SPIKING NEURAL NETWORKS

Daniel Price and Brandon Nguyen

Artificial Neural Networks

- Poorly represent the human brain
- Matrix Multiplication (GEMM)
    - More Computation -> More Energy
- Offline learning
- Backpropagation
    - High training Cost

Spiking Neural Networks

- Incorporates neuron models
- Spikes convert GEMM into accumulation
    - Less Computation -> Less Energy
- Online learning
- Surrogate Gradient Descent / STDP (Unsupervised)
    - Low training cost

Applications

- **Robotics**
- Neuroscience
- **Autonomous systems**
- Natural language processing
- **Computer vision**
- etc.

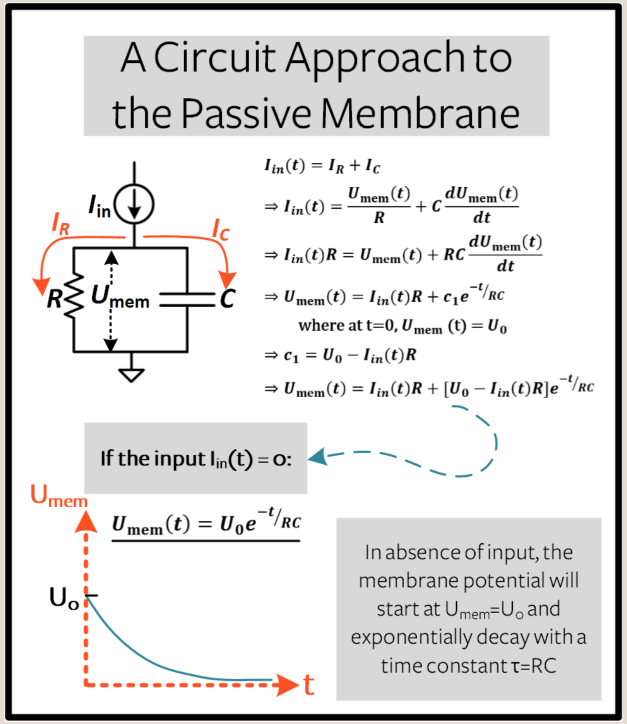
Canada's Mars Exploration Science Rover

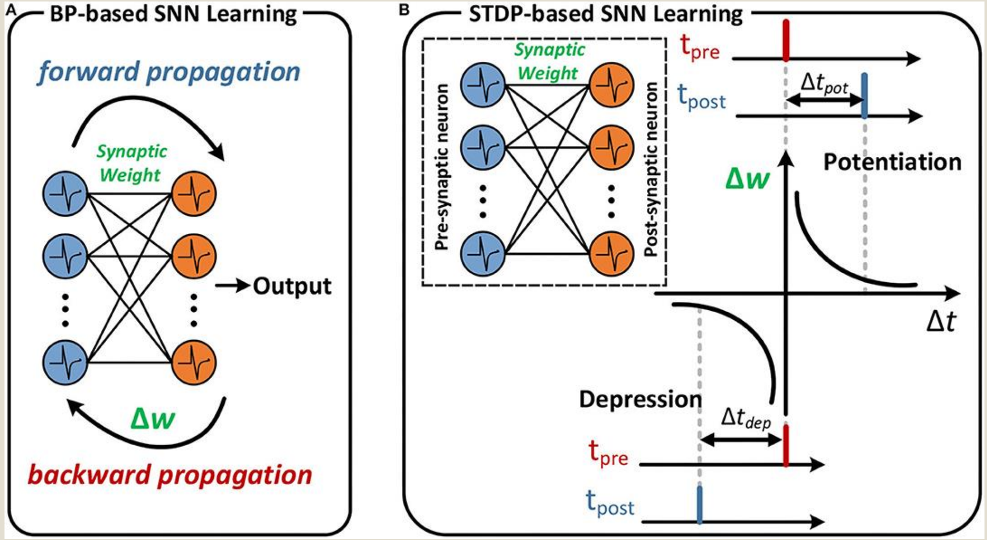
Leaky-Integrate and Fire

- Simple Neuron Model
- Represents neurons as an RC circuit
- Input current increase membrane potential
- When membrane potential breaks threshold, potential resets

Learning

- Surrogate Gradient Descent
  - Step function learning (spike or no spike)
  - Replace derivative with a surrogate function
- STDP (Spike-Time Dependent Plasticity)
  - Unsupervised learning
  - Updates weights on spiking behavior
    - Presynaptic / Postsynaptic spikes





Workflow Distribution

Brandon:

- Visualization of model architecture
- Data Processing and Data Visualization construction
- Code Documentation

Daniel:

- CNN and CSNN Architecture Design
- Generation of results and case study data
- Hyperparameter tuning

Problem Description

Train a SNN on MNIST Dataset for a supervised multi-class classification task.

Evaluation Goals:

Compare with ANN / CNN on **accuracy** and **training loss**	Compare computation cost in **FLOPs** (Floating Point Operations)

Initial Approach

- LIF / STDP model
- Struggled to correlate images to labels

Why?

- STDP struggles with classification
  - Unsupervised learning
  - Learning does not use labels
- STDP is used more commonly in TNNs (Temporal Neural Networks)
- SNNs are emerging:
  - Limited application
  - Research focused

Attempts to salvage Initial Approach

Normalization & Regularization

- Varying threshold voltage
- Reward modulated STDP (R-STDP)
  - Rewards/Punishes neurons based on prediction

Hyperparameter tuning variation

- R/C/Tau
- Timesteps
- Learning Rate

New Approach

Batch Normalization (BN)

$$\rightarrow \boxed{\text{layer}} \rightarrow x \rightarrow \hat{x} = \frac{x-\mu}{\sigma} \rightarrow y = \gamma\hat{x} + \beta$$

- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

Image: http://tzutalin.blogspot.com/2017/07/deep-learning-batch-normalization-note.html

- **BNTT (Batch Normalization Through Time)**
  - Codebase detailing implementation
  - Batch Normalization
    - Normalizes inputs across batch
    - Reduces covariant shift
    - Applies to each timestep
  - Backpropagation
    - Preferable with SNNs / Classification

- **Solid foundation, but more complex than necessary.**

Code: https://github.com/Intelligent-Computing-Lab-Yale/BNTT-Batch-Normalization-Through-Time/blob/main/model.py

New Approach: Our Implementation

Simplify Architecture:

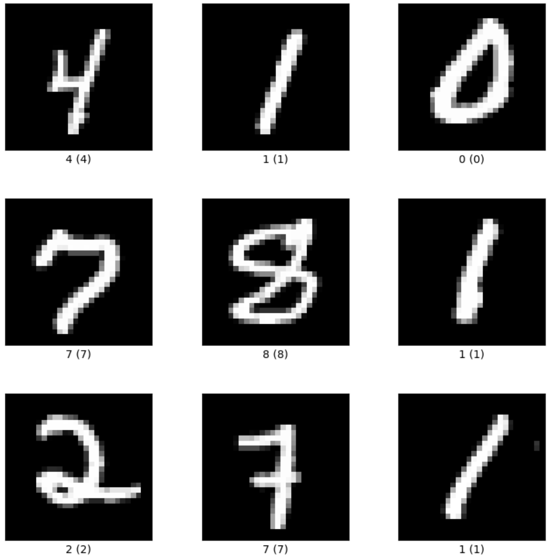
- Reduce layers from 9 to 3
- Reduce Shape / Input channels
  - Match input (MNIST)
  - Lower Complexity (hidden shapes)
- Reduce epochs and timesteps

Justification:

- Model is too complex for dataset

Experimental Methodology

- Dataset: MNIST Handwritten Digits
- Model Architecture
- Hyperparameters used
- Encoding type
- Case Study
- Evaluation: Classification accuracy, training loss, and FLOPs



MNIST Examples

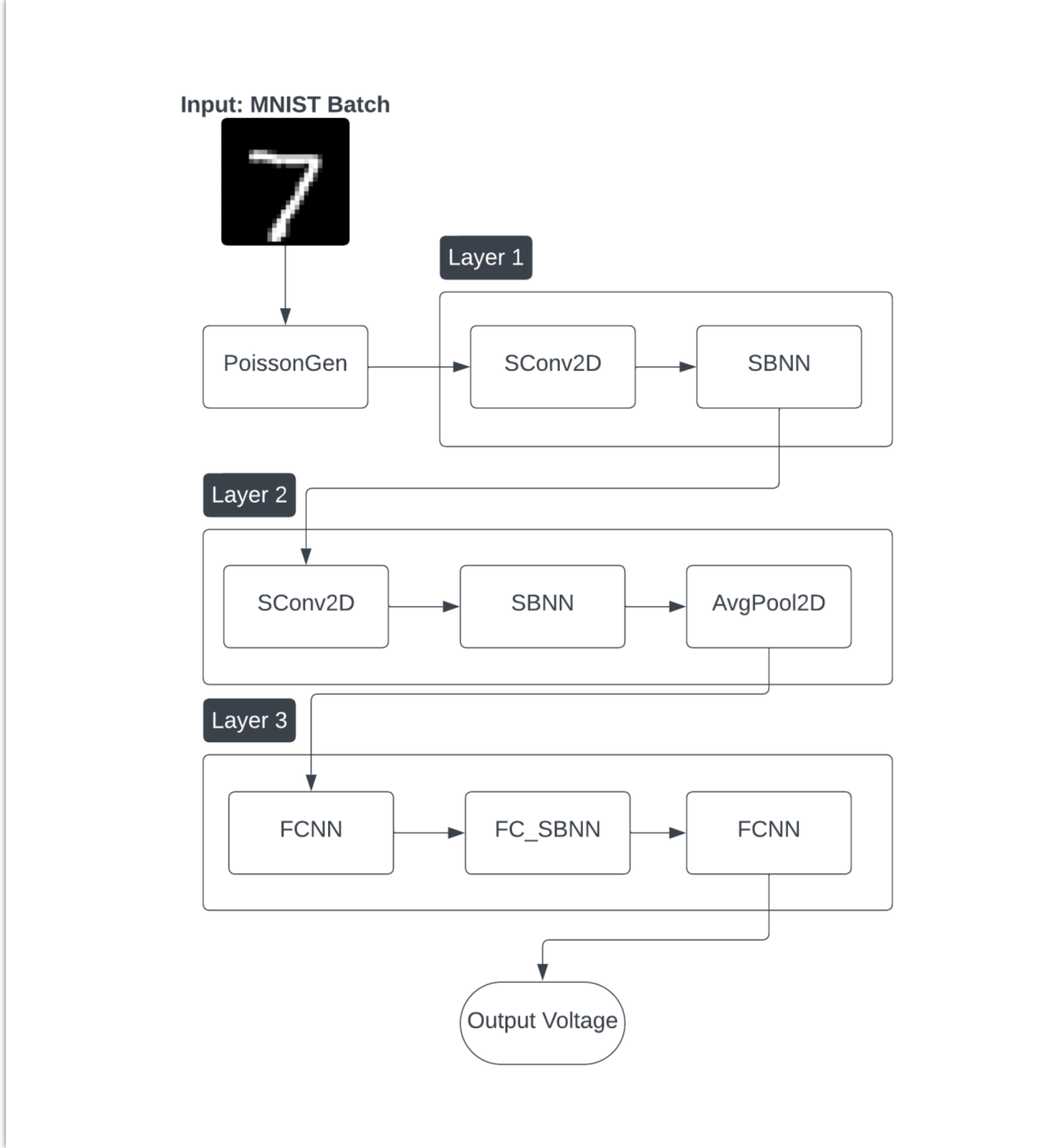
MNIST Dataset

A database of handwritten digits containing a training set of 60,000 examples, and a test set 10,000 examples.

Using 10% of the dataset due to training overhead (SNN)

Training on 6,000 samples.	Testing on 1,000 samples.

Goal: Correctly classify images as one of the ten digits.



Model

- Visual diagram of our model architecture and flow of our forward pass function if observing layers only.
- Baseline CNN ignores PoissonGen and uses non-spiking equivalents
  - SConv2D -> Conv2D

Hyperparameters - Baseline

- **Global Hyperparameters:**
  - batch size, learning rate, number of epochs
- **Convolutional Layer 1**
  - Input Channels: 1; Output Channels 64; Kernel Size: 1; Stride: 1
- **Convolutional Layer 2**
  - Input Channels: 64; Output Channels: 64; Kernel Size: 3; Stride: 1
- **2D Batch Normalization between Layers**
  - Input Channels: 64; Epsilon: 1e-4; Momentum: 0.1;
- **Pooling (after L2 and BN)**
  - Average Pooling; Kernel Size: 2
- **Fully-connected Layer 1**
  - Input Dimension: 14 * 14 * 64; Output Dimension: 512
- **1D Batch Normalization**
  - Input Dimension: 512; Epsilon: 1e-4; Momentum: 0.1
- **Fully-connected Layer 2**
  - Input Dimension: 512; Output Dimension: 10
- **Activation Function**
  - Rectified Linear Unit following each BN operation.

Hyperparameters - CSNN

Global Hyperparameters:
- number of time steps, leakage memory, batch size, learning rate, number of epochs

Convolutional Layer 1
- Input Channels: 1; Output Channels 64; Kernel Size: 1; Stride: 1

Convolutional Layer 2
- Input Channels: 64; Output Channels: 64; Kernel Size: 3; Stride: 1

Spiking 2D Batch Normalization between Layers
- Input Channels: 64; Epsilon: 1e-4; Momentum: 0.1;

Pooling (after L2 and BN)
- Average Pooling; Kernel Size: 2

Fully-connected Layer 1
- Input Dimension: 14 * 14 * 64; Output Dimension: 512

Spiking 1D Batch Normalization
- Input Dimension: 512; Epsilon: 1e-4; Momentum: 0.1

Fully-connected Layer 2
- Input Dimension: 512; Output Dimension: 10

Activation Function
- N/A

Encoding Type

Rate Encoding

- Represents data as firing rate (frequency)

Latency Encoding

- Represents data as spike timings

BNTT uses rate encoding

- SNNs prefer Rate encoding
- TNNs prefer Latency encoding

Case Study

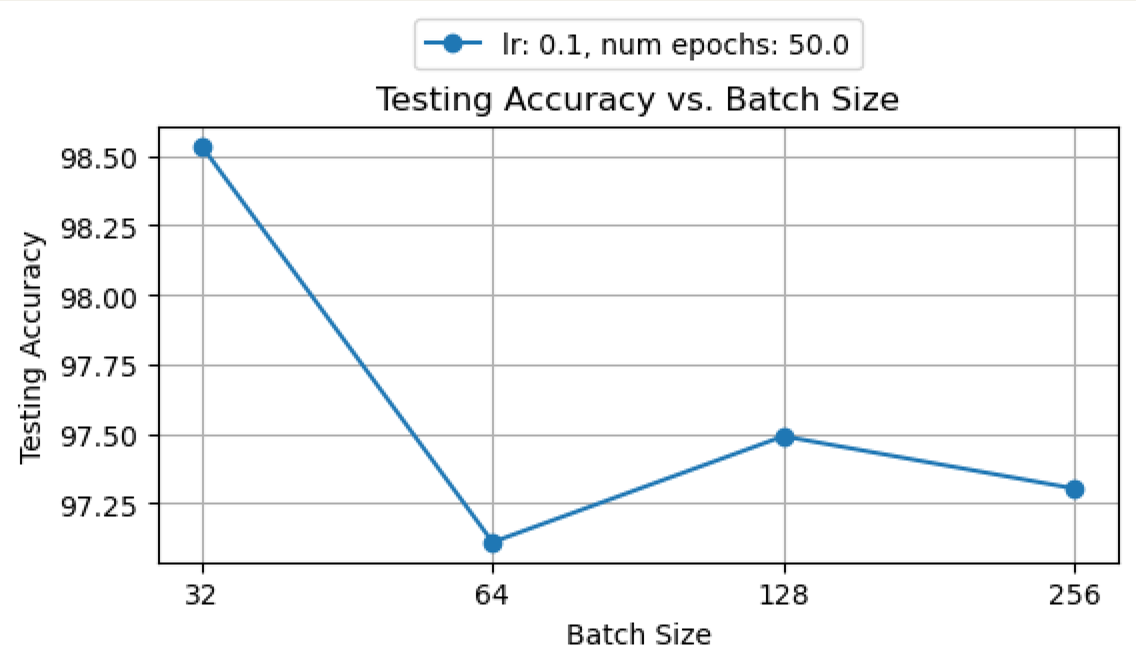
Hyperparameter Variation

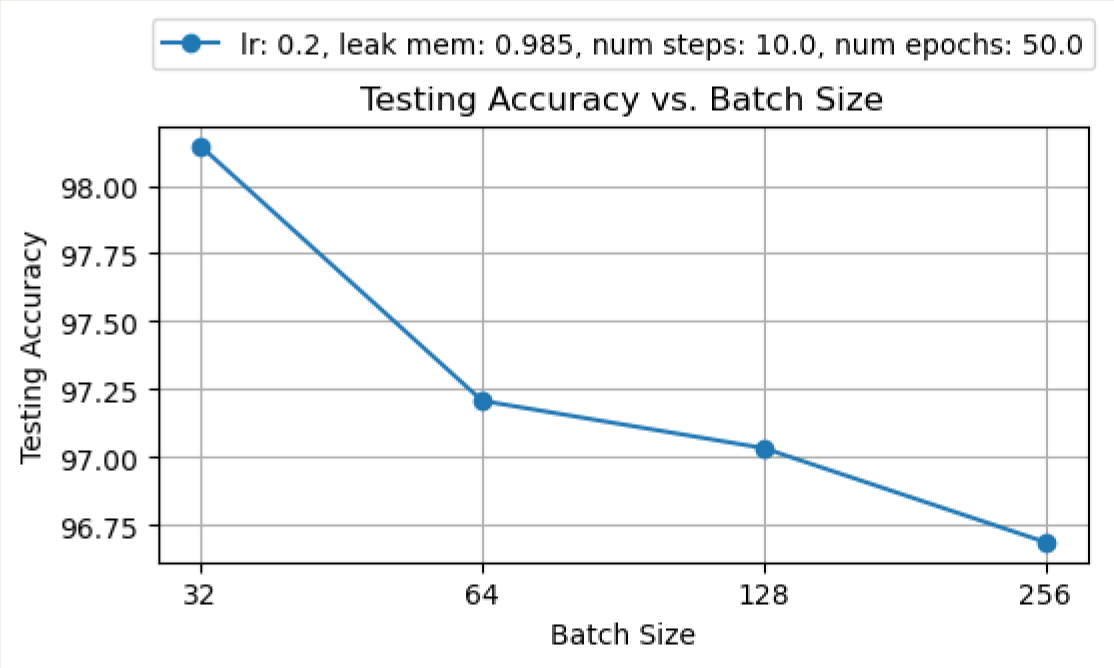
- Batch Size
- Epochs
- Timesteps
  - Steps in each forward pass of a neuron
- Leakage Voltage
  - Percentage of voltage retained between each step

Metrics

- Testing Accuracy
- Total FLOPs (Normalized)

Case Study – Testing Accuracy vs. Batch Size

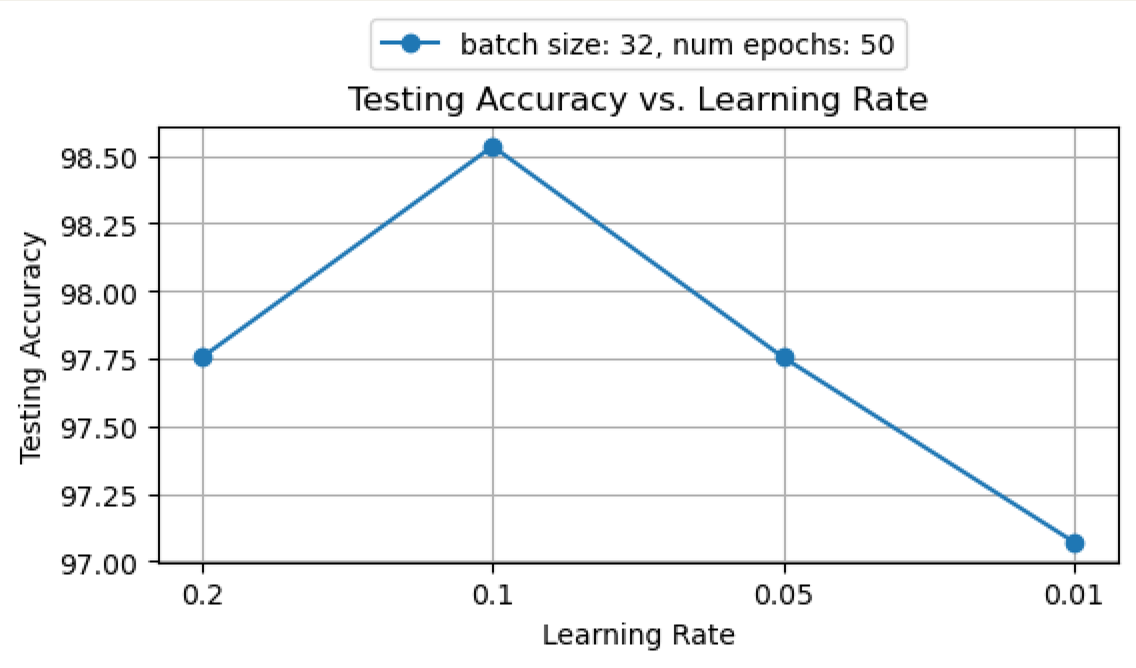


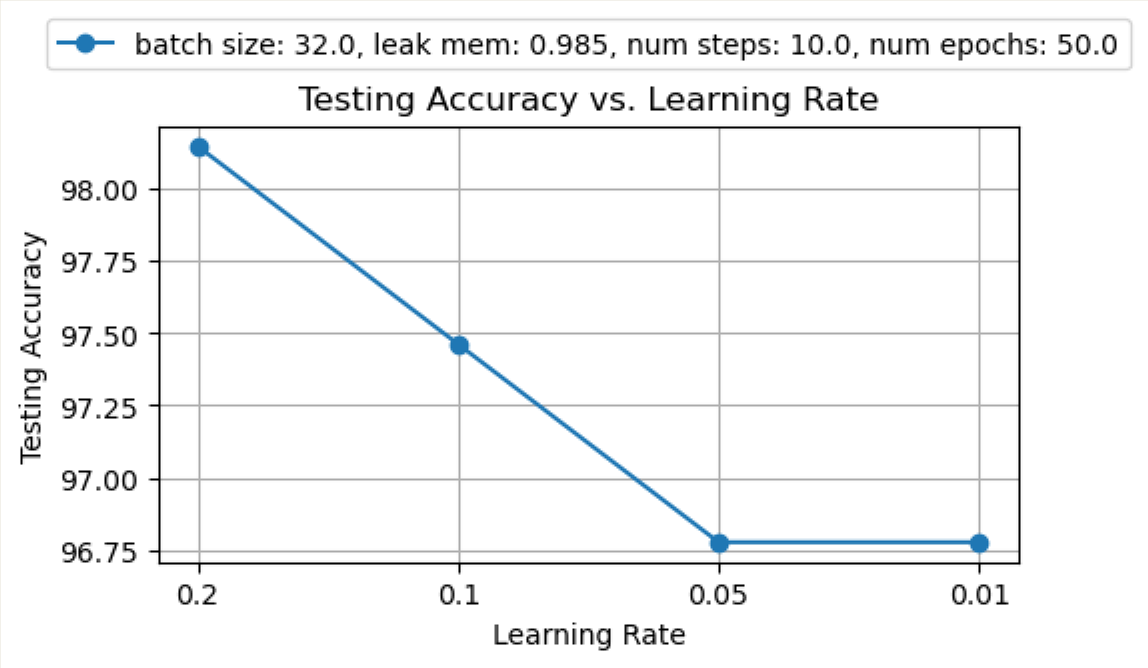


CNN Results:

CSNN Results:

Case Study – Testing Accuracy vs Learning Rate

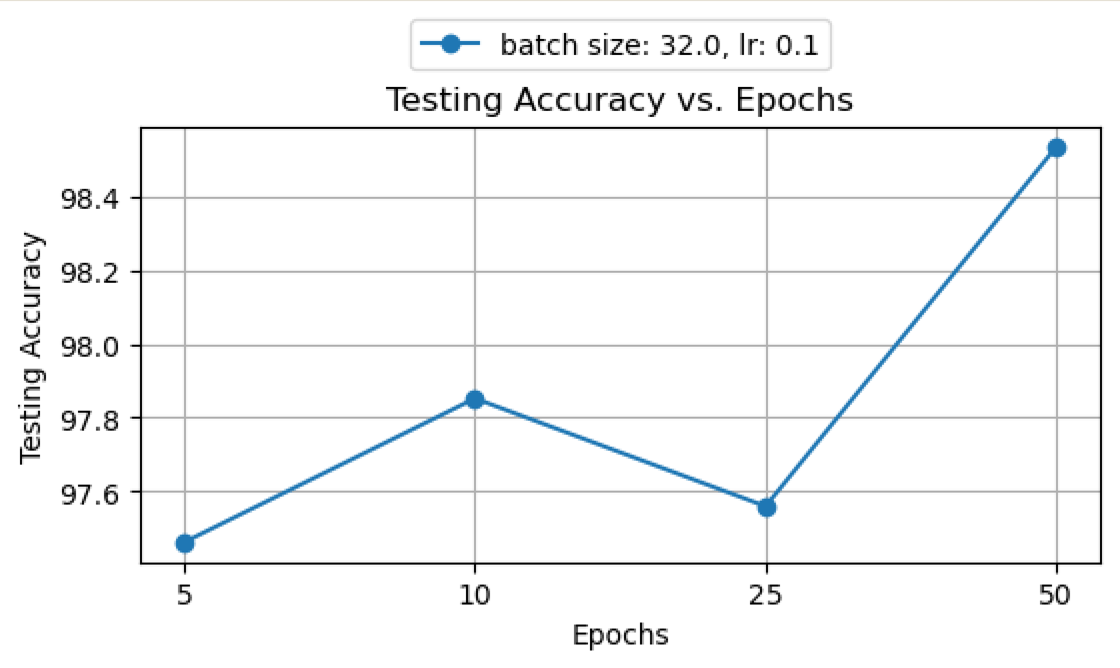


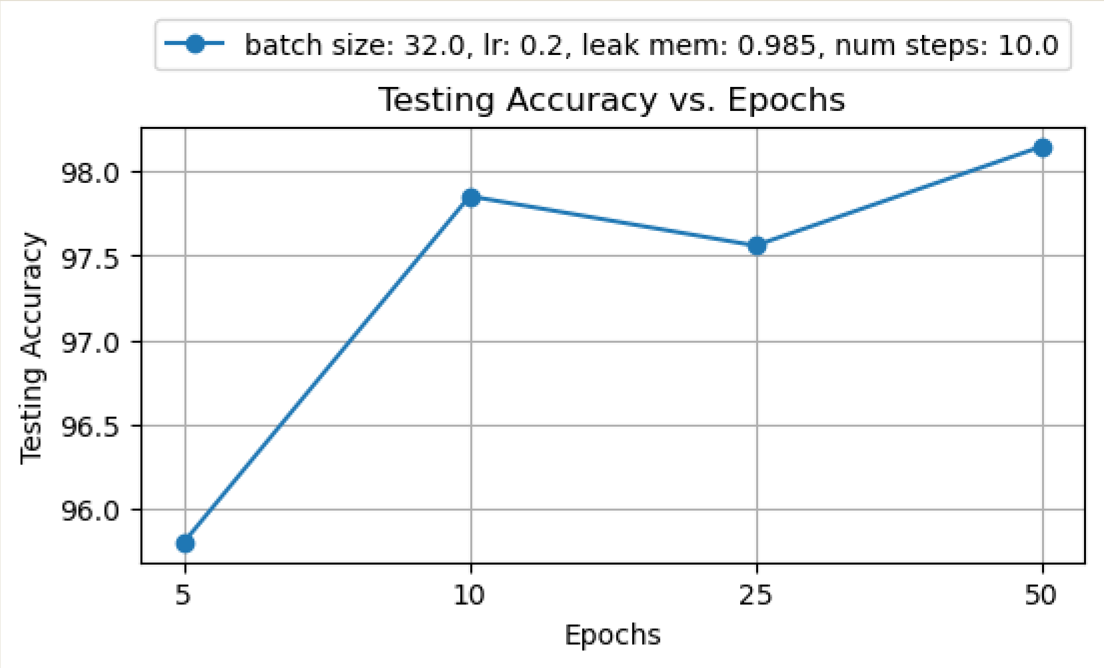


CNN Results:

CSNN Results:

Case Study – Testing Accuracy vs Epochs

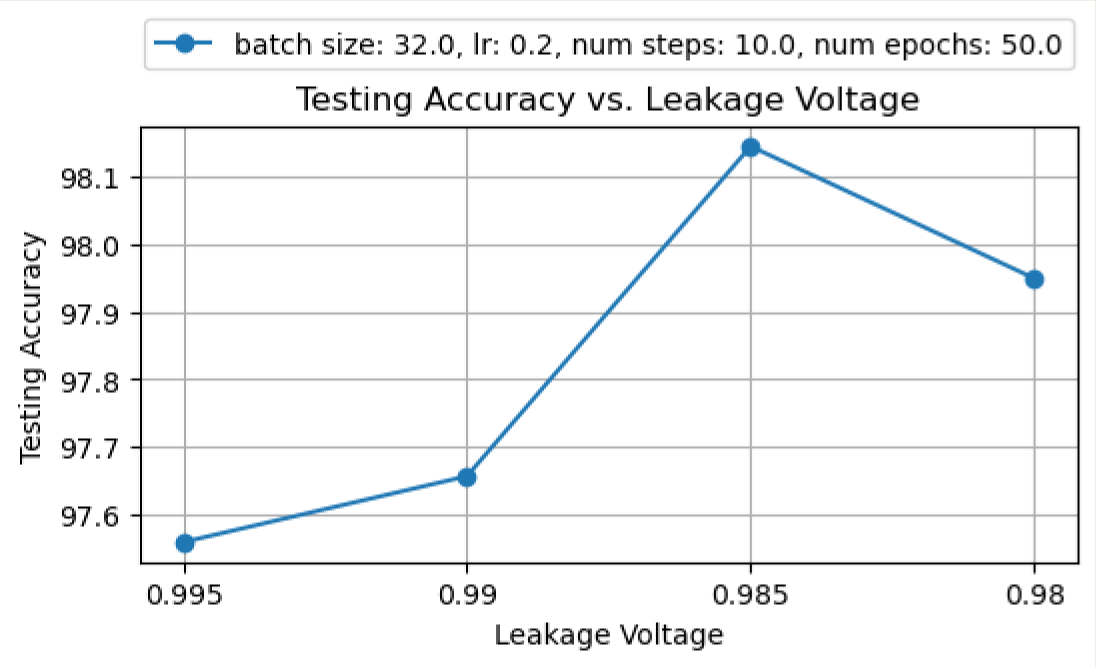


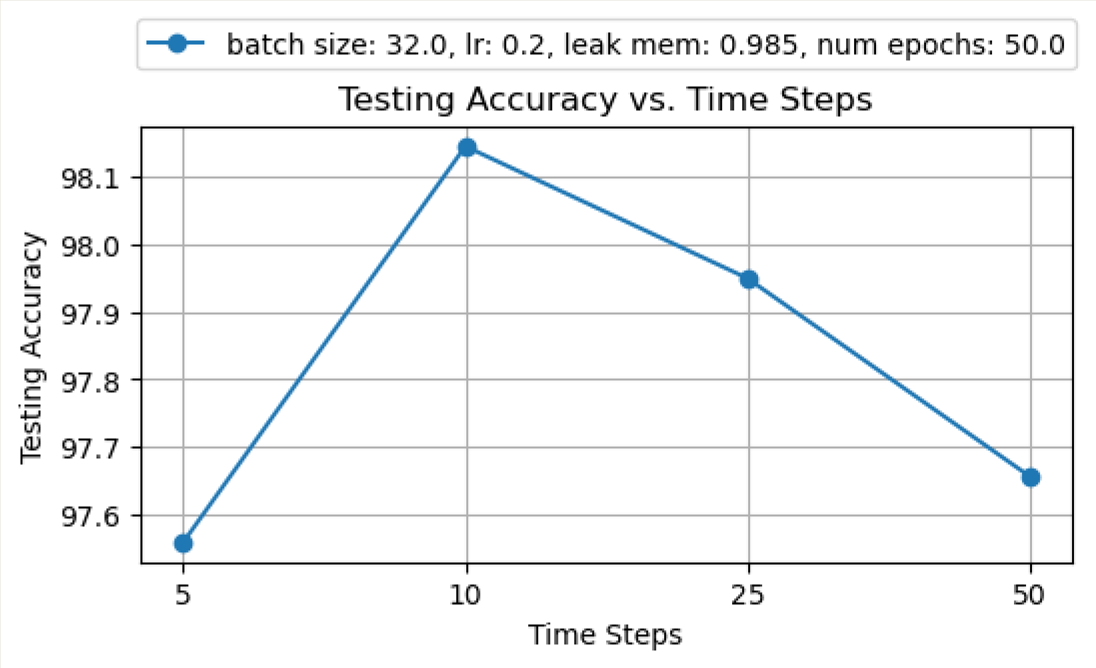


CNN Results:

CSNN Results:

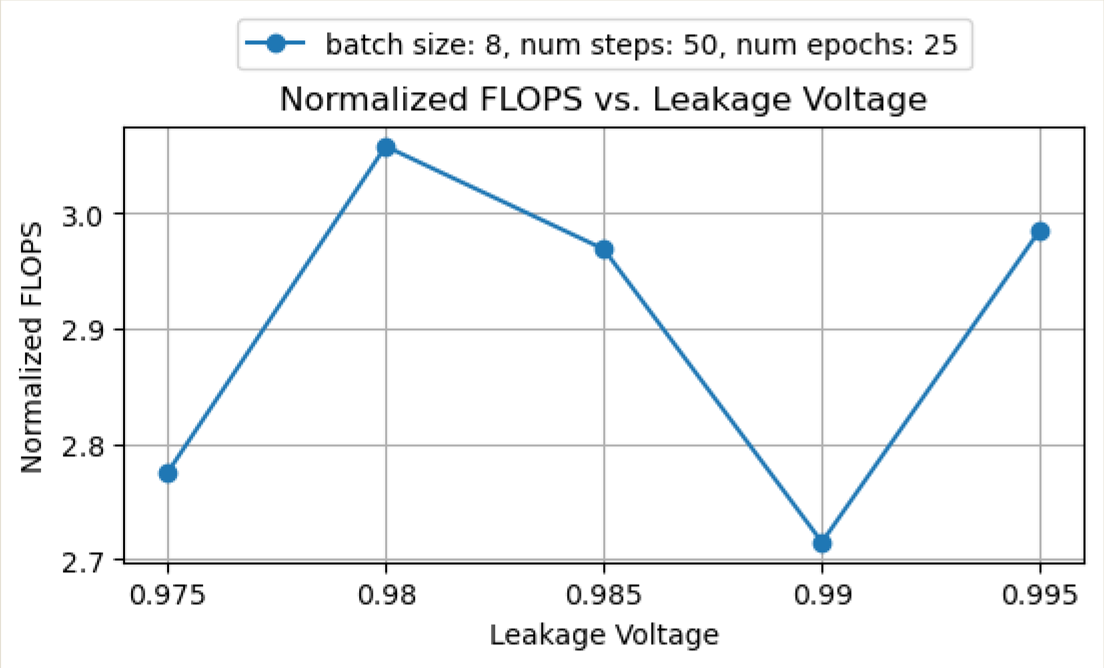
Case Study – Testing Accuracy vs Leakage Voltage / Time Steps

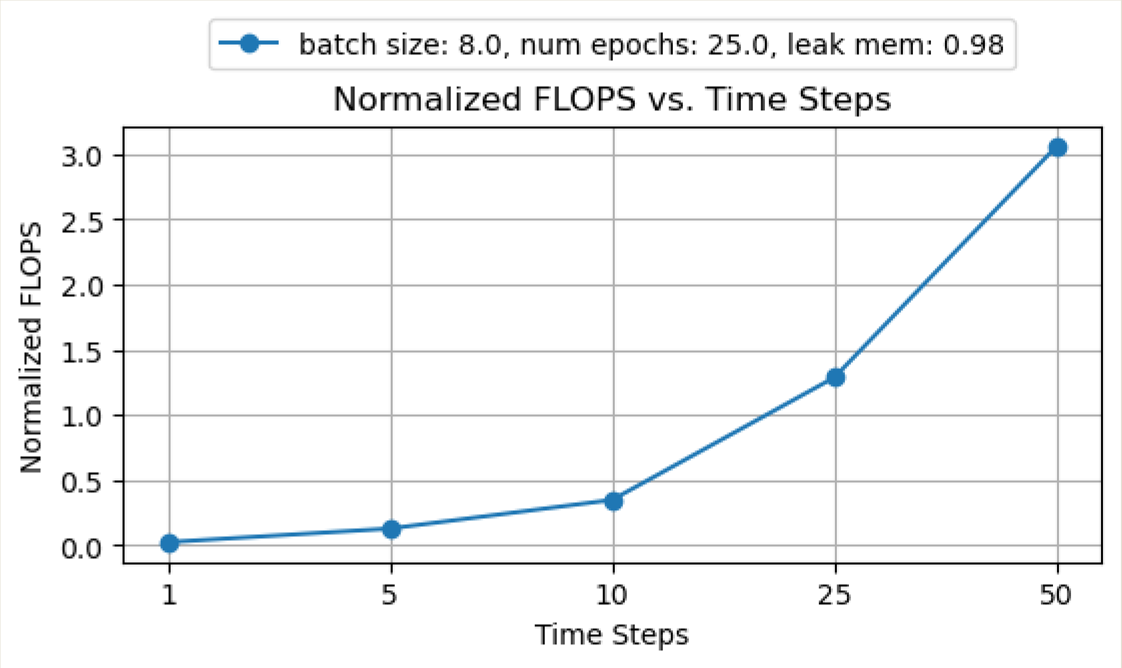




CSNN Results:

Case Study – Norm FLOPs vs Leakage Voltage / Time Steps





CSNN Results:

Model	CNN	SNN
Batch Size	32	32
Learning Rate	0.1	0.2
Epochs	50	50
Time Steps	N/A	10
Leakage Voltage	N/A	0.985
Training Loss	0.00098	0.00371
Testing Accuracy	98.535	98.145
Normalized Inference FLOPs	1x	0.428x

Evaluation

- Training Loss
- Classification Testing Accuracy
- Floating Point Operations comparison

Takeaway

- CSNN Model classifies with similar accuracy
- Computes 42.8% flops compared to CNN
- Ignores multiplier, majority of computational energy
- Achieves Significant decrease in energy usage.

QUESTIONS?