Implementing a Vision Transformer Using a Spiking Neural Network

Connor Cabrera
University of Central Florida
connorcabreral7@gmail.com

Dr. Gonzalo Vaca-Castano
University of Central Florida
gonzalo.vaca@gmail.com

Abstract
Spiking Neural Networks (SNN’s) are being explored as an energy efficient alternative to traditional neural networks. On the other hand, Vision Transformers (ViT’s) have been recognized as a leading competitor to state-of-the-art models. In this work, we were successfully able to create a Hybrid SNN ViT that achieves accuracies similar to that of a regular ViT on the MNIST dataset and slightly lower on CIFAR10 dataset. By using SNN components, this model also achieves the energy efficiency inherent in those parts.

1. Introduction
As models developed by industry get larger and more complex, they require an ever increasing amount of processors and electricity to train. Companies continue to choose to make this trade-off since they have the resources at hand, however this approach is not sustainable and is harmful to the environment. Spiking Neural Networks attempt to be more efficient by mimicking biological responses in the human brain. The main differences present in an SNN is the way they encode data and how they replace a traditional activation function.

1.1. Image Encoding
In both training and evaluation, SNN’s take an input image and convert it into what are called Spike Trains. Spike Trains are simply put, a representation of an image in 0’s and 1’s. There are two methods that have been developed to do this: Temporal Encoding and Rate Encoding. Temporal Encoding makes it so that higher pixel intensities spike first on the train, and Rate Encoding is where higher pixel intensities cause a higher frequency of spikes. For this project we chose to pursue Rate Encoding as it was easier to understand and implement in the model.

1.2. Rate Encoding
To go into further detail on Rate Encoding, first each image has its values Normalized between 0 and 1. This becomes the spiking probability for each pixel, or the chance that the encoded image will have a 1 at that location. We then reconstruct the image before feeding it to the network.

1.3. Leaky Integrate and Fire
SNN’s are similar to regular Artificial Neural Networks (ANN’s) in that they both go through the weighted summations of their layer neurons. However, where an ANN would have an activation function such as RELU (or in our case GELU), SNN’s have what is called a Leaky Integrate and Fire (LIF) neuron. The input causes a Membrane Voltage to increase until it reaches a threshold at which point the voltage resets and an output spike is produced. The absence of a spike on the input causes the voltage to decay. The output spike train is complete once you have reached the end of the input train.

2. Related Work
2.1. Methods of Developing SNN’s
There are three mainstream methods of creating / training SNN’s.

2.1.1 Spike Timing Dependent Plasticity Learning (STDP)
This is an unsupervised approach where the input and output spike trains are compared to each other. Neurons whose input trains had spikes before or after a spike on the output train get their weights increased or decreased, respectively. The greater the distance between those spike comparisons, the greater the weight update. Unfortunately this has only proven successful for simple tasks as its size and computation increase exponentially with larger problems.

2.1.2 ANN to SNN Conversion
This method takes a pre-trained ANN and converts all activation functions with LIF neurons. The goal is to reach a statistical equivalence to the old activation function which lies in properly configuring the threshold at which an output spike would be produced. These converted models show an increase in efficiency during execution, however they still
required the weights of a pre-trained model somewhat defeating the purpose. They also require a very large number of time-steps, a concept which will be covered in the 3. Approach section.

### 2.1.3 Surrogate Back-propagation

Spike Trains are discontinuous, making it so you can not perform gradient descent to update the weights like you would with an ANN. Surrogate Back-propagation allows you to approximate the derivative. This results in a trainable SNN that requires a low number of time-steps and produces a high accuracy. This is the approach we chose to pursue for our work.

### 2.2. SNN Tool Box [7] [6]

SNN Tool Box is a Python GitHub Repository that provides a library which can perform the ANN to SNN conversion and evaluate it. It also comes with numerous visualization methods to make it easy to understand the I/O of each layer and review activations (a measure of efficiency). Early on in the project, before finding a way to perform surrogate back-propagation, we explored this as how we’d convert a ViT to an SNN ViT. In addition to the inherent disadvantages of ANN to SNN conversion, this library is meant for simple linear models and isn’t in active development, meaning it would be difficult to convert a complex model such as a ViT.

### 2.3. snnTorch [1]

snnTorch is a Python GitHub Repository that fully integrates SNN encoding and LIF into PyTorch.

#### 2.3.1 Spikegen Rate Conv

This function performs all the necessary operations as described in 1.2 Rate Encoding. It takes an image with normalized values as input and returns the spike train representation.

#### 2.3.2 SNN Leaky

This function fully implements the LIF neuron and it is trainable being based off the PyTorch Heaviside Step Function. [5]

We used the following values for the function:

- Beta (Voltage Decay Rate): 0.95
- Membrane Threshold: 1
- Learnable Beta and Threshold: True

### 3. Approach

The best results were achieved by creating a Hybrid SNN ViT. We converted the MLP of each Transformer Block to LIF, however the final MLP Head at the very end of the model needed to remain with GELU or else the accuracy
would be unusable. At this time, the cause of this is unknown. To elaborate on the concept of time-steps mentioned earlier, when converting an image to its spike representation, only using 0’s and 1’s creates a sparsity that might make it difficult for the model to perform classification. We introduce time in order to have multiple spike trains for the same image, which, averaged out more accurately represent the original. To test this, three different versions of the Hybrid SNN ViT were created.

### 3.1. No Time

This is the most basic form of the Hybrid SNN ViT. It incorporates no aspect of time and trains the model based on a single rate encoding of each image.

### 3.2. Time Averaged Output (TAO)

This version takes a time-step parameter in the training and evaluation functions. For each number of time-steps specified, the model creates a rate encoded image and gathers the outputs. It then averages those outputs and sends the result to the loss function where it is compared with the target.

### 3.3. Time Averaged Loss (TAL)

Time Averaged Loss is similar to Averaged Output in that it takes time as a parameter. In this implementation, it calculates a full output for each rate encoding of the input image and calculates the loss for each. It then takes the average of those losses before performing back-propagation.

### 4. Results

The goal of this project was to create a model with an accuracy similar to that of a normal ViT using SNN components. Although we are not able to directly compare the power efficiency as we’d need a special piece of hardware called a Neuromorphic Processor to take advantage of the SNN, it is safe to assume that using the SNN components does in fact cause better efficiency as the sparsity from Rate Encoding and the LIF neuron will cause fewer neuron activations.

| Validation Accuracy | | | |
|----------------------|-----|----------------|---|---------|
| Dataset              | Base ViT | Hybrid No Time | Hybrid TAO | Hybrid TAL |
| MNIST                | 97.94% | 97.36%          | 97.27%      | 97.29%     |
| CIFAR10              | 82.25% | 67.23%          | 69.04%      | 68.17%     |

Time Hybrids were run using (5) Time-steps for MNIST.
Time Hybrids were run using (5) Time-steps for CIFAR10.

For a more complex dataset such as CIFAR10, the hybrid models struggle to keep up. Incorporating time clearly provided an advantage. In earlier epochs, they were about 5% more accurate than not using time, though the gap began to close towards the end, with Time Averaged Output pulling ahead of Time Averaged Loss. Increasing the number of time-steps should improve the model, however we were unable due to time and memory constraints.

### 5. Conclusion and Future Work

For the most part, we achieved what we set out to do, but there is plenty of room for improvement. First and foremost would be optimizing the code in order to solve the memory issues and test how much larger time-steps improve the results. Additionally, we’d want to test the Hybrid SNN ViT on more datasets: CIFAR100, SVHN, and eventually Im-
ageNet. This would be accomplished by first tuning our base ViT from scratch to achieve standard validation accuracies for each dataset, as was done for MNIST and CIFAR10. We are unable to load pre-trained models as we must access the architecture in order to perform the hybrid conversion. Another issue that was encountered was that the model was quickly overfitting on these datasets, but that was resolved with data augmentation. This should phase out as the datasets get more complex. Once a proper base has been established, we will perform the Hybrid conversion hoping for a similar accuracy.

5.1. Natural Robustness to Adversarial Attacks

This aspect of the project is incredibly promising, but the idea was just conceived and is purely hypothetical at this time. Adversarial Attacks add noise to an image, indistinguishable to the human eye, with the aim of causing the model to incorrectly classify an image.

One of these attacks is called the Projected Gradient Descent (PGD) attack. This is a white-box attack, meaning the attacker knows the model’s weights. We hypothesize the act of Rate Encoding an image fed to the Hybrid SNN ViT is enough to completely eliminate the presence of threat from adversarial noise in an image. The goal of the attacker is to fool the model while remaining undetected to humans. This means their perturbations are slight enough such that a person can not tell the image has been tampered with, but the model is thrown off. These changes are so slight that, while they may disrupt a regular model’s weights, once used as probabilities for Rate Encoding the perturbations will vanish. This should be especially true if averaged out over multiple time-steps. So long as the Hybrid SNN ViT can achieve good accuracy on a non-adversarial dataset, it should possess a natural robustness as a total byproduct. This aspect will be explored in the coming months.

References