-task scenarios. *Neural Networks* **116**, 56 – 73 (2019).

[55] H. MARKRAM, W. GERSTNER, AND P. J. SJÖSTRÖM. Spike-Timing-Dependent Plasticity: A Comprehensive Overview. *Frontiers in Synaptic Neuroscience* **4**, 2 (2012).

[56] T. MASQUELIER. "Spike-based computing and learning in brains, machines, and visual systems in particular (HDR Report)". PhD thesis, Université Toulouse III - Paul Sabatier (10 2017).

[57] T. MASQUELIER, R. GUYONNEAU, AND S. J. THORPE. Spike Timing Dependent Plasticity Finds the Start of Repeating Patterns in Continuous Spike Trains. *PLOS ONE* **3**(1), 1–9 (01 2008).

[58] T. MASQUELIER AND S. J. THORPE. Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity. *PLoS Computational Biology* **3**, 1762 – 1776 (2007).

[59] W. MCKINNEY ET AL.. Data structures for statistical computing in python. In "Proceedings of the 9th Python in Science Conference", vol. 445, pp. 51–56. Austin, TX (2010).

[60] B. MEFTAH, O. LEZORAY, AND A. BENYETTOU. Segmentation and Edge Detection Based on Spiking Neural Network Model. *Neural Processing Letters* **32**(2), 131–146 (Oct 2010).

[61] M. MOZAFARI, M. GANJTABESH, A. NOWZARI, S. THORPE, AND T. MASQUELIER. Combining STDP and Reward-Modulated STDP in Deep Convolutional Spiking Neural Networks for Digit Recognition. *arXiv e-prints* (03 2018).

[62] M. MOZAFARI, M. GANJTABESH, A. NOWZARI-DALINI, AND T. MASQUELIER. SpykeTorch: Efficient Simulation of Convolutional Spiking Neural Networks With at Most One Spike per Neuron. *Frontiers in Neuroscience* **13**, 625 (2019).

[63] M. MOZAFARI, S. R. KHERADPISHEH, T. MASQUELIER, A. NOWZARI-DALINI, AND M. GANJTABESH. First-Spike-Based Visual Categorization Using Reward-Modulated STDP. *IEEE Transactions on Neural Networks and Learning Systems* **29**(12), 6178–6190 (Dec 2018).

[64] E. O. NEFTCI, C. AUGUSTINE, S. PAUL, AND G. DETORAKIS. Event-Driven Random Back-Propagation: Enabling Neuromorphic Deep Learning Machines. *Frontiers in Neuroscience* **11**, 324 (2017).

[65] B. NESSLER, M. PFEIFFER, L. BUESING, AND W. MAASS. Bayesian Computation Emerges in Generic Cortical Microcircuits through Spike-Timing-Dependent Plasticity. *PLOS Computational Biology* **9**(4), 1–30 (04 2013).

[66] M. A. NIELSEN. Neural Networks and Deep Learning (Jan 2015).

[67] G. ORCHARD, A. JAYAWANT, G. K. COHEN, AND N. THAKOR. Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades. *Frontiers in Neuroscience* **9**, 437 (2015).

[68] A. ORORBIA. Spiking Neural Predictive Coding for Continual Learning from Data Streams (2019).

[69] P. PANDA, A. AKETI, AND K. ROY. Towards Scalable, Efficient and Accurate Deep Spiking Neural Networks with Backward Residual Connections, Stochastic Softmax and Hybridization (2019).

[70] G. I. PARISI, R. KEMKER, J. L. PART, C. KANAN, AND S. WERMTER. Continual lifelong learning with neural networks: A review. *Neural Networks* **113**, 54 – 71 (2019).

[71] L. PAULUN, A. WENDT, AND N. KASABOV. A Retinotopic Spiking Neural Network System for Accurate Recognition of Moving Objects Using NeuCube and Dynamic Vision Sensors. *Frontiers in Computational Neuroscience* **12**, 42 (2018).

[72] M. PFEIFFER AND T. PFEIL. Deep Learning With Spiking Neurons: Opportunities and Challenges. *Frontiers in Neuroscience* **12**, 774 (2018).

[73] C. POSCH, D. MATOLIN, R. WOHLGENANNT, M. HOFSTÄTTER, P. SCHÖN, M. LITZENBERGER, D. BAUER, AND H. GARN. Live demonstration: Asynchronous time-based image sensor (ATIS) camera with full-custom AE processor. In "Proceedings of 2010 IEEE International Symposium on Circuits and Systems", pp. 1392–1392 (May 2010).

[74] A. N. REFENES, A. ZAPRANIS, AND G. FRANCIS. Stock performance modeling using neural networks: A comparative study with regression models. *Neural Networks* **7**(2), 375 – 388 (1994).

[75] P. REINAGEL AND R. C. REID. Temporal Coding of Visual Information in the Thalamus. *Journal of Neuroscience* **20**(14), 5392–5400 (2000).

[76] A. ROCKE. "The weight transport problem". paulispace (Jun 2017).

[77] B. D. ROUHANI, A. MIRHOSEINI, AND F. KOUSHANFAR. DeLight: Adding Energy Dimension To Deep Neural Networks. In "Proceedings of the 2016 International Symposium on Low Power Electronics and Design",

ISLPED âĂŹ16, p. 112âĂŞ117, New York, NY, USA (2016). Association for Computing Machinery.

[78] B. RUECKAUER, I.-A. LUNGU, Y. HU, AND M. PFEIFFER. Theory and Tools for the Conversion of Analog to Spiking Convolutional Neural Networks. *arXiv e-prints* p. arXiv:1612.04052 (Dec 2016).

[79] V. SAXENA, X. WU, I. SRIVASTAVA, AND K. ZHU. Towards Neuromorphic Learning Machines Using Emerging Memory Devices with Brain-Like Energy Efficiency. *Journal of Low Power Electronics and Applications* **8**(4) (2018).

[80] J. SCHMIDHUBER. Deep learning in neural networks: An overview. *Neural Networks* **61**, 85 – 117 (2015).

[81] A. SHAWON, M. JAMIL-UR RAHMAN, F. MAHMUD, AND M. M. AREFIN ZAMAN. Bangla Handwritten Digit Recognition Using Deep CNN for Large and Unbiased Dataset. In "2018 International Conference on Bangla Speech and Language Processing (ICBSLP)", pp. 1–6 (Sep. 2018).

[82] E. SHELHAMER, J. LONG, AND T. DARRELL. Fully Convolutional Networks for Semantic Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **39**(4), 640–651 (April 2017).

[83] J. SJÖSTRÖM AND W. GERSTNER. Spike-Timing Dependent Plasticity. *Scholarpedia* **5**(2), 1362 (2010). revision #184913.

[84] S. SKORHEIM, P. LONJERS, AND M. BAZHENOV. A Spiking Network Model of Decision Making Employing Rewarded STDP. *PLOS ONE* **9**(3), 1–15 (03 2014).

[85] A. E. SMITH AND A. K. MASON. COST ESTIMATION PREDICTIVE MODELING: REGRESSION VERSUS NEURAL NETWORK. *The Engineering Economist* **42**(2), 137–161 (1997).

[86] E. STROMATIAS, M. SOTO, T. SERRANO-GOTARREDONA, AND B. L.-B. LINARES-BARRANCO. An Event-Driven Classifier for Spiking Neural Networks Fed with Synthetic or Dynamic Vision Sensor Data. *Frontiers in Neuroscience* **11**, 350 (2017).

[87] A. TAVANAEI, M. GHODRATI, S. R. KHERADPISHEH, T. MASQUELIER, AND A. S. MAIDA. Deep Learning in Spiking Neural Networks. *arXiv e-prints* p. arXiv:1804.08150 (Apr 2018).

[88] A. TAVANAEI, Z. KIRBY, AND A. MAIDA. Training Spiking ConvNets by STDP and Gradient Descent. *2018 International Joint Conference on Neural Networks (IJCNN)* pp. 1–8 (07 2018).

[89] A. TAVANAEI AND A. S. MAIDA. Multi-layer unsupervised learning in a spiking convolutional neural network. In "2017 International Joint Conference on Neural Networks (IJCNN)", pp. 2023–2030 (May 2017).

[90] A. TAVANAEI, T. MASQUELIER, AND A. S. MAIDA. Acquisition of visual features through probabilistic spike-timing-dependent plasticity. *2016 International Joint Conference on Neural Networks (IJCNN)* pp. 307–314 (July 2016).

[91] R. VAILA, J. CHIASSON, AND V. SAXENA. Deep Convolutional Spiking Neural Networks for Image Classification. *arXiv e-prints* p. arXiv:1903.12272 (Mar 2019).

[92] R. VAILA, J. CHIASSON, AND V. SAXENA. Feature Extraction Using Spiking Convolutional Neural Networks. In "Proceedings of the International Conference on Neuromorphic Systems", ICONS âĂŹ19, New York, NY, USA (2019). Association for Computing Machinery.

[93] R. VAILA, J. CHIASSON, AND V. SAXENA. Spiking CNNs with PYNN and NEURON. In "NICE Workshop Series", Portland, Oregon, USA (Feb. 2019). Intel.

[94] R. VAILA, J. CHIASSON, AND V. SAXENA. A Deep Unsupervised Feature Learning Spiking Neural Network with Binarized Classification Layers for EMNIST Classification (2020).

[95] R. VAILA, D. LLOYD, AND K. TETZ. Regression with Deep Learning for Sensor Performance Optimization (2020).

[96] R. VANRULLEN. Perception Science in the Age of Deep Neural Networks. *Frontiers in Psychology* **8**, 142 (2017).

[97] X. WU, V. SAXENA, AND K. ZHU. Homogeneous Spiking Neuromorphic System for Real-World Pattern Recognition. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **5**(2), 254–266 (June 2015).

[98] X. WU, V. SAXENA, K. ZHU, AND S. BALAGOPAL. A CMOS Spiking Neuron for Brain-Inspired Neural Networks With Resistive Synapses and In Situ Learning. *IEEE Transactions on Circuits and Systems II: Express Briefs* **62**(11), 1088–1092 (Nov 2015).

[99] D. ZAMBRANO, R. NUSSELDER, H. S. SCHOLTE, AND S. M. BOHTÉ. Sparse Computation in Adaptive Spiking Neural Networks. *Frontiers in Neuroscience* **12**, 987 (2019).

[100] F. ZENKE, B. POOLE, AND S. GANGULI. Continual learning through synaptic intelligence. In D. PRECUP AND Y. W. TEH, editors, "Proceedings of the 34th International Conference on Machine Learning", vol. 70 of "Proceedings of Machine Learning Research", pp. 3987–3995, International Convention Centre, Sydney, Australia (06–11 Aug 2017). PMLR.

[101] J. ZYLBERBERG, J. T. MURPHY, AND M. R. DEWEESE. A Sparse Coding Model with Synaptically Local Plasticity and Spiking Neurons Can Account for the Diverse Shapes of V1 Simple Cell Receptive Fields (2011).

APPENDIX

A.1 Effect of lateral inhibition in pooling layers on subsequent convolution layers

We studied the effects of lateral inhibition [35][34] in convolution and pooling layers in terms classification accuracy and features learned. Not having lateral inhibition in pool 1 layer results in better classification provided overtrain in L4 is prevented.

A.2 With lateral inhibition in pooling layer

Features learnt in the subsequent layers tend to be more complex looking if there is lateral inhibition in this layer and less complex looking if lateral inhibition is not applied. When lateral inhibition is applied, neurons in pooling layers have no more than one spike per image thereby allowing only the most dominant neuron at a location (u, v) and across all the maps to spike. So, out of all the neurons that could have spiked, the synapses of the neuron that spiked first (dominant) correlate the most with the receptive field. Hence the features that are learned in the subsequent convolution layers are more complex looking.

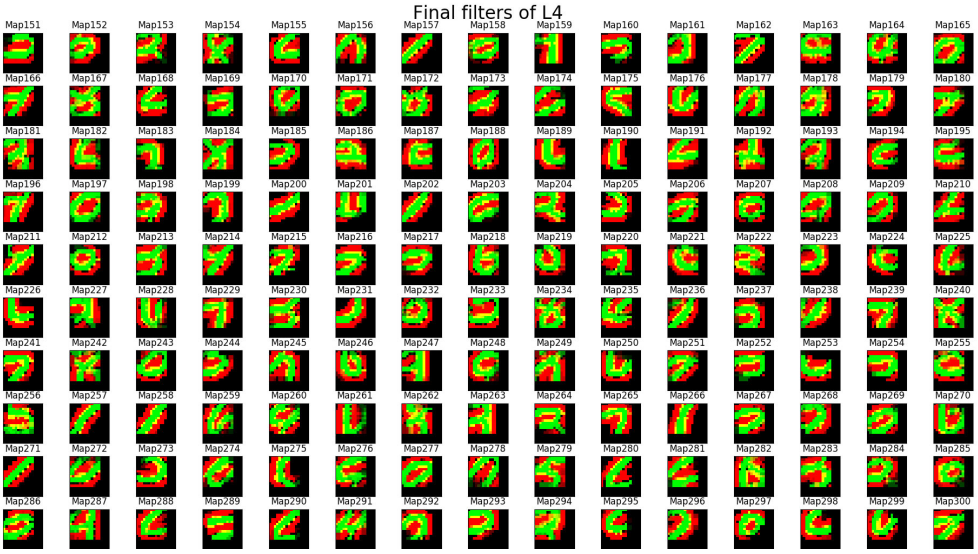

Figure A-1: Weights of first 150 maps of L4 that is trained by in coming spikes with lateral inhibition in L3, STDP competition region in L4 set to $\mathbb{R}^{500\times3\times3}$ and with homeostasis signal applied in L4, notice that the reconstructed features are quite complex and they could well represent a digit or a major section of a digit, note that all neurons of a map in a layer will have shared weights. In this experiment number of maps is L4 was set to 500.

A.3 Scarcity of the spikes

With lateral inhibition in pooling layer (L3), number of spikes available at L4 is reduced drastically. This prevents the build up of the max pooled potentials of the L4 layer thus it gets harder for a classifier to classify these vectors.