Spiking Neural Networks for Image Classification

Osaze Shears
oshears@vt.edu
Virginia Polytechnic Institute and State University
Blacksburg, Virginia

Ahmad Hossein Yazdani
ahmadyazdani@vt.edu
Virginia Polytechnic Institute and State University
Blacksburg, Virginia

ABSTRACT

Artificial neural networks (ANNs) have greatly advanced the fields of video and image processing. These machine learning algorithms have been proven to excel at tasks such as object detection and handwritten digit recognition. Although ANNs have achieved incredibly high levels of accuracy on these tasks when simulated in traditional computing environments, there is an increasing demand for performing these tasks in real time on embedded computer systems with low power consumption.

Researchers believe that spiking neural networks (SNNs) are a suitable alternative because of their energy efficiency and event-driven architectures. However, more research is required on SNNs to determine the best neural models, encoding methods, and training techniques for their use in applications such as image processing applications.

Our group expands on works that have compared the classification performance of ANNs to SNNs. We simulate our SNNs in Python and use different neural models, encoding methods, and training techniques to study how these factors affect the SNN model accuracy. With one epoch of training on the MNIST data set, we observed that the Diehl and Cook model, with input encoded through a Poisson distribution of spikes, achieves the highest accuracy amongst the others. This shows the effectiveness of adjusting the membrane threshold dynamically. Also, it proves that designing the network with only one neuron that fires at each layer aids the learning process.

KEYWORDS

spiking neural networks, MNIST, image classification, neuromorphic hardware

1 INTRODUCTION

Artificial neural networks (ANNs) have greatly advanced the fields of video and image processing. These machine learning algorithms have been proven to excel at tasks such as object detection and handwritten digit recognition. This is shown in Wan et al.'s (2013) approach that performs handwritten digit recognition on the MNIST data set with 0.21% error [13], and Kolesnikov et al.'s (2019) approach that performs image classification on the CIFAR10 dataset with 99.37% accuracy [6].

Although ANNs have achieved incredibly high levels of accuracy on these tasks when simulated in traditional computing environments, there is an increasing demand for performing these tasks in real time on embedded computer systems with low power consumption. For example, an autonomous unmanned aerial vehicle running on a battery power supply may employ an ANN to assist with collision avoidance. For tasks such as this, researchers believe that spiking neural networks (SNNs) are a suitable alternative because of their energy efficiency and event-driven architectures [1].

SNNs are networks of neurons that communicate information through short pulses of data called spikes. A spiking neuron will only output a spike to other neurons once a specific threshold of spikes have been received, thus making them more energy efficient than ANNs. Spiking neuron models are also inherently capable of processing temporal information, which leads researchers to believe that they are more capable of processing spiking event data from devices such as dynamic vision sensors [1]. However, more research is required on SNNs to determine the best neural models, encoding methods, and training techniques for their use in image processing applications.

Our group expands on the research of Deng et al. (2020) in comparing ANNs to SNNs for image classification tasks [3]. Deng et al. (2020) evaluate the performance of SNNs relative to ANNs by (1) measuring their accuracy in classifying the MNIST and CIFAR10 benchmarks, (2) comparing each of the networks' memory cost for storing weights, and (3) comparing cost of performing computations with each network [3]. Our group reimplements the tests performed using different neural models, encoding methods, and training techniques to study how these factors affect the SNN model accuracy.

These experiments are performed using the PyTorch and BindsNET Python packages [5]. PyTorch is a framework that allows researchers to quickly develop ANN models to test, while BindsNET is a recently developed extension of PyTorch that provides a way to efficiently create and train SNN models.

1.1 Optimizing Hardware for Neural Networks

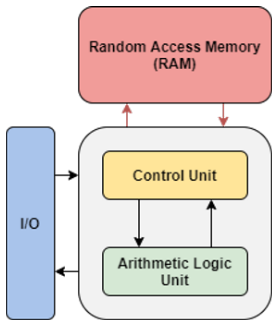


Figure 1: Central Processing Unit (CPU) Computing Architecture

A primary concern when performing machine learning tasks with ANNs is the speed at which inference is performed. On standard hardware that implements a von Neumann style architecture, as shown in **Figure 1**, neurons are evaluated in the **central processing unit, called the CPU**, while the synapse weight information and neuron outputs are stored in the random-access memory, or RAM. The challenge with this approach is that the transfer of data between the CPU and RAM limits the speed at which the network can be evaluated. This is known as the "memory bottleneck" since the latency of transmitting data between the CPU and RAM constrains the evaluation speed of the network. Furthermore, the inherent parallelism of the neural network is not able to be realized since the CPU has a limited number of cores that are able to concurrently update the values of the neurons.

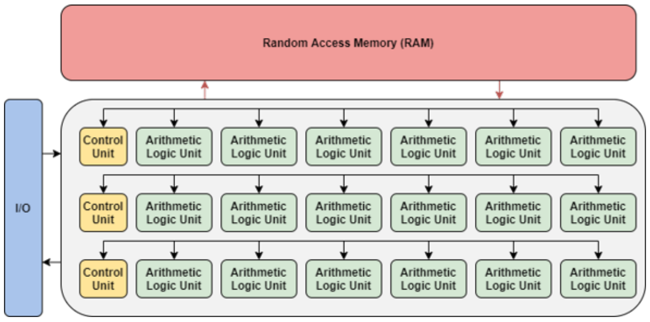


Figure 2: Graphics Processing Unit (GPU) Computing Architecture

Graphics processing units, or GPUs, have been able to improve the speed of neural networks because of their capability to perform large-scale matrix multiplication operations, which consume most of the execution time in ANNs. As shown in **Figure 2**, these devices feature several rows of logic units to perform computations in parallel. However, GPUs suffer from high power consumption because of the number of cores they employ, and like the simpler CPU and RAM model, they are susceptible to the memory bottleneck when their results are stored in the RAM.

To address these issues, researchers have looked towards the development of **neuromorphic hardware** to accelerate the speed of both neural network inference and training. Neuromorphic hardware attempts to more precisely model the architecture of a neural network by (1) featuring computing units that correspond to the neurons in the network, as shown in **Figure 3**, and (2) providing in-memory computing capabilities. By following this approach, the hardware is capable of achieving lower power consumption and faster training and inference times.

One of the key features of neuromorphic hardware is that many designs do not use hardware multiplier units, which are slow components and require a significant amount of area and power. These designs are instead created to emulate the execution of **spiking neural networks (SNNs)**, a more biologically accurate neural network model.

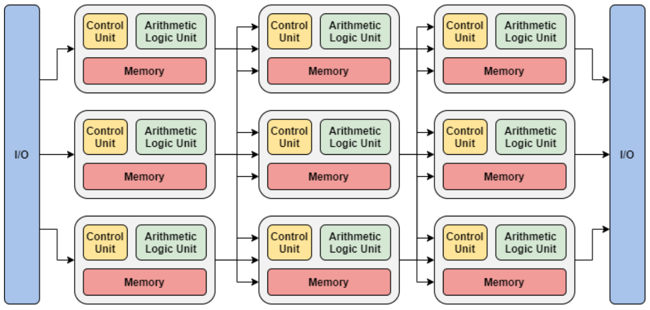


Figure 3: Neuromorphic Computing Architecture

1.2 Spiking Neural Networks

Spiking neural networks are a variant of neural networks that more closely mimic the behavior of the brain. These “third generation” networks receive data in the form of spikes. Each spike corresponds to a specific weight specified by the synapse it travels across. As a neuron receives spikes, its membrane potential increments towards a threshold value. When the membrane potential exceeds this threshold value, the neuron emits a spike which gets transmitted to all other neurons connected to it. After firing, the neuron waits for a period of time known as the “refractory period” before its membrane potential can start accumulating again. This operation continues for a distinct span of time[8].

Table 1 shows an overview of the differences between ANNs and SNNs. Traditional ANNs perform inference instantaneously, compared to SNNs that need to operate over a specified duration of time before an input’s classification can be determined. Second, traditional ANNs use raw numerical values as inputs for the neurons, compared to SNNs that use spikes that occur over a specified duration of time as inputs to the network. Finally, the output value for a traditional ANN neuron is determined by passing the sum of the input values multiplied by their respective synapse weights through an activation function such as ReLU or sigmoid. An SNN neuron, on the other hand, first adds the weights of the synapses that carried spikes at the current timestep to its membrane potential, then compares the value of its membrane potential to a threshold value to determine if it should emit a spike.

SNNs are believed to have more potential than traditional ANNs in several aspects. The energy efficiency of an SNN can outperform that of an ANN depending on the technique used to encode the input data into spikes. The reason for this is because an input neuron will not necessarily produce a spike at each timestep, reducing the amount of power consumed.

SNNs also do not inherently feature any large-scale matrix multiplication. Each neuron is only required to add the weight values of the synapses that had spikes at each timestep to its current membrane potential, and to check if the potential exceeds its threshold value. Thus, SNN neurons can be realized with only adders and comparators, which combined have a lower area and power consumption than the multipliers and adders required by a traditional ANN hardware implementations[1]. SNNs may also be more capable of learning patterns in time series and real time problems due to their temporal properties, and their biologically accurate

Table 1: Differences Between ANNs and SNNs

Network Type	Execution Time	Neuron Inputs/Outputs	Output Mechanism
ANN	Instantaneous	Raw Numerical Value	Activation Function (e.g., ReLU, Sigmoid)
SNN	Duration of Time	Binary Spike Value	Threshold Value

representation may allow them to better accelerate neuroscience research.

SNNs have three primary components needed for simulation: an encoding scheme, a neural model and a learning technique. The following section will detail how these components work.

1.3 Encoding Schemes

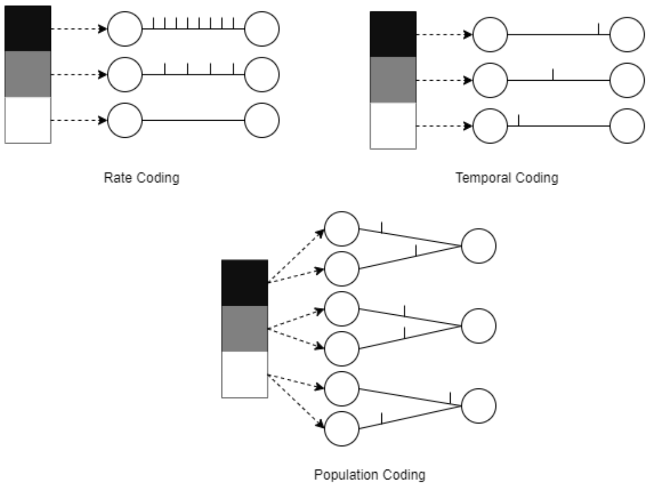


Figure 4: Spiking Neural Network Encoding Schemes

Many schemes exist for encoding input data into spikes that can be fed into an SNN. **Rate coding** is one of the most popular encoding schemes which maps an input's intensity to its corresponding input neuron's firing rate. The higher the intensity of the input, the more frequently the input neuron will fire a spike. Rate coding is often supplemented with Poisson and Bernoulli distributions to add a stochastic element to the spike firing times.

Temporal coding takes a more sparse approach at encoding the input data. Each input neuron only fires once, and higher intensity inputs fire earlier than lower intensity inputs. Latency coding is a type of temporal coding scheme where the firing time of an input neuron is proportional to its intensity. Rank order coding is a temporal coding scheme that orders the firing times discretely from neuron with the highest intensity to the lowest[11].

Population coding is a yet another approach where each input corresponds to several neurons, and the input is encoded by adjusting the times at which each of the input neurons fire. The usage of Gaussian receptive fields with each neuron corresponding to one of the Gaussian distributions is a common implementation of population coding[9].

Each encoding scheme has its trade-offs. A rate coding scheme is very robust to noise that may be transmitted with the input spikes; however, neurons firing at a high rate cause the circuit to consume more power. Additionally, the SNN needs to be simulated for an adequate amount of time to accurately measure the input neuron rates. A temporal coding scheme uses less energy because spikes are only transmitted once, but it is very susceptible to noise that gets transmitted in place of spikes. Population coding is a more balanced approach compared to rate and temporal coding, but it requires more neurons to represent the input data.

1.4 Neural Models

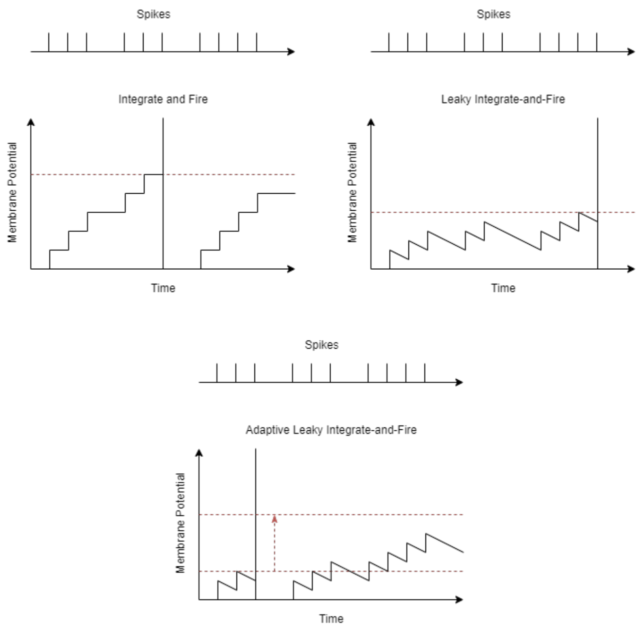


Figure 5: The Integrate-and-Fire, Leaky Integrate-and-Fire and Adaptive LIF Neural Models

Similar to changing the activation function for a neuron in a standard ANN, the behavior of each neuron can be adjusted in an SNN by using different neural models. The **integrate-and-fire, or IF**, neuron is a simple model where the membrane potential is accumulated normally until a threshold value is reached before emitting a spike.

The **leaky integrate-and-fire, LIF, model** has also been derived as a more biologically accurate model. In an LIF neuron, the membrane potential decrements towards a resting value at each timestep. The longer the neuron goes without receiving an input spike, the closer the membrane potential will get to its resting state[2].

The **adaptive LIF model** is similar to the LIF model, except when the neuron emits a spike, the neuron's threshold value increases to prevent it from quickly firing again. The threshold value will continue to increase each time the neuron fires, and gradually decays towards a resting value when the neuron is not firing[4].

Lastly, the **spike response model 0, or SRM0**, is a neuron model that also features a leaking membrane voltage. In addition, the spikes produced by this model are generated stochastically, where the probability of the neuron spiking increases as a function of the membrane potential[12].

1.5 Spike-Timing Dependent Plasticity

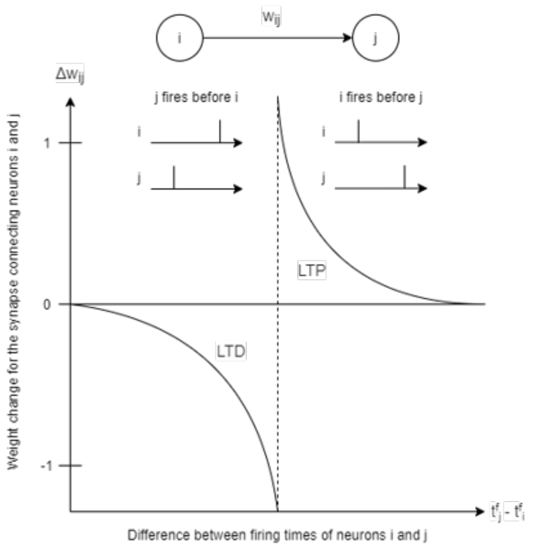


Figure 6: Spike-Timing Dependent Plasticity

One of the most commonly used learning techniques for SNNs is **spike-timing dependent plasticity, or STDP**. This unsupervised learning rule operates on the behavior: "neurons that fire together, wire together". As shown in **Figure 6**, if a pre-synaptic neuron (i) fires just before a post-synaptic neuron (j), the weight between those two neurons is increased. This behavior is called long-term potentiation (LTP) and the intuition is that the connection between the neurons should be strengthened if the pre-synaptic neuron causes the post-synaptic neuron to spike. Alternatively, if a pre-synaptic neuron (i) fires just after a post-synaptic neuron (j), the weight between those two neurons is decreased. This behavior is called long-term depression (LTD) and the intuition is that the connection should be weakened if the pre-synaptic neuron has no relation to the post-synaptic neuron[10].

1.6 Implementation Challenges

Though they have extraordinary potential, as outlined by their advantages, there are several challenges that are limiting the application of SNNs in machine learning problems.

SNNs primarily suffer because training them is not intuitive. Spike events are non-differentiable, which makes optimizing the cost function of the network challenging. SNNs have not been able to reach the same accuracy level on similar classification tasks performed by ANNs because efficient ways to train SNNs are still under development. Even the process of comparing the performance of SNNs and ANNs needs to be studied more since SNNs and ANNs perform differently depending on the format of the dataset provided. The programming frameworks used to model SNNs are also still in their infancy and need to be more comprehensive to support efficient SNN simulation[7].

Lastly, there are several parameters of the encoding scheme, neural model, and learning technique that can be adjusted when modeling an SNN. More research needs to be conducted to identify the trade-offs between each of these variations for machine learning tasks. The research in this project aims to understand the trade-offs of these variations for a simple image classification task which has been shown in several previous works.

2 RELATED WORK

2.1 Diehl and Cook (2015)

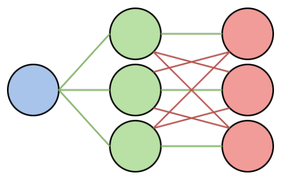


Figure 7: Diehl and Cook's Spiking Neural Network Architecture

In this work, Peter U. Diehl and Matthew Cook, from the University of Zurich, proposed an unsupervised algorithm for training a spiking neural network on the MNIST dataset. As depicted in **Figure 7**, this network consisted of three layers: a 784 neuron input layer, an excitatory layer of adaptive LIF neurons, and an inhibitory layer of LIF neurons[4]. Each excitatory layer neuron was connected to only one neuron in the inhibitory layer. On the other hand, the output of the neurons in the inhibitory layer were connected to all the neurons of the excitatory layers except for the one it received input from. Other excitatory neurons were inhibited once an excitatory neuron emitted a spike. An inhibitory neuron inhibited the internal voltage of a neuron once it received a spike. This was helpful since an active neuron could prevent the voltage of other neurons from increasing. The layer of adaptive LIF neurons further operated in a way that only one excitatory neuron fired per input, which is also referred to as the winner-take-all (WTA) property.

The group used four different versions of STDP when training the network training, and used Poisson rate coding to encode the input data. With their proposed version of STDP, the best accuracy they achieved was around 95% in a 784x6400x6400 style network.

2.2 Deng et al. (2020)

In this work, Deng et al. (2020) compared the classification performance of various ANN and SNN models using the MNIST, CIFAR10, NMNIST, and DVS-CIFAR10 datasets. Each of the networks was trained using a type of the supervised backpropagation algorithm, and the SNNs used LIF neuron models. The group observed that one of the SNNs was able to achieve 99.22% accuracy when tested against the MNIST dataset, and 74.23% accuracy when tested against the CIFAR10 dataset[3].

The authors also conducted an analysis on the memory usage and computation resources required for each network. They found that one of their SNN models was ideal for MNIST classification because of its lower compute cost and reasonable accuracy. Additionally, the authors noted that studying the performance of various encoding schemes, aside from rate coding, is important to better understand the capabilities of SNNs.

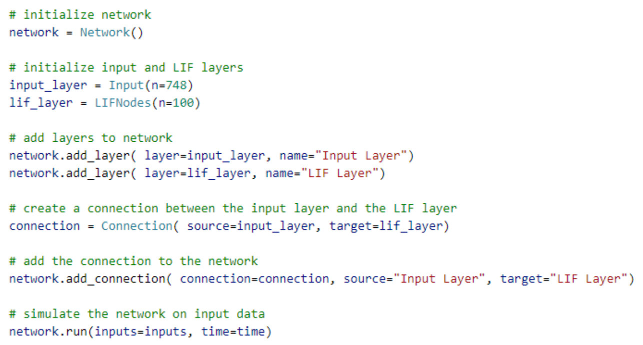
3 METHOD

Our project expands on the research presented by Deng et al. (2020). We compare the classification accuracy of a two-layer ANN to a two-layer SNN using the MNIST dataset. Both networks were created in Python using the PyTorch framework as a foundation.

3.1 ANN Experiment

The ANN tested in this project was developed in PyTorch, a deep learning framework widely used by researchers, because it provides facilities to work with GPUs. The network implemented features a two-layer 784x100 neuron architecture. The inputs of the first layer are summed and passed through the ReLU activation function, while the inputs of the second layer are passed through the softmax function. The network is trained using stochastic gradient descent with a learning rate of 0.01. Mini-batch training is used to reduce the training time with a batch size of 64.

3.2 SNN Experiment



```
# initialize network
network = Network()

# initialize input and LIF layers
input_layer = Input(n=748)
lif_layer = LIFNodes(n=100)

# add layers to network
network.add_layer( layer=input_layer, name="Input Layer")
network.add_layer( layer=lif_layer, name="LIF Layer")

# create a connection between the input layer and the LIF layer
connection = Connection( source=input_layer, target=lif_layer)

# add the connection to the network
network.add_connection( connection=connection, source="Input Layer", target="LIF Layer")

# simulate the network on input data
network.run(inputs=inputs, time=time)
```

Figure 8: A simple two layer network created using BindsNET

The SNNs simulated in this project were created in the BindsNET framework which uses PyTorch as the backend library. An example of how to construct a network in BindsNET is shown in **Figure 8**. The SNNs in this project feature a 784x100 neuron architecture with an inhibitory recurrent connection at the output layer, a similar architecture to that proposed by Diehl and Cook (2015)[4].

Three encoding schemes were used to convert the input images into spike trains. These images were constrained to have a maximum input intensity of 128 and were encoded over a period of 100 milliseconds. The encoding schemes used include:

Table 2: Spiking Neural Network Hyperparameters

Models	Hyperparameter	Value
All	Threshold Voltage	-52
All	Post-Spike Reset Voltage	-60
All	Refractory Period	5
LIF, SRM0, D&C	Membrane Potential Decay Constant	100
LIF, SRM0, D&C	Resting Voltage	-65
D&C	Threshold Voltage Increase (Adaptive)	0.05
D&C	Threshold Voltage Decay Constant	$10^{-}7$

Table 3: Spiking Neural Network Learning Hyperparameters

Hyperparameter	Value
LTP Learning Rate	$10^{-}2$
LTD Learning Rate	$10^{-}4$
Minimum Weight Value	0
Maximum Weight Value	1

- Poisson Rate Coding
- Bernoulli Rate Coding
- Rank Order (Temporal) Coding

Four variations of neuron models were analyzed against the input data over the period of 100 milliseconds. The hyperparameters used for these neurons are noted in **Table 2**. These neuron models include:

- Integrate-and-Fire Model (IF)
- Leaky Integrate-and-Fire Model (LIF)
- Spike Response Model 0 (SRM0)
- Diehl and Cook's Adaptive LIF Model (D&C)

Three unsupervised learning rules were used to train the network. The hyperparameters used for these learning rules are noted in **Table 3**. The learning rules include:

- STDP
- Weighted STDP
- Hebbian Learning

Similar to the ANN, the SNN also used mini-batch training with a batch size of 64, and was trained for only one epoch to reduce training time. To determine the predicted output of the SNN, each excitatory neuron was assigned a label based on the amount of times it produced a spike for the presented input data. During training, the labels were adjusted based on the spike frequency of the excitatory neurons. Once training was completed, the labels were fixed and evaluated against the test data.

4 RESULTS AND EVALUATION

4.1 Results

The ANN was first trained using the MNIST dataset and tested to have a basis for comparison to the SNNs. The ANN was able to achieve a classification accuracy of 88% using our approach. **Table 4**, shows the results of training several SNNs using the neuron model and learning rule variations with a Poisson rate coding scheme applied to the input data. The best performing network was

Table 4: Classification Accuracy for Poisson Encoded Input

Neuron Model	Learning Rule	Accuracy
IF	PostPre (STDP)	0.10
IF	WeightDependentPostPre	0.09
IF	Hebbian	0.10
LIF	PostPre (STDP)	0.10
LIF	WeightDependentPostPre	0.13
LIF	Hebbian	0.10
SRM0	PostPre (STDP)	0.10
SRM0	WeightDependentPostPre	0.10
SRM0	Hebbian	0.10
DiehlAndCook	PostPre (STDP)	0.79
DiehlAndCook	WeightDependentPostPre	0.79
DiehlAndCook	Hebbian	0.80

the Diehl and Cook neuron model with an accuracy of 80% when the Hebbian learning rule was used. **Table 5** shows that when a Bernoulli rate coding scheme is used, the Diehl and Cook model with the PostPre (STDP) learning rule performs the best with an accuracy of 37%. Lastly, **Table 6** shows that when a rank order (temporal) coding scheme is used, the Diehl and Cook model with the PostPre learning rule performs the best with an accuracy of 14%.

4.2 Discussion

The results obtained from this project were not what we anticipated. All of the SNNs that used a neural model without an adaptive threshold value performed poorly compared to the Diehl and Cook model. Since we used the same hyperparameters that Diehl and Cook used to configure their network, we believe that one reason for this poor performance is because the hyperparameters were not optimized for the other neuron models tested. We believe that if the parameters are adjusted based on the neuron model used, then we would have observed better classification accuracy. This is especially true for the rank order encoding networks since each input neuron only emits one spike.

We believe another reason for the poor accuracy of the models is because of the duration of time we simulated the SNNs. In this project, we used a simulation time of 100 milliseconds for all networks; however, some groups report that running the SNN simulation for a longer duration of time allows it to more accurately predict the output, specifically when a rate coding scheme is used. Increasing the number of excitatory neurons used may also have a positive influence on the accuracy, as shown in Diehl and Cook's research [4]. Finally, reducing the batch size can increase the accuracy by allowing each of the training samples to more substantially influence the weight changes.

In conclusion, we found that SNNs have several advantages as outlined in our introduction, however, learning how to effectively tune their attributes so that they may accurately perform image classification tasks is challenging and requires further research.

Table 5: Classification Accuracy for Bernoulli Encoded Input

Neuron Model	Learning Rule	Accuracy
IF	PostPre (STDP)	0.10
IF	WeightDependentPostPre	0.10
IF	Hebbian	0.10
LIF	PostPre (STDP)	0.10
LIF	WeightDependentPostPre	0.10
LIF	Hebbian	0.10
SRM0	PostPre (STDP)	0.10
SRM0	WeightDependentPostPre	0.09
SRM0	Hebbian	0.10
DiehlAndCook	PostPre (STDP)	0.37
DiehlAndCook	WeightDependentPostPre	0.34
DiehlAndCook	Hebbian	0.35

Table 6: Classification Accuracy for Rank Order Encoded Input

Neuron Model	Learning Rule	Accuracy
IF	PostPre (STDP)	0.10
IF	WeightDependentPostPre	0.10
IF	Hebbian	0.10
LIF	PostPre (STDP)	0.10
LIF	WeightDependentPostPre	0.10
LIF	Hebbian	0.10
SRM0	PostPre (STDP)	0.10
SRM0	WeightDependentPostPre	0.10
SRM0	Hebbian	0.10
DiehlAndCook	PostPre (STDP)	0.14
DiehlAndCook	WeightDependentPostPre	0.11
DiehlAndCook	Hebbian	0.13

REFERENCES

[1] Maxence Bouvier, Alexandre Valentian, Thomas Mesquida, Francois Rummens, Marina Reyboz, Elisa Vianello, and Edith Beigne. 2019. Spiking neural networks hardware implementations and challenges: A survey. *ACM Journal on Emerging Technologies in Computing Systems* 15, 2 (apr 2019). https://doi.org/10.1145/3304103

[2] Anthony N Burkitt. 2006. A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input. *Biological cybernetics* 95, 1 (2006), 1–19.

[3] Lei Deng, Yujie Wu, Xing Hu, Ling Liang, Yufei Ding, Guoqi Li, Guangshe Zhao, Peng Li, and Yuan Xie. 2020. Rethinking the performance comparison between SNNS and ANNS. *Neural Networks* 121 (jan 2020), 294–307. https://doi.org/10.1016/j.neunet.2019.09.005

[4] Peter U. Diehl and Matthew Cook. 2015. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience* 9, AUGUST (aug 2015), 99. https://doi.org/10.3389/fncom.2015.00099

[5] Hananel Hazan, Daniel J. Saunders, Hassaan Khan, Devdhar Patel, Darpan T. Sanghavi, Hava T. Siegelmann, and Robert Kozma. 2018. BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python. *Frontiers in Neuroinformatics* 12 (dec 2018), 89. https://doi.org/10.3389/fninf.2018.00089

[6] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. [n.d.]. *Big Transfer (BiT): General Visual Representation Learning.* Technical Report. arXiv:1912.11370v3

[7] Guoqi Li, Lei Deng, Yansong Chua, Peng Li, Emre O Neftci, and Haizhou Li. 2020. Spiking Neural Network Learning, Benchmarking, Programming and Executing. *Frontiers in Neuroscience* 14 (2020).

[8] Wolfgang Maass. 1997. *Networks of Spiking Neurons: The Third Generation of Neural Network Models.* Technical Report 9. 1659–1671 pages.

[9] Boudjelal Meftah, Olivier Lezoray, and Abdelkader Benyettou. 2010. Segmentation and edge detection based on spiking neural network model. *Neural Processing Letters* 32, 2 (2010), 131–146.

[10] J. Sjöström and W. Gerstner. 2010. Spike-timing dependent plasticity. *Scholarpedia* 5, 2 (2010), 1362. https://doi.org/10.4249/scholarpedia.1362 revision #184913.

[11] Simon Thorpe and Jacques Gautrais. 1998. Rank order coding. In *Computational neuroscience*. Springer, 113–118.

[12] Eleni Vasilaki, Nicolas Frémaux, Robert Urbanczik, Walter Senn, and Wulfram Gerstner. 2009. Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail. *PLoS Comput Biol* 5, 12 (2009), e1000586.

[13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Lecun, and Rob Fergus. 2013. *Regularization of Neural Networks using DropConnect*. Technical Report.