Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 213 (2022) 536–545

Procedia Computer Science

www.elsevier.com/locate/procedia

2022 Annual International Conference on Brain-Inspired Cognitive Architectures for Artificial Intelligence: The 13th Annual Meeting of the BICA Society

Cyber-Neuro RT: Real-time Neuromorphic Cybersecurity

Wyler Zahm[a], Tyler Stern[a], Malyaban Bal[b], Abhronil Sengupta[b], Aswin Jose[a], Suhas Chelian[a], Srini Vasan[a,*]

[a]*Quantum Ventura, 1 South Market Street, Suite 1715, San Jose, CA 95113, USA*
[b]*Penn State University, University Park, PA, 16802 USA*

Abstract

High throughput environments, such as those found in high performance computing (HPC) clusters, run at substantially greater scales than standard business IT domains. As a result, cybersecurity tools built for businesses utilizing standard machine learning frameworks are unable to handle the increased amount of traffic and connections in high throughput environments. Neural networks, in combination with edge-based next-generation embedded technologies such as neuromorphic processors, offer a solution to cybersecurity challenges in high throughput environments. Deep learning (DL), deep learning to spiking neural network (DL-to-SNN) conversions, and design exploration inside SNNs were among the experiments employed to explore this area. Neuromorphic implementations of deep learning networks often provide the same accuracy as full precision models while saving substantial power and cost. We explore this statement in the cybersecurity domain. Results are promising but will be further investigated.

Keywords: BrainChip Akida™; cybersecurity; deep learning; Intel Loihi; network intrusion

1. Introduction

Compared to traditional enterprise IT domains, high throughput environments such as those in high performance computing (HPC) clusters operate at much larger scales. Thus, cybersecurity tools developed for the enterprises

* Corresponding author. Tel.: +1 424-227-1417
E-mail address: srini@quantumventura.com

using traditional machine learning frameworks are not able to support the greater number of connections and traffic in high throughput environments. Neural networks combined with edge-based hardware-resident next-generation technologies such as neuromorphic processors can monitor and even predict events in high throughput environments and hence provide an up-and-coming solution to cybersecurity in HPC.

We sought to evaluate the viability of a real-time HPC-scale neuromorphic cybersecurity system called Cyber-Neuro RT. We began by developing a full precision deep learning (DL) network for 450,000 Zeek log entries with a mixture of normal and malicious data, including eight different types of attacks. Deep learning to spiking neural network (DL-to-SNN) conversions were conducted for two vendors, Intel Loihi and BrainChip Akida™ to explore their platforms. Design space explorations to increase SNN inference speed were also conducted. Neuromorphic implementations of deep learning networks frequently achieve the same level of accuracy as full precision models while saving substantial power and cost. We investigate this statement in the context of cybersecurity. The results are promising, but more research is needed.

2. Related work

Our survey of related work began with benchmark testing between CPU/GPU-based artificial neural networks (ANNs) and spiking neural networks (SNNs) in simulation and implemented on neuromorphic hardware. For example, Sandia National Labs' Neuromorphic Data Microscope (NDM) is an SNN for network analytics implemented on FPGA. It is demonstrated to detect cyber-attacks, and its performance is compared against RE2, a then-state-of-the-art CPU-based algorithm for cyber-attack detection. In validated results (i.e., from the set in which RE2 and the NDM results matched), the NDM showed a ~2000x improvement in time/energy performance [1]. The performance advantages of neuromorphic software and hardware design were further validated in a technical report demonstrating Mosaic, which is a technique for increasing the computational yield of a neuromorphic network by reusing the neurons for different stages of an algorithm, which this group incorporates into a hybrid system with high-density analog non-volatile memory. In testing, the neuromorphic RNN outperforms a standard CNN run on CPU, using 13-38x less energy. They utilize the Whetstone training system and the CrossSIM hardware simulation environment to test this hybrid system using an RNN with added noise, on the Fashion MNIST dataset [2]. The Whetstone training method produces spiking CNNs and deep multi-layer perceptrons compatible with neuromorphic hardware. In another technical report from Sandia National Labs, it is deployed using the SpiNNaker architecture on a 48-node neuromorphic system-on-a-chip which allows for highly parallel implementations. Whetstone uses faster binary representations of weights rather than slower temporal coding representations, which are typical for translating larger networks [3].

We more deeply investigated the specific results of SNN models designed to detect network traffic anomalies and cyber-attacks. The work of Alam et al. [4] is of particular interest in this regard. In their 2019 publication, they demonstrate a 92.91% effective intrusion detection algorithm which uses an autoencoder translated onto memristor-based neuromorphic hardware. Another autoencoder is used for real-time unsupervised learning of novel attack vectors, and is able to detect anomalous packets, from within the set of detected malicious packets, upon their first presentation with 98.89% accuracy, demonstrating an ability to quickly learn zero-day attacks. As anomalies are presented more times, they are detected less often as they become more familiar. Additionally, Ref. [5] demonstrates a memristor-based neuromorphic system which uses the Extreme Learning Machine (ELM) algorithm, based on Adaptive Resonance Theory (ART), to detect cyber-attacks. This is an unsupervised neural network designed for few-shot fast learning of novel attacks. It shows 99.97% accuracy for a network traffic dataset which includes zero-day attacks in MATLAB SPICE simulations of the memristor-based hardware. Components of ART such as Instar learning were simulated in the spiking domain by Chelian and Srinivasa [6] under the DARPA SyNAPSE program.

We also investigated adversarial attempts to thwart classification by machine learning models, as demonstrated in Kim et al.'s work on adversarial images [7]. This is relevant for cybersecurity because malicious actors may try to disguise their attacks as normal traffic. Adversarial images are images which have been altered in a way to cause misclassification by machine learning models, often in ways imperceptible to human vision. This publication shows a method for sparse reconstruction of images using neuromorphic hardware (including the Intel Loihi) and software simulation, which achieves low reconstruction error and high classification accuracy on these adversarial images [7].

3. Methods

3.1. Data selection and pre-processing

To simulate a high throughput environment, data was adapted from the UNSW data set, which includes heterogeneous data sources collected from telemetry datasets of IoT and IIoT sensors, Windows 7 and 10, Ubuntu 14 and 18 LTS, and Network traffic datasets. The datasets were gathered in a parallel manner to collect several normal and cyber-attack events from IoT networks. The data was generated at the UNSW IoT lab to connect many virtual machines, physical systems, hacking platforms, cloud and fog platforms, IoT and IIoT sensors to mimic the complexity and scalability of enterprise, industrial IoT and Industry 4.0 networks [8]. Our training data of Zeek log files consisted of 450,000 entries with normal data and eight attack types. Sixty-six percent of data was normal. If a machine learning model declared all traffic as normal, it would achieve 66% accuracy, thus creating a chance lower bound on accuracy. Details of attack types and class distributions and how each attack type was performed are shown in Table 1.

Table 1: Distribution of normal and eight attack types in our dataset of 450,000 entries.

Class	Percentage	Class (continued)	Percentage (continued)	Class (continued)	Percentage (continued)
Normal	66.5%	DDOS	4.5%	Backdoor	4.0%
DOS	4.5%	Password	4.0%	Ransomware	4.0%
Injection	4.5%	XSS	4.0%	Scanning	4.0%

Data cleaning was performed by replacing null or hyphen values with zeros, converting IP addresses from IPv4 numbers into binary form (in network byte order), etc. The dataset has 45 original features but we removed labels, type and timestamp leaving 42 features. We then performed features selection, feature encoding, etc. shown in Figure 1.

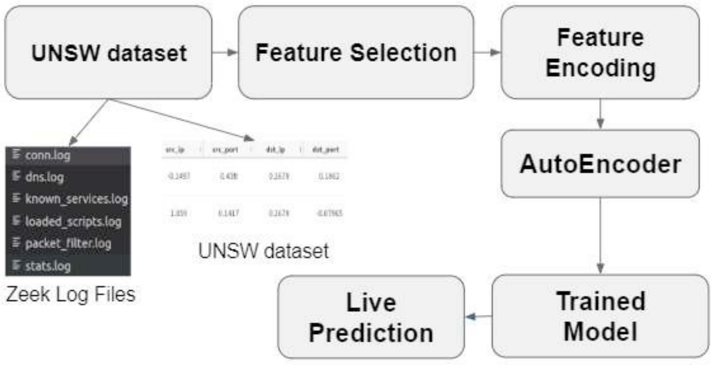

Figure 1: After data cleaning (not shown), feature selection and encoding were performed to feed the data into deep learning networks.

For feature selection we used a random forest classifier to find the most important features. The classifier selected a subset of the original features. (For new datasets, the same features could be used or a random forest classifier or other methods can be used to select new features.). In Table 2, we list the features used.

Table 2: Features selected by a random forest classifier from our dataset.

Feature Name	Description	Feature Name (continued)	Description (continued)	Feature Name (continued)	Description (continued)
src_ip_bytes	Source IP bytes	src_bytes	Source bytes	dst_ip	Destination IP

Feature Name	Description	Feature Name (continued)	Description (continued)	Feature Name (continued)	Description (continued)
dst_port	Destination port	dst_bytes	Destination bytes	proto	Protocol
conn_state	Connection state	dns_rejected	Whether DNS rejected	src_pkts	Source packets
dst_ip_bytes	Destination IP bytes	src_ip	Source IP address	dst_pkts	Destination packets
duration	Duration	src_port	Source port	service	Service code

Additionally, label encoding was used on categorical features. All features have been scaled to be between 0 and 1. In addition, an autoencoder was used to select the most salient underlying features of the dataset to reduce the input dimensionality for the core model. Output from the trained encoder, but not decoder, layers are then fed into the core model for training and testing. The full dataset was split randomly into 85% for training (382,000) and 15% (67,500) for testing.

3.2. Full precision deep learning network

Data was then fit into a 10-layer deep neural network (9 network layers plus the input layer). Its inputs were the outputs from the autoencoder's encoder (but not decoder) and its outputs were the 9 possible classes plus 1 class for future proofing. We also implemented an initial hyperparameter tuning step using Keras Tuner. We used the hyperband algorithm to pick the best hyperparameter values to initiate the training. Though there are several parameters available for tuning (optimizer, epoch, activation function, batch size, number of neurons, learning rate etc.), we selected the dense layer units and the learning rate for tuning purposes. In Table 3, we show several details of the neural network.

Table 3: Parameters for the full precision deep learning network.

Parameter	Value	Parameter (continued)	Value (continued)
Activation Function in Hidden layers	ReLU	Loss	Categorical cross-entropy
Activation Function in Output layer	SoftMax	Dropout rate	0.30
Optimizer	Adam	Batch size	1000

3.3. Intel Loihi and BrainChip Akida

The 10-layer deep learning network was converted into formats compatible with Intel and BrainChip. In both cases, precision of the weights was varied to see how it affected model size and accuracy. Timing results were also produced. For Intel, we accessed Loihi (Nahuku32) chips via the cloud and NxSDK 1.0; for BrainChip, we simulated the chips on CPU via their hardware abstraction layer. Timing results are not comparable because CPU or GPU simulations are much faster than running on neuromorphic hardware.

3.3.1. Intel

We used the SNN-TB toolbox, version 0.6.0 [9], which automatically converts a full precision artificial neural network (ANN) to a spiking neural network (SNN) implementation through a two-step process of parsing and conversion as shown in the Figure 2. SNN-TB is an open-source toolbox developed by the Institute of Neuroinformatics at the University of Zurich and ETH Zurich, as a hardware agnostic ANN-to-SNN conversion tool. It produces SNN models compatible with the Intel Loihi chip, and accepts models built using Tensorflow/Keras or PyTorch. The toolbox is built with Python and requires a user-generated configuration file to guide the conversion process. The configuration file specifies which parameters and algorithms are used for conversion, with options guiding the toolbox to the model and testing data, whether and how much to normalize the weights of the model, and which algorithms to use for conversion of softmax and max pooling layers.

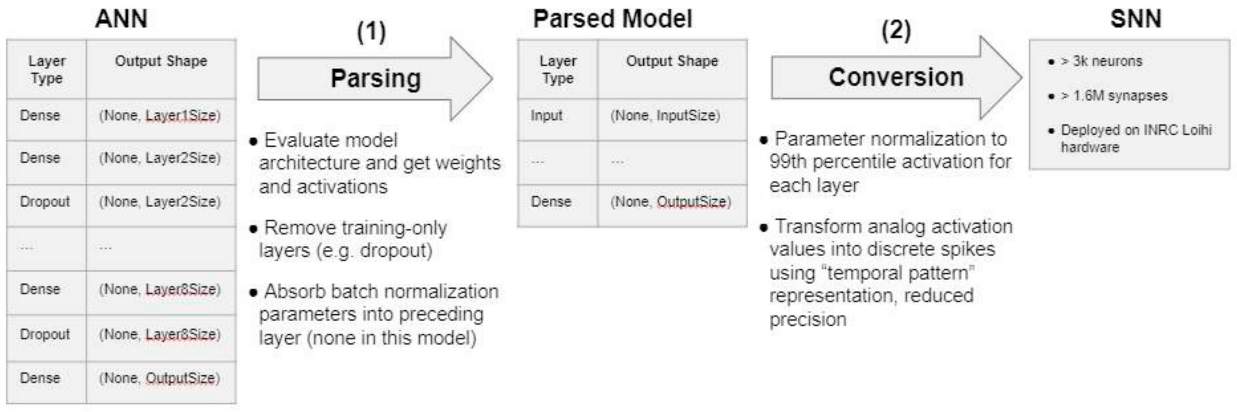
ANN
Layer Type | Output Shape
Dense | (None, Layer1Size)
Dense | (None, Layer2Size)
Dropout | (None, Layer2Size)
... | ...
Dense | (None, Layer8Size)
Dropout | (None, Layer8Size)
Dense | (None, OutputSize)
(1)
Parsing
• Evaluate model architecture and get weights and activations
• Remove training-only layers (e.g. dropout)
• Absorb batch normalization parameters into preceding layer (none in this model)
Parsed Model
Layer Type | Output Shape
Input | (None, InputSize)
... | ...
Dense | (None, OutputSize)
(2)
Conversion
• Parameter normalization to 99th percentile activation for each layer
• Transform analog activation values into discrete spikes using "temporal pattern" representation, reduced precision
SNN
• > 3k neurons
• > 1.6M synapses
• Deployed on INRC Loihi hardware

Figure 2: Intel ANN to SNN conversion process.

In the parsing step (1) of Figure 2, the original ANN is converted to a conversion-compatible ANN and then evaluated to ensure that the parsing does not reduce overall model accuracy. Some layers are removed, such as dropout layers, which are only used in training and therefore unnecessary for deployment. Some layers, such as ReLU activation layers, are absorbed into their corresponding dense layers, to ensure compatibility with the conversion process. The toolbox then completes the evaluation of the parsed model on the testing data. In this conversion process, the parsed model had identical accuracy to the original model using the same testing data, validating that the model is effectively represented by its parsed counterpart.

In conversion (2) of Figure 2, the layers are mapped onto neurons and the activation values are transformed into discrete spikes. To select an algorithm to transform the analog activation values into digital spikes, there are three primary options available within the toolbox: temporal mean rate, temporal pattern, and time to first spike. Using the temporal mean rate algorithm, the number of spikes a neuron receives during a set duration is simply counted. With the temporal pattern algorithm, the spike timings are taken into account and a binary spike train is generated to represent the activation values. When using time to first spike as the representation, the length of time until the neuron first fires is used as a proxy for the activation, which has the potential to decrease inference time and therefore system latency significantly. We compared accuracy for converted models using each of the spiking algorithms, both with and without normalization, and in all cases besides the one in which the temporal pattern spiking algorithm was used with threshold normalization, the prediction accuracy was below 67% and all test samples were predicted to be of the normal class, which dominated the training and testing sets. Initial results with the temporal pattern spike code algorithm showed 81.4% accuracy and therefore we decided to limit the search space for additional hyperparameter tuning to options including this configuration. Hyperparameter tuning also included the selection of number of timesteps allowed for processing. We tested various numbers of timesteps in increments of 20 up to 100, then in increments of 100 up to 1000, and found that increases in accuracy plateaued after 100 timesteps. Therefore we conducted all following runs with 100 timesteps.

The configuration file for the conversion also specifies the quantization of the model to a specific bit depth. For simulation purposes, available bit depths for the conversion are 4, 8, 16, and 32. However, the Loihi 1 hardware only supports bit depths 4 and 8.

The neuron thresholds are normalized based on the training data, which is necessary due to the relatively unbalanced class distribution this model expects to see. The normalization process used the entire training set as input in this case, although it can be conducted on a smaller subset of the data. The digital nature of the Loihi hardware represents the values of analog activation functions with varying accuracy depending on where the values are within the range of potential activations. This means that threshold normalization is required in many cases to ensure that the activation values fit into the ideal range of firing rates for Loihi neurons.

3.3.2. BrainChip

To convert the standard ANN to an Akida compatible SNN, the BrainChip MetaTF CNN2SNN toolkit was used in three main steps. First, one must ensure the ANN model definition is compatible with the Akida platform, (1) in Figure 3. For this step BrainChip provides the function "check_model_compatibility" which returns a true or false result. If the model is not compatible, one must redefine the model to respect the Akida platform constraints. Akida supports the most commonly used deep learning layers found in many popular models, but not all. (Subsequent hardware iterations may support additional features.) Akida also requires strict layer ordering (i.e., Dropout must come before AvgPooling) and certain layers must abide by certain parameter limitations. For our application, functional changes were not required, only syntactic ones. If the model is considered compatible, the next step involves quantizing the network parameters and reducing activation function output precision, (2) in Figure 3. This is done using BrainChip's "quantize" function which requires parameters for the precision of the model input weights, internal weights, and activation function outputs. Next, the model may be retrained using quantization aware training to recover any loss in accuracy due to quantization, (3) in the Figure 3. Lastly, BrainChip provides the function "convert" to transform the quantized model into an Akida compatible model. This step removes some advanced layer types used only in training, such as Dropout functionality, (4) in Figure 3. At this point the compatible model may be evaluated in simulation using the CNN2SNN toolkit and testing data [10].

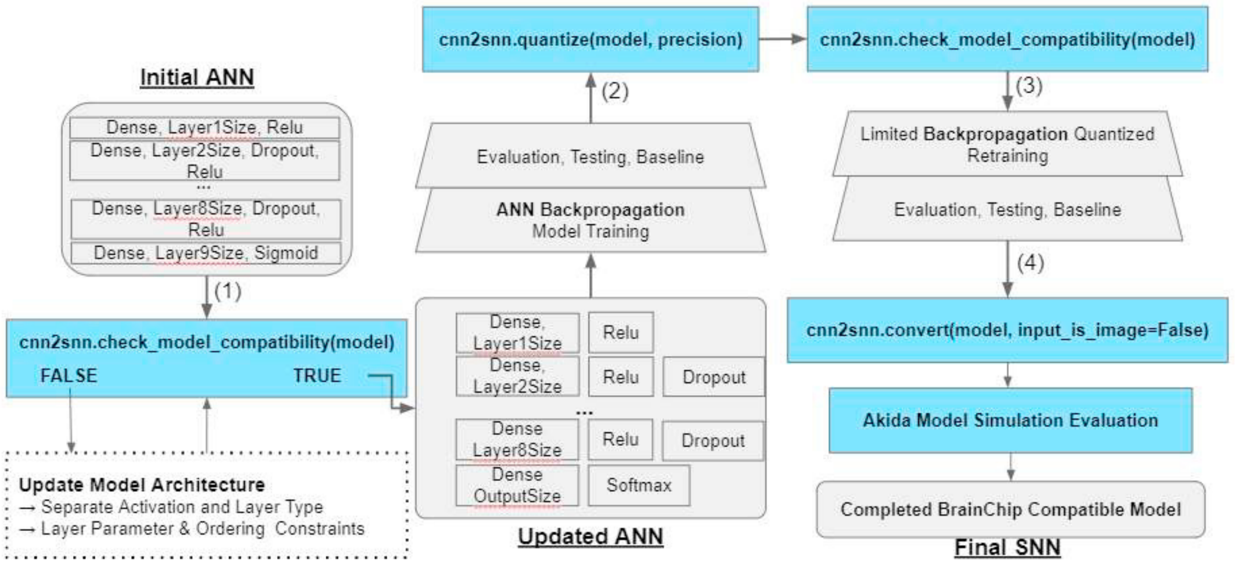
Initial ANN
Dense, Layer1Size, Relu
Dense, Layer2Size, Dropout, Relu
...
Dense, Layer8Size, Dropout, Relu
Dense, Layer9Size, Sigmoid
(1)
cnn2snn.check_model_compatibility(model)
FALSE
TRUE
Update Model Architecture
→ Separate Activation and Layer Type
→ Layer Parameter & Ordering Constraints
Dense, Layer1Size
Relu
Dense, Layer2Size
Relu
Dropout
...
Dense Layer8Size
Relu
Dropout
Dense OutputSize
Softmax
Updated ANN
ANN Backpropagation Model Training
Evaluation, Testing, Baseline
(2)
cnn2snn.quantize(model, precision)
cnn2snn.check_model_compatibility(model)
(3)
Limited Backpropagation Quantized Retraining
Evaluation, Testing, Baseline
(4)
cnn2snn.convert(model, input_is_image=False)
Akida Model Simulation Evaluation
Completed BrainChip Compatible Model
Final SNN

Figure 3: BrainChip ANN to SNN conversion process.

For the ANN we have constructed, achieving Akida compatibility and a SNN simulation required a few syntactic changes to the model's definition, but no functional changes. Specifically, all layers that used Keras' latest layer definition standard (defining the processing layer with its associated activation function in a single call) required modification to separate the activation and processing components of the layer. Syntactically this requires an extra line of code, but functionally the model is identical. This was required for all dense layers in the model.

Akida also constrains the initial (input) layer type. This layer must be a Dense or Convolutional layer such that it may be converted to Akida's generic input layer type, InputData, or its image input layer type, InputConvolutional.

Quantized layers must be of a certain maximum precision to be compatible for conversion. The learned weights of processing layers (i.e., layers with learned parameters) must be unsigned integers with at most four bits of precision. This means that, when using the provided "quantize" function, the weight_quantization parameter must be at most four. Similarly, activation function output bit width must be limited to at most four bits. For this requirement to be met, the activ_quantization parameter in the "quantize" function call must be at most four. We found that this activation function precision limitation greatly affects the accuracy of the quantized model. In future work, we hope to study this further.

The input data, i.e., the inputs to a model's inference procedure also have certain limitations. Firstly, the data must be of the unsigned integer type. Secondly, input features values must also be no larger than 24 or 15 in value. This step is of great importance to final Akida model accuracy, as heavily skewed or zero centered data may not be translated well to an unsigned format. In this scenario, we found that scaling the input data improves Akida model accuracy. Finally, the input data must be of shape [Batch Size, 1, 1, Num. Features] for prediction to function properly.

The process of converting the existing ANN into an Akida compatible model resulted in a moderate performance drop. Obtaining the 82% accuracy in Akida hardware simulation accuracy required experimentation. The experiments, excluded for brevity, fell into two categories. First were experiments with CNN2SNN toolkit's various built-in conversion methods. This included attempting various built-in weight quantizers and activation function quantizers all at different levels of quantization precision. All of these experiments resulted in high accuracy ($\geqslant$ 87%) prior to Akida hardware compatible conversion, however the converted models performed at a near-random accuracy in simulation. We hope to investigate this further in upcoming work.

The second set of experiments involved three components: (A) data scaling, (B) quantization aware training, and (C) step-based quantization precision reduction. For (A), scaling the data from the original range of between zero and one to a more low precision compatible range of zero to ten resulted in higher simulation accuracy. (This was done by multiplying the input data by 10 and then rescaling to a 4-bit unsigned integer). For (B), retraining the quantized model post-quantization and pre-conversion produced a model with greater post-quantization performance. Lastly for (C), the quantized model (not converted) performed better at lower precisions when it was first quantized to a higher precision (i.e., full precision -> 8-bit precision), then retrained, and then converted to a lower precision (i.e., 8-bit retrained -> 4-bit precision). The results of these experiments were that a combination of all three methods resulted in an Akida hardware simulation accuracy of 82.5%, far greater than random. It should be noted that more experiments may provide insight into which methods had the strongest impact on this result, and which methods could be pushed further to provide an even better final model.

3.4. Design space exploration

To perform an algorithmic design space exploration without neuromorphic processor specific constraints, we explored simple control knobs [11] in design time as well as in runtime that can be used to reduce inference latency without compromising accuracy or involving complex and computationally costly training/fine-tuning approaches for the SNN.

For design time control, we compared two neuron types: Reset-Integrate-and-Fire (RIF) and Subtractive–Integrate-and-Fire (SIF). SIF neurons help to reduce the accuracy degradation of converted SNNs by removing the discontinuity from the neuron function at the event when a neuron fire [12]. However, this is achieved at the cost of higher spiking activity. The number of computations however do not greatly increase for the SIF model (RIFs use about 84% of the number of operations of SIF), and it has reduced delay and better accuracy than the RIF model. Thus we studied the SIF model. For run time control, the neural threshold of each network layer is an additional hyperparameter that can be tuned. By decreasing it, one can increase inference speed but as a result compromise on accuracy slightly due to coarser approximations of network dynamics. One common technique called threshold balancing [13] works by normalizing the neuron thresholds to the maximum ANN activation by passing the entire dataset through the network after training is completed. However using the maximum ANN activation results in a much higher latency since higher threshold results in fewer neurons firing. Here, we explored threshold balancing using different percentiles from the activation histogram rather than just choosing the maximum activation value. We observed that network latency reduces as we choose lower percentiles, however it also comes at a cost of lower accuracy. Thus we chose a percentile value such that the accuracy degradation is minimal. We also explored an Early Exit inference method in order to reduce unnecessary computations as well as improve latency. In this strategy, we assign a particular threshold value to the final layer, and if the maximum membrane potential of the neurons in the final layer crosses that value, then we consider that the SNN inference is complete. This results in an efficient and dynamic SNN inference procedure such that simpler instances can be classified earlier, thus reducing redundant computations and resulting in lower average inference latency.

Now we list additional details of training our networks: ANNs were trained with constraints of no bias and batch-normalization layers in accordance with previous work [14]. Dropout layers were inserted after every ReLU layer to introduce some degree of regularization in absence of batch-normalization. We directly apply the inputs as a current to the SNN instead of converting them to Poisson spike trains [15]. The trained ANN is converted to an iso-architecture SNN by replacing the ReLUs with IF spiking neuron nodes. Weights were quantized to 8 bits of precision. The SNN weights are normalized by randomly sampling a subset from the training set and computing the maximum ANN activities. The SNN implementation is done using BindsNET [16], a python package based on PyTorch.

4. RESULTS

4.1. Full precision deep learning network

Final accuracy was high at 93.7% (chance performance is 66%). Time to process 10,000 records, a large subset of the testing data, was 0.34 seconds and model size of 19.7 MB. The full precision network is a baseline against which to assess the other models.

4.2. Intel Loihi and BrainChip Akida

For Intel, accuracy increased with precision. At 8 bits, the Intel model had the same accuracy as the full precision model, 93.7%; at 4 bits it had 66.8% accuracy. For timing, the 8-bit model took 20 seconds and the 4-bit model took 53 seconds. These results were produced by taking the average of 30 single input runs (chosen at random) and multiplying by 10,000, the size of the dataset used by the other models. The following sub-steps were included: Transferring spikes, Executing; while the following were excluded: Transferring probes, Configuring registers, Processing timeseries. The first set of sub-steps are transferring data into the chip, executing the model, and transferring the results out. The second set of steps are for setting up probes and the model which would not be done per record in a production environment. It should be noted that the time to run the data with Intel hardware is not comparable to the time to run the data through the full precision network or the Akida simulator. This is due to running single inputs on a single neuromorphic device in serial versus software simulation which allows faster batch parallelization.

For BrainChip, accuracy also increased with precision; maximum accuracy was 82.5% at 8 bits and 66.8% at 4 bits. The 8-bit model produced results in 1.23 seconds, the 4-bit model in 1.20 seconds. Time was significantly higher than the full precision model but that is to be expected as neuromorphic hardware is being simulated whereas the full precision model does not have this constraint. Going forward, we hope to tune the BrainChip model to increase its accuracy and time its operation in hardware, not just CPU simulation.

File size of the saved trained models was greatly reduced by conversion from ANN to SNN. Both the Akida and Loihi conversion processes resulted in similarly sized saved models *prima facie*. The Intel 4-bit model had a size of 6.4 MB while the 8-bit model had a size of 6.5 MB. For BrainChip, it was 6.6 MB for both the 4-bit and 8-bit models. However there are likely to be differences when we access the Akida hardware itself, not its simulator. Compared to the change in file size because of the initial conversion, the differences in file size were not as drastic between different levels of quantization precision for both platforms (e.g., one may expect a 4-bit precision model to be half the size of an 8-bit model). This is likely due to the degree of reduction by the initial conversion to SNN (which includes the removal of some layers such as DropOut, as well as a changes in weight representations) being significantly greater than the corresponding degree of file size reduction as a result of changing precision of quantization. The different levels of quantization resulted in models which differed in size by a matter of KBs, rather than MBs.

4.3. Design space exploration

For design time control, we considered the SIF model over the RIF model because of its reduced computational overhead. We were able to achieve a maximum accuracy of 91.5% using SIF neurons and using 99.9 percentile value from the activation histogram for threshold balancing and running the network for 2150 timesteps. In contrast the RIF model achieved less than 85% accuracy regardless of the number of timesteps used. In order to get faster latency, we modified our model to use the 98.7 percentile element from activation histogram for threshold balancing and ran it for only 290 timesteps to get an accuracy of 85%. Further, utilizing our Early Exit strategy, the average value of inference timesteps is 106, which is significantly lower than 290 for the case without early exit. Additional details are shown in Figure 4.

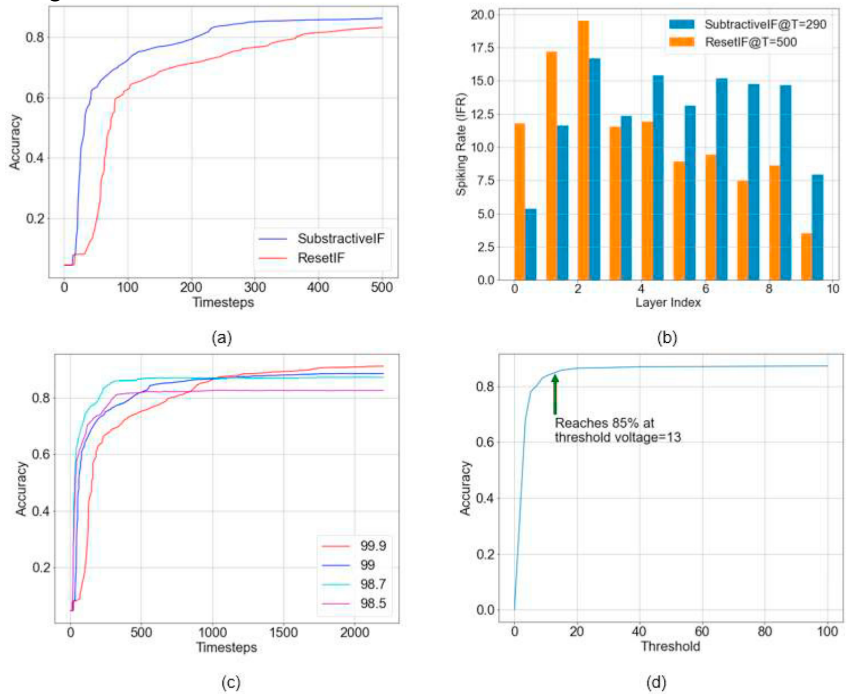

Figure 4: Design-time SNN optimization explorations: (a) Accuracy versus timesteps for full-precision model with SIF and RIF neuron models. (b) Layer wise spiking rate (total number of spikes averaged per neuron over the inference time window as specified for baseline accuracy of 85%) for full precision model with SIF and RIF neurons. Run-time SNN optimization explorations: (c) Analysis for threshold balancing. Accuracy versus timesteps for full-precision model with varying normalization factors for threshold balancing. (d) Early exit strategy where the SNN inference stops once membrane potential of wining neuron in the final layer reaches a particular threshold voltage. The accuracy reaches 85% at a voltage of ~ 13.

5. DISCUSSION AND CONCLUSION

The feasibility of a real-time HPC-scale neuromorphic cybersecurity system was assessed. We call the system Cyber-Neuro RT. We used algorithms including full precision deep learning (DL), deep learning to spiking neural network (DL-to-SNN) conversions, and design exploration within SNNs. Neuromorphic implementations of the full precision network were roughly comparable to the full precision model in terms of accuracy but would offer significant savings in power and cost. Thus the approach has been validated. We will further explore the concept along several dimensions such as further tuning networks to reduce false positives, trying the DL-to-SNN converted algorithms on hardware, and using larger or different datasets. In addition, Cyber-Neuro RT can also be deployed on GPUs if a site prefers to use GPUs over neuromorphic computers (e.g., the site may be performing other tasks on the GPUs, etc.)

For DL-to-SNN conversion, we determined that conversion of ANNs to SNNs varies across different neuromorphic vendors. Intel offers a cloud-accessible neuromorphic processing infrastructure and an API for interacting with third-party tools (e.g., NengoDL and SNN Toolbox) whereas BrainChip provides a stand-alone proprietary development/simulation environment (MetaTF SDK). These conversion tools provide a theoretically similar process for transformation from ANN to SNN, but with differences in the ANN design and requirements, as well as different conversion results. In addition, Intel and BrainChip will provide new or updated toolboxes (e.g., SLAYER for Intel), new algorithms, etc. which are likely to affect performance. Akida released their products in late 2021. Intel, on the other hand, is only offering their chips for research purposes at this time. Due to this constraint, our experiments were conducted using Intel's cloud environment and the stand-alone simulation environment for Akida. There are drastic differences between software simulation and hardware and we intend to study those further.

For design space exploration, there are design and control time knobs which can reduce inference latency while offering the same or slightly less accuracy compared to full precision models. Still other explorations are possible such as backpropagation through time or semi-supervised or unsupervised learning with spike timing dependent plasticity (STDP) learning rules.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research program under Award Number DE-SC0021562.

References

[1] Follett, David R., Duncan Townsend, Pamela L. Follett, Gabe D. Karpman, John H. Naegle, Roger A. Suppona, James B. Aimone, and Conrad D. James. (2017) "Neuromorphic data microscope." Technical Report No. SAND-2017-8752J. Sandia National Lab.

[2] Aimone, James Bradley, Christopher H. Bennett, Suma George Cardwell, Ryan Anthony Dellana, and Patrick Xiao. (2020) "Mosaic, The Best of Both Worlds: Analog devices with Digital Spiking Communication to build a Hybrid Neural Network Accelerator." Technical Report No. SAND-2020-10583. Sandia National Lab.

[3] Vineyard, Craig Michael, Ryan Dellana, James Bradley Aimone, and William Mark Severa. (2021) "Low-Power Deep Learning Inference using the SpiNNaker Neuromorphic Platform." Technical Report No. SAND-2019-2471R. Sandia National Lab.

[4] Alam, Md Shahanur, B. Rasitha Fernando, Yassine Jaoudi, Chris Yakopcic, Raqibul Hasan, Tarek Mohammad Taha, and Guru Subramanyam. (2019) "Memristor Based Autoencoder for Unsupervised Real-Time Network Intrusion and Anomaly Detection." In Proceedings of the International Conference on Neuromorphic Systems. ACM, New York, Article 2, 1–8.

[5] Alam, Md Shahanur, Chris Yakopcic, Guru Subramanyam, and Tarek Mohammad Taha. (2020) "Memristor Based Neuromorphic Adaptive Resonance Theory for One-Shot Online Learning and Network Intrusion Detection." In Proceedings of the International Conference on Neuromorphic Systems 2020. ACM, New York, Article 25, 1–8.

[6] Chelian, Suhas and Narayan Srinivasa. (2016) "Bio-inspired method and apparatus for feature detection with spiking dynamics." (U.S. Patent No. 9,443,189). U.S. Patent and Trademark Office.

[7] Kim, Edward, Jessica Yarnall, Priya Shah, and Garrett T. Kenyon. (2019) "A Neuromorphic Sparse Coding Defense to Adversarial Images." In Proceedings of the International Conference on Neuromorphic Systems 2019. ACM, New York, Article 12, 1–8.

[8] Moustafa, Nour and Jill Slay. (2015) "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." In Proceedings of the IEEE Military Communications and Information Systems Conference (MilCIS). IEEE, Piscataway, NJ, 1-6.

[9] Rueckauer, Bodo. (2022, April 25). Spiking neural network conversion toolbox. https://snntoolbox.readthedocs.io/en/latest/.

[10] BrainChip, Inc. (2022, April 25). CNN2SNN Toolkit. https://doc.brainchipinc.com/user_guide/cnn2snn.html.

[11] Lu, Sen and Abhronil Sengupta. (2020) "Exploring the Connection Between Binary and Spiking Neural Networks." Frontiers in Neuroscience, 14 (June 2020). https://doi.org/10.3389/fnins.2020.00535.

[12] Rueckauer, Bodo, Iulia Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. (2017) "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification." Frontiers in Neuroscience, 11 (Dec 2017), 682. https://doi.org/10.3389/fnins.2017.00682.

[13] Diehl, Peter U., Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. (2015) "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing." In Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN), 2015, pp. 1-8, doi: 10.1109/IJCNN.2015.7280696.

[14] Sengupta, Abhronil, Yuting Ye, Robert Wang, Chiao Liu and Kaushik Roy. (2019) "Going deeper in spiking neural networks: VGG and residual architectures." Frontiers in Neuroscience, vol. 13, (March 2019). https://doi.org/10.3389/fnins.2019.00095.

[15] Rueckauer, Bodo and Shih-Chii Liu. (2018). "Conversion of analog to spiking neural networks using sparse temporal coding." In Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, Piscataway, NJ, 1–5.

[16] Hazan, Hananel, 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, 89. (December 2018). https://doi.org/10.3389/fninf.2018.00089.