Optimizing neural networks for next-gen AI

Andrew Deutsch[1], Yihui Ren[2], Sandeep Mittal[3]

[1]*Department of Physics, Stevens Institute of Technology, Hoboken, NJ 07030*

[2]*Mentor and* [3]*Co-mentor, Computational Science Initiative, Brookhaven National Lab, Upton, NY 11973*

August 11, 2021

Most scientific data challenges involve real-time decision making on high-volume data flows. Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) have been promising solutions for many challenging problems, but are limited to training and evaluation in an offline manner on a Graphics Processing Unit (GPU). In this work, we aim to transform pre-trained DNNs and CNNs into Spiking Neural Networks (SNNs), a better, biologically inspired form that can be implemented and deployed on novel hardware. Unlike discrete matrix multiplication based DNNs, an SNN acts on continuous trains of signals. First, we investigate the sparsity nature of DNNs, pruning techniques, and their connections to SNNs. We then investigate DNN to SNN conversion using NengoDL, an existing framework based on TensorFlow. Specifically, we optimize the accuracy of converted models by properly scaling the firing rates of neurons and applying a lowpass filter over the spike trains. We systematically run these experiments using the popular benchmark image datasets MNIST and CIFAR-10. For further optimization, we limit scaling of firing rates to only one layer and we find evidence that scaling late layers in a converted DNN and early layers in a converted CNN leads to higher accuracy. Further, we investigate the cause of this discrepancy by extracting firing rate data from both networks. Our findings provide insight into the co-design of novel energy-efficient neuromorphic chips for SNNs.

I. INTRODUCTION

Spiking Neural Networks (Figure 1) are fundamentally time-dependent, as they operate on continuous trains of discrete spikes. Instead of representing information as a vector of real numbers, as is done with conventional DNNs and CNNs, the SNN encodes these values in the frequency domain with higher spiking rates corresponding to higher values and vice versa. The Rectified Linear Unit (ReLU) activation functions at each neuron of conventional networks are replaced with Leaky Integrate and Fire (LIF) neurons (Figure 2), which compute a membrane potential as a weighted sum of the input spikes followed by exponential decay. When the

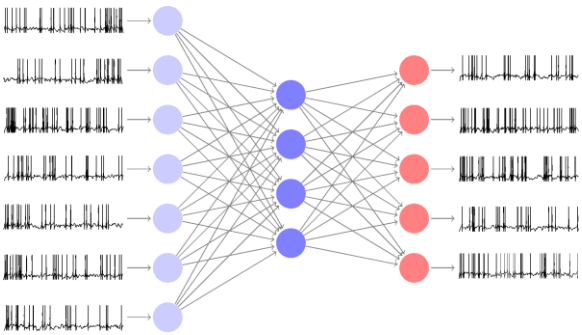

Figure 1: *An illustration of a simple 3-layer Spiking Neural Network, complete with input and output spike trains.*

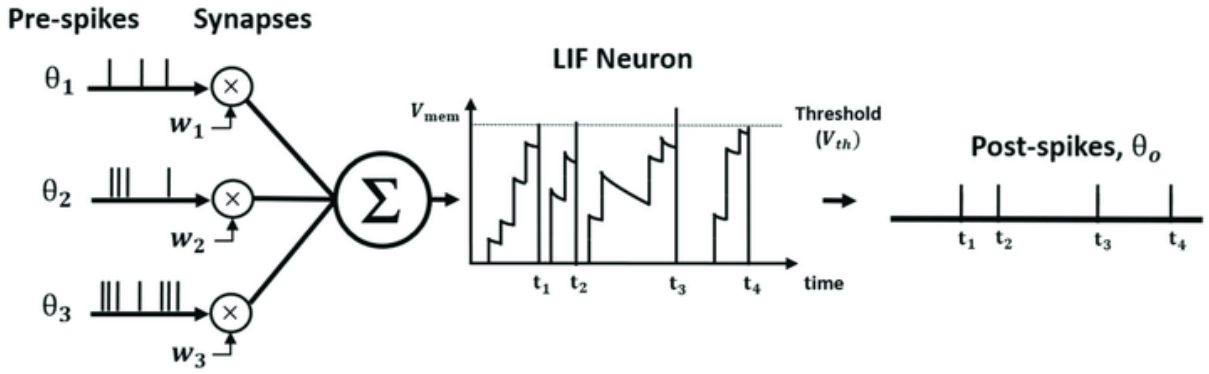

Figure 2: *The Leaky Integrate and Fire (LIF) neuron, transforming three presynaptic spike trains into a single postsynaptic spike train.*

membrane potential exceeds a threshold, the neuron fires, thus transforming any number of input spike trains to one output spike train.

The sparsity nature of SNNs means they provide approximately the same functionality as DNNs and CNNs, but much less communication is required. While all of the results in this paper are obtained from tests on a digital computer, SNNs are capable of implementation of novel energy-efficient analogy neuromorphic chips. It is for this reason that their optimization is of great importance.

We begin with an analysis of network pruning techniques. Network pruning is a common method of reducing the size of a neural network while maintaining a desired level of accuracy. During the training process, neuron weights that are found to have little to no effect on the outcome of the network are sent to zero. The result is that much less information is required to store these networks and, in some cases, accuracy can improve. Often times, the prescribed network is much larger than needed to solve the problem at hand. We can further specify a sparsity schedule to prune the network after a particular epoch. In this work, we test the limits of network pruning on a simple CNN trained to read handwritten digits in the MNIST dataset (Figure 3). While network pruning is a crucial part of optimizing any network, it is not specific to SNNs and we instead choose to focus on optimizing the conversion process of DNNs and CNNs to SNNs.

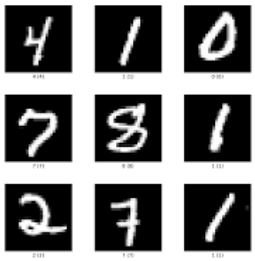

Figure 3: *A small subset of the MNIST image dataset of handwritten digits used to train these pruned networks. It consists of 70,000 images belonging to 10 classes.*

Our analysis of SNNs begins as a two-dimensional optimization problem. We then test the effects of scaling the firing rates in individual layers. Finally, we justify our results by directly measuring the firing rates in layers throughout our SNNs. All SNN optimization tests are systematically performed using NengoDL on the benchmark image dataset CIFAR-10 shown below.

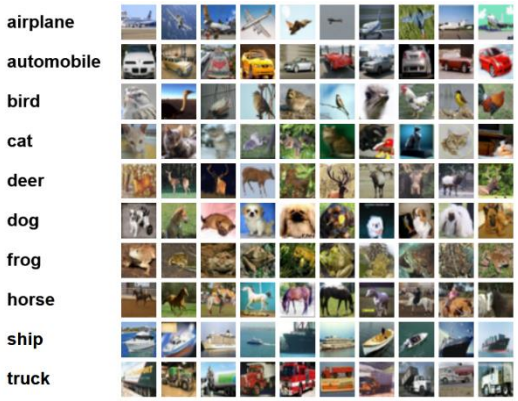

Figure 4: *The CIFAR-10 image dataset used to train and test our SNNs consists of 60,000 images belonging to 10 classes.*

II. NETWORK PRUNING

To test the effect of network pruning, we construct a simple CNN in the TensorFlow framework. This network consists of two sets of 3x3 convolutional and 2x2 pooling layers with 28 and 56 neurons, respectively. These layers are followed by a `Flatten()` layer, a fully connected `Dense()` layer with 300 neurons and finally a `Dense()` output layer with `softmax` activation functions. We utilize the traditional ReLU activation function for all other layers.

The MNIST dataset is then normalized to make all input values between 0 and 1, and one-hot encoding is used on the labels.

Now, we are ready to train out network with varying levels of sparsity. Sparsity values range from 0% to 90% and represent the proportion of weights in the original network sent to zero during training or, equivalently, the reduction in size of the network. The sparsity schedule is specified to be a polynomial pruning which takes effect after epoch 5. The plots below show the training accuracy recorded over 20 epochs for each of the networks. For simplicity, we only present the training accuracy for networks with sparsity above 60%.

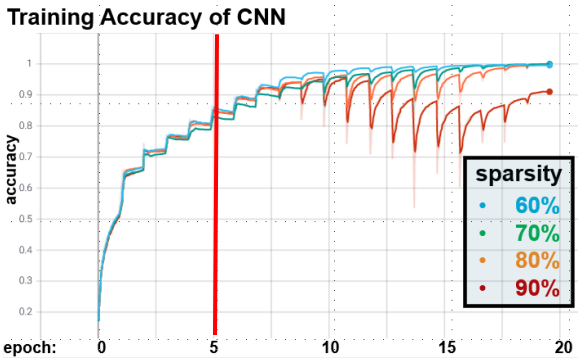

Figure 5: *Training accuracy over 20 epochs of the same CNN pruned with varying sparsity after epoch 5.*

Our findings indicate that this simple CNN can be pruned by up to 80% (only 20% of the initial weights remain) before the training accuracy begins to suffer. Only at 90% sparsity do we observe a dip in training accuracy after 20 epochs. This result hints at the ease of training networks for MNIST; very few neurons are required for a simple CNN to correctly interpret these handwritten digits 99% of the time. This result serves mainly as a proof-of-concept that accuracy can be maintained while drastically reducing the size of a network.

III. SNN OPTIMIZATION

To optimize the conversion process of DNNs and CNNs to SNNs, we make use of the NengoDL framework. This provides us with the tools necessary to swap neuron types and encode all information in the spike trains. However, a brute force translation to SNNs usually results in very poor accuracy. To remedy this, we make use two parameters: `synapse` and `scale_firing_rates`. The `synapse` parameter is the time constant τ of a low-pass filter which is applied over all the spike trains. This behaves as a rolling average in an exponential window function.

$$f(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}}$$

The `scale_firing_rates` parameter simply scales up the firing rates so as to increase the amount of information propagating through the network while making communication less sparse.

For the sake of comparison, we run all the following optimization tests on both a converted DNN and a converted CNN, with architectures shown to the right.

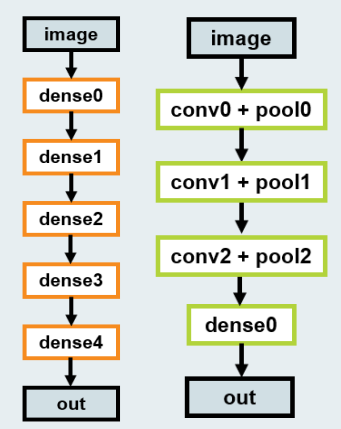

Figure 6: *Architectures for DNN (left) and CNN (right)*

IV. SYNAPSE AND SCALING

We are now presented with a two-dimensional optimization problem: how can we obtain optimal test accuracy for our converted SNN as we vary both `synapse` and `scale_firing_rates`? To investigate this problem, we generate a heat map of the test accuracy for various combinations of these two parameters. Specifically, we vary `synapse` from 0 (None) to 0.045 seconds and `scale_firing_rates` from 1 to 91.

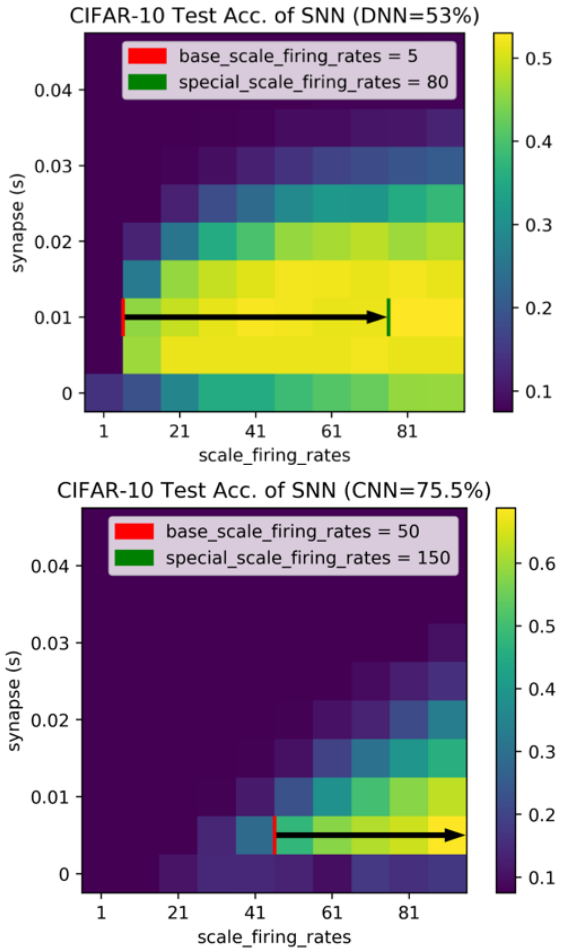

Figure 7: *Test accuracy heat maps for the converted DNN (top, with non-spiking accuracy of 53%) and CNN (bottom, with non-spiking accuracy of 75.5%). Arrows illustrate scaling of individual layers (Section V)*

Accuracy is observed to increase with `scale_firing_rates` directly, while `synapse` has an optimal value in both cases. While increasing the firing rates results in higher accuracy, this comes with a tradeoff: communication is less sparse. As we increase the firing rates to infinity, the accuracy of the SNN approaches that of the non-spiking network, but this defeats the entire purpose of the SNN as it will require an enormous amount of information processing.

Somewhat more interestingly, as the low-pass filter time constant `synapse` increases, we see an initial jump, followed by a steady decrease in accuracy. This means we can extract these optimal values of `synapse` for each network (0.01 seconds for DNN and 0.005 seconds for CNN). The behavior of `synapse` indicates that there exist specific low-pass filters which provide optimal accuracy for each network.

V. SCALING BY LAYER

NengoDL also allows us to limit scaling of firing rates to specific layers, posing another optimization problem. Perhaps we can achieve the same levels of accuracy by only scaling the firing rates in one layer. We do exactly this, only we provide a base scaling to all layers so as to maintain decent accuracy. This can be visualized in Figure 7, where we scale all layers by `base_scale_firing_rates` and only one layer by `special_scale_firing_rates`. For example, the following is the scaling configuration in the spiking DNN for `dense4`.

LAYER:	input	dense0	dense1	dense2	dense3	dense4	output
SCALE:	5	5	5	5	5	80	5

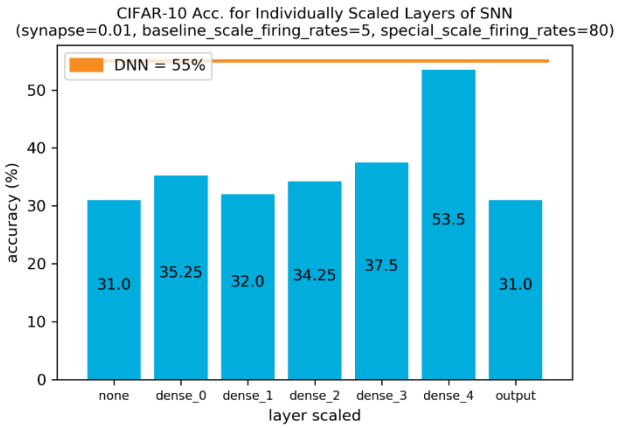

Figure 8: *Test accuracy of the spiking DNN by layer scaled.*

Here, we notice that for the converted DNN the accuracy is universally poor when only one layer is scaled, with the exception of `dense4`, the last hidden layer. In this case, the accuracy is very close (53.5%) to the baseline non-spiking accuracy (55%). Scaling the output layer has no effect. These results indicates that scaling later, rather than earlier, in the converted DNN leads to higher accuracy. The final hidden layer appears to be especially important.

Now, we repeat this test for the CNN. Scaling of pooling layers was found to have no effect, so they are omitted from the graph below.

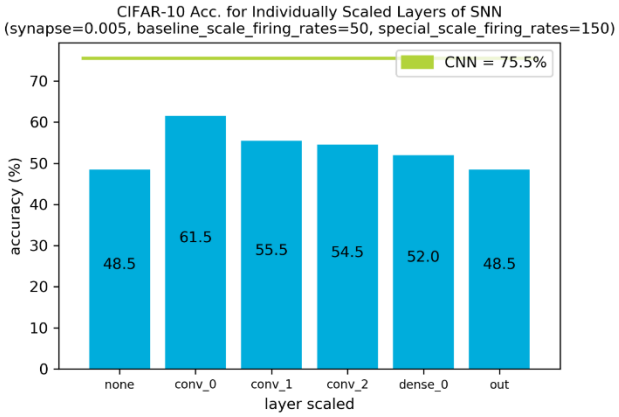

Figure 9: *Test accuracy of the spiking CNN by layer scaled.*

We find the opposite to be true in the case of the converted CNN: scaling the initial convolutional layer results in the highest accuracy. We see a steady decrease afterwards. Another discrepancy is the wider gap between the highest SNN accuracy (61.5%) and the baseline non-spiking accuracy (75.5%).

To eliminate unwanted degrees of freedom, both networks were subsequently changed so that the number of neurons in each layers was the same. The conclusion was essentially identical, showing that these results are not due to the relative number of neurons in each layer.

VI. FIRING RATES

To further investigate the cause of these results, we directly probe the neurons in each network and extract firing rate statistics by layer.

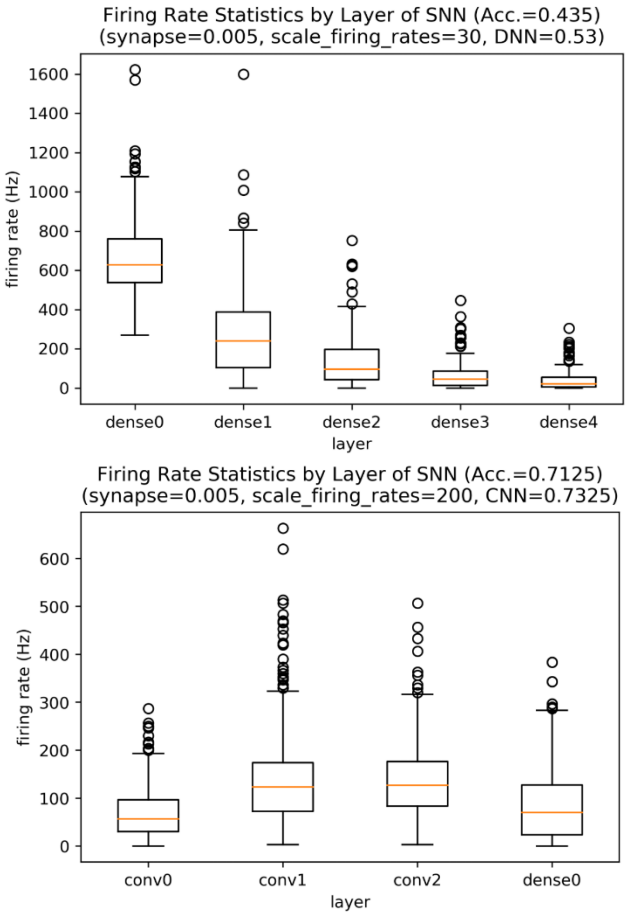

Figure 10: *Box-and-whisker plots showing the minimum, median (orange), IQR, maximum, and outliers for the firing rate data by layer of the converted DNN (top) and CNN (bottom)*

In the case of the converted DNN, spiking rates die off drastically as we move deeper into the network. The opposite appears to be true in the case of the converted CNN, where firing rates generally increase across the convolutional layers.

Looking at the firing rate statistics, together with the results from *Section V*, we can see a clear relationship that explains our findings. Scaling the firing rates is most beneficial when firing rates are comparatively low. In the DNN, this occurs in the final hidden layer and in the CNN, this occurs in the first convolutional layer. We can now conceptualize `scale_firing_rates` as a parameter which, when applied correctly, revives spike trains that have become too sparse.

To look at this result from another angle, we plot the firing rate data as a histogram with a best-fit curve.

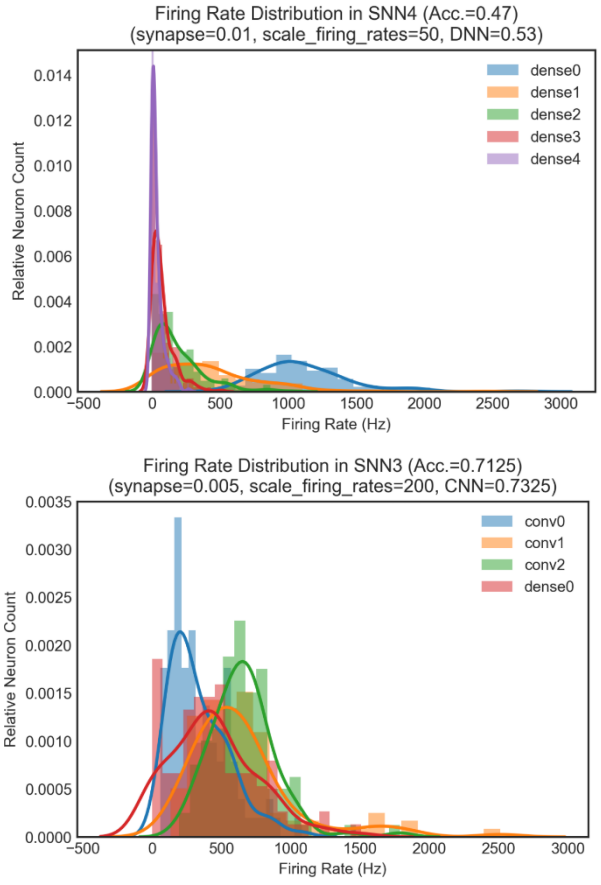

Figure 11: *Firing rate distributions by layer of the converted DNN (top) and CNN (bottom)*

Finally, we notice in the converted DNN an initially wide-spread distribution of firing rates, indicative of healthy activity. As we move deeper into the network, however, more and more neurons refrain from firing. In the converted CNN, the opposite is true: initially, many of the neurons are firing at low rates and, as we move deeper into the network, the distribution of firing rates has a greater deviation and fires at a generally higher rate.

VII. CONCLUSION

These results indicate that the conversion of traditional neural networks to Spiking Neural Networks can be optimized (1) by pre-conversion pruning techniques, (2) through the specification of a low-pass filter time constant, and (3) by scaling the firing rates in layers with comparatively sparse spike trains.

Future work involves the expansion of these networks into deeper residual spiking networks, which are capable of obtaining higher test accuracy than the networks presented.

VIII. ACKNOWLEDGEMENTS

I would like to give special thanks to my mentor, Ray, for all his guidance and support through the SULI program. I would also like to thank Sandeep Mittal for his thoughtful help and generosity. In addition, I wish to thank the Office of Educational Programs (OEP) and my OEP team leader Amy Engel for fostering a virtual environment conducive of advanced research and science communication. This project was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships Program (SULI).

IX. REFERENCES

[1]Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer and Shih-Chii Liu, "Conversion of Continuous-Values Deep Networks to Efficient Event-Driven Networks for Image Classification." https://www.frontiersin.org/articles/10.3389/fnins.2017.00682/full

[2]Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, Kaushik Roy, "Going Deeper in Spiking Neural Networks: VGG and Residual Architectures." https://www.frontiersin.org/articles/10.3389/fnins.2019.00095/full

[3]K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[4]Figure 1: https://fzenke.net/index.php/2017/02/19/learning-in-multi-layer-spiking-neural-networks/

[5]Figure 2: Lee, Chankyu & Sarwar, Syed & Panda, Priyadarshini & Srinivasan, Gopalakrishnan & Roy, Kaushik. (2020). "Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures." Frontiers in Neuroscience. 14.119.10.3389/fnins.2020.00119.

[6]Figure 3: https://deepai.org/dataset/mnist

[7]Figure 4: https://www.cs.toronto.edu/~kriz/cifar.html