Scilab case study project on

Spiking Neural Networks for Image Classification in Scilab

Sumat Dhuwariya

Department of Computer Science, VIT Bhopal University

Image Processing

May 23, 2026

Abstract

This project presents handwritten digit classification using a single-layer Spiking Neural Network (SNN), implemented from scratch within Scilab. An 8x8 image dataset was used (3823 images), with an 80-20 train-test split. Static pixels were translated into a stochastic sequence of spikes using Poisson spike train encoding, and the Leaky Integrate-and-Fire (LIF) model to process images. A supervised Spike-Perceptron learning rule was implemented to train the network using a discrete target-rate firing mechanism. To optimize classification capacity and overall accuracy, we give penalty to non-target neurons, alongside momentum-assisted weight updates and stable firing rate to distinguish overlapping digit structures. As a result, the model achieved a final, reproducible classification accuracy of 91.90%.

1. Introduction

The case study aims to implement a single-layer SNN, using mathematical equations and discrete-time Euler integration in Scilab. We have implemented, LIF model (used in [2]), Poisson Spike Encoding, Membrane Dynamics (drawn from [3]) and momentum entirely using base Scilab mathematics. We adjust neuron's synaptic weights with time to prune weak connections inspired by Turrigiano [6]. However we avoided using certain topics which were either not required for our case study, or due to other constraints, such as python libraries and use of server quality GPUs. Primarily, we didn't use Spike-Timing-Dependent Plasticity (given in [2]) because we used a supervised teacher target (8 spikes). A target is set to fire certain amount of spikes to cross the threshold, this is decided by the maximum frequency and total simulation time. Moreover, we didn't used any continuous activation functions and our error correcting architecture is close to Ponulak & Kasiński [4] and Rumelhart et al. [5].

Metric / Feature	**Our Scilab Implementation**	**Diehl & Cook (Ref 2)**	**Ponulak & Kasiński (Ref 4)**
Learning Paradigm	Supervised (Spike-Perceptron)	Unsupervised (STDP)	Supervised (ReSuMe)
Accuracy	91.90% on 8x8 digits	95.00% on MNIST	Varies by sequence task
Network Size	64 Input, 10 Output	Up to 6,400 Neurons	Single/multi-neuron variants
F1-Score Evaluated	Yes (92.03% Macro)	No (Focus on global accuracy)	No (Focus on spike shifting)

Table 1: Results Comparison across References

2. Problem Statement

Spiking Neural Networks (SNNs) bridge Neurobiology and Artificial Intelligence. Consider a biological neuron that operates using electrochemical signals. The intensity and frequency determine its function and inherent tasks. Researchers across the world, from both domains, have worked hand in hand to study their respective backgrounds in an interdisciplinary format; consider the example of Artificial Neural Networks (ANNs), which helped us understand cognitive neuroscience. However, these neural networks, although mimicking the biological functions of neurons, have continuous activation functions instead of a discrete neural spikes. They require a significant amount of computational and hardware resources. We must understand the concept of 'Spike' inside SNNs, the third generation neural networks which solved the former problems, they are predictable, with identical pattern, shape, location and duration each time they occur, characterized by their firing time without continuous monitoring. In our study, we used the leaky-integrate model, which fires spikes when the membrane potential crosses a threshold. But there is much more to it, such as, Momentum, Depolarization, and Synaptic Weight matrix. All these concepts are explained in the next section in detail.

3. Basic concepts related to the topic

i. Leaky Integrate-and-Fire (LIF) Model: A biological neuronal model where a neuron steadily accumulates incoming electrical charge (integration) simultaneously losing charge over time (leaking). It triggers an output spike only when a voltage threshold is crossed.

ii. Spike-Perceptron Learning: A supervised machine learning algorithm that observes the final output error and updates the synaptic weights, forcing the network to align its spike frequency with the desired target label.

iii. Poisson Spike Encoding: Used to translate static image pixels into dynamic time-series data. If the random number is less than threshold probability, pixel will be converted into spike. The probability of a pixel firing at any given millisecond is directly proportional to its normalized brightness:

$$\mathrm{P(spike)} = \frac{I_{pixel} \times F_{max}}{1000} \tag{1}$$

iv. LIF Membrane Dynamics: The discrete time Euler integration was used to simulate the physical change in a neuron's membrane voltage (*mem*) based on injected current (I_{inj}), resting potential (V_{rest}), and the membrane time constant (τ_m):

$$mem[t+1] = V_{rest} + \beta_{decay} \cdot \left(mem[t] - V_{rest}\right) + \frac{cur_{\mathrm{in}}[t] \cdot R_m \cdot dt}{\tau_m} \tag{2}$$

v. Depolarization and Hyperpolarization: Depolarization occurs when a neuron's membrane potential becomes more positive, reaching closer to the threshold required to fire a spike. Conversely, hyperpolarization makes the internal voltage more negative, making it more difficult to trigger a firing event.

vi. Spike-Perceptron Weight Update: Updating the weights using learning rate η, and error calculated by difference of desired and output spikes. We are multiplying with

Error Transpose (E^T), to update all 640 connections (64 inputs × 10 outputs) in one single matrix calculation:

$$\Delta W = \eta(\text{Inputs} \times E^T) \quad (3)$$

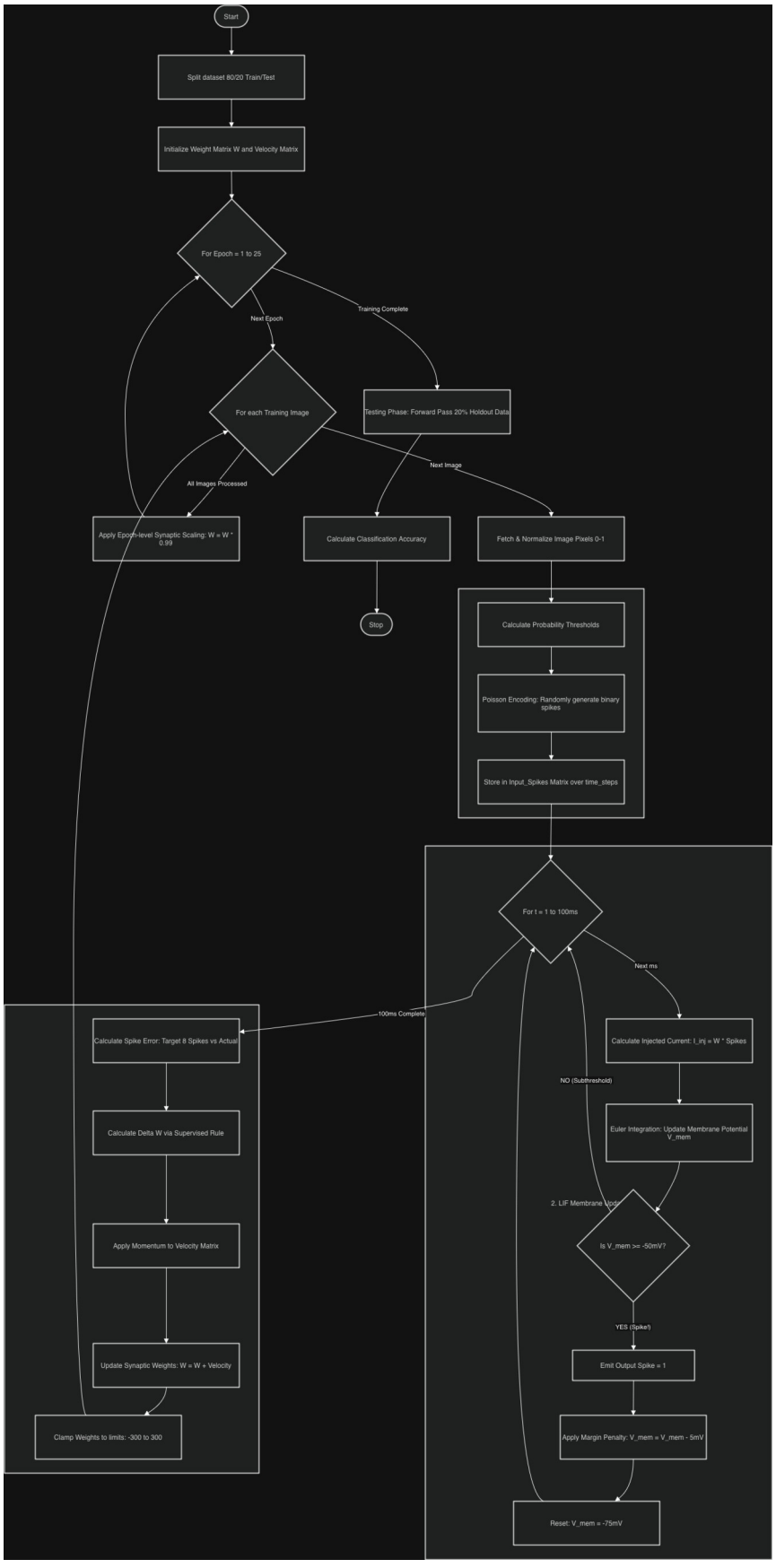
While running the training loop we made sure to initialize it within the range of [-10, 10] to dynamically learn and update itself. The weights had a range of [-300, 300] for entire learning period.

vii. Momentum Optimization: An acceleration mechanism used to prevent the network from stalling in local mathematical minima during training. It accumulates a friction-adjusted fraction of past weight updates (velocity, V_w) to give the algorithm mathematical "momentum" through flat gradient spaces:

$$V_w = (\beta \times V_w) + \Delta W \quad (4)$$

ix. L2 Regularization: A mechanism that universally decays synaptic weights by 1% (retention factor 0.99) at the conclusion of each training epoch. This process suppresses weak, noisy synaptic connections and prevents overfitting.

4. Flowchart


Start
Split dataset 80/20 Train/Test
Initialize Weight Matrix W and Velocity Matrix
For Epoch = 1 to 25
Next Epoch
Training Complete
For each Training Image
Testing Phase: Forward Pass 20% Holdout Data
Next Image
All Images Processed
Apply Epoch-level Synaptic Scaling: W = W * 0.99
Calculate Classification Accuracy
Fetch & Normalize Image Pixels 0-1
Stop
Calculate Probability Thresholds
Poisson Encoding: Randomly generate binary spikes
Store in Input_Spikes Matrix over time_steps
For t = 1 to 100ms
Next ms
100ms Complete
Calculate Injected Current: I_inj = W * Spikes
NO (Subthreshold)
Euler Integration: Update Membrane Potential V_mem
2. LIF Membrane Updat
Is V_mem >= -50mV?
YES (Spike!)
Emit Output Spike = 1
Apply Margin Penalty: V_mem = V_mem - 5mV
Reset: V_mem = -75mV
Calculate Spike Error: Target 8 Spikes vs Actual
Calculate Delta W via Supervised Rule
Apply Momentum to Velocity Matrix
Update Synaptic Weights: W = W + Velocity
Clamp Weights to limits: -300 to 300

5. Software/Hardware used

Operating System: macOS 26.3.1 (25D2128)

Scilab Version: 2026.0.1

Toolboxes Used: Base Scilab (No additional toolboxes required)

Hardware: Apple Silicon M3 processor, 8GB RAM

6. Procedure of execution

- Install Scilab software (version 2026.0.1 or equivalent). Ensure the dataset file (digits_dataset.csv) is in your current directory.
- Open and execute main.sce. A graphical wait bar will track the completion of the 25 training epochs and subsequent test data predictions (trained_brain_supervised.sod (weights) and test_results.sod (raw prediction data) will be saved to the directory), wait for some time 30-45 seconds approximately.
- Execute singleTest.sce to test a single row of dataset. To generate and export three visual plots to the working directory.
- Execute diagnostic_visualizer.sce to render the 8x8 receptive fields for all 10 output neurons.
- To generate the final academic metrics, execute statistical_analysis.sce. This script imports the previously saved test_results.sod and calculates the confusion matrix.

NOTE: The Leaky Integrate-and-Fire (LIF) neuronal physics, Poisson spike generation, Spike-Perceptron learning rule executed in this project are implemented mathematically from scratch using Scilab only, without relying on any external machine learning tensor libraries.

Pipeline:

- **Reproducibility:** We start by setting rand("seed", 42) ensuring that every time the code is run, we get the same results.
- **Membrane Dynamics (tau_m):** This is membrane constant time, after which leaking will start, to ensure we give enough time to neuron for guessing probable output.
- **Voltage Thresholds:** We set V_rest = -70.0 and V_th = -50.0. Because a jump of 20mV is required for depolarization (as per David A. McCormick in his book for Fundamental

Neuroscience Chapter 5, under the section: Na+, K+, and Cl− Contribute to the Determination of the Resting Membrane Potential).

- **Electrical Resistance:** We set R_m = 1.0, according to Ohm's Law, $V=IR$, for easier calculations between V and I. If we will increase the resistance the potential will increase, but the current is sensitive to such changes, so we update the weights to control potential of spikes.
- **Simulation Time:** We run each image for 100ms because taking too short or long will lead to false prediction and overfitting.
- **Loading:** We import the CSV file for dataset, and normalise the data in [0,1] by dividing with the maximum value.
- **Poisson Encoding:** We multiply the image by the max spike probability to create probability thresholds. Then, throw a random dart, if the random number is less then threshold probability, pixel will be converted into spike.
- **Spikes to Current:** The binary input spikes (*input_spikes*) are multiplied with synaptic weight matrix (W) to convert them into a numerical value representing Injected Current (I_{inj}). If a connection has a high weight, a spike creates a "strong" current; if it has a low weight, the spike creates a "weak" current.
- **LIF Integration:** Start with a vertical list with *V_rest* values. To inject current, multiply with transpose of weight to create a list of 10 values. These values then are sent to Euler's Integration Formula to continuously solve the membrane leakage differential equation to calculate the change in potential over time.
- When an output neuron spikes, it immediately applies a negative voltage penalty (e.g., -5.0 mV) to all neighboring neurons in the same layer. This helps us to differentiate between the confusing digits such as 8 and 9. Eventually, the potential of output spikes is reset to much lower state (-75mV) as cooling period.
- **Refractory Period:** Reseting the fired spikes for cool down" by dropping it to *V_reset* (-75mV).
- **Momentum:** Initialize a *Velocity_W* array so that we safely converge the error. This prevents the learning algorithm from getting stuck.

- **Learning Rate Decay:** An exponential curve for "decreasing learning rate" over time, allowing the network to make big adjustments early on and fine-tune later.
- **Weight Boundaries:** Dynamically updating the max min values, and clamping the weights so they never cross absolute limit of [-300, 300]. This range is calculated by experiments, Consider the equation 2 (*cur_in * R_m * dt / tau_m)* if we put the maximum values, *cur_in* = 300, *R_m* = 1, *dt* = 1, and *tau_m* = 20, we will get maximum jump of 15 mV which is less than 20mV. 15 mV will give the weight to sensitive pixels, also will not cross the threshold directly to inject more noise.
- **L2 Regularization:** At the end of every epoch, multiply the entire weight by 0.99 to decrease the weight by 1% to reduce overfitting.
- **Testing:** Once the 25 epochs are finished, we run the exact same ‘snn_forward_pass’ function, but we only require output spikes. Find the neuron that spiked the most, and predict. Finally, execute save("test_results.sod", "all_true_labels", "all_pred_labels").

7. Result

The model processed an 80-20 split of 3,823 images with accuracy of **91.90%.** To ensure the model did not over-predict a single class, statistical metrics were generated. The proximity of the Macro F1-Score (92.03%) to the Global Accuracy (91.90%) proves the network learned highly balanced geometric representations across all ten-digit classes. Macro Precision is 92.27% and Macro Recall is 92.17% .

Digit	Precision	Recall	Specificity	F1-Score
0	1	1	1	1
1	0.77	0.93	0.97	0.85
2	0.92	0.92	0.99	0.92
3	0.91	0.98	0.99	0.94
4	0.95	0.92	0.99	0.93
5	0.94	0.98	0.99	0.96
6	0.94	0.94	0.99	0.94
7	0.97	0.97	1	0.97
8	0.87	0.76	0.99	0.81
9	0.95	0.81	1	0.87

Table 2: Statistical Outputs

Some Graphs:

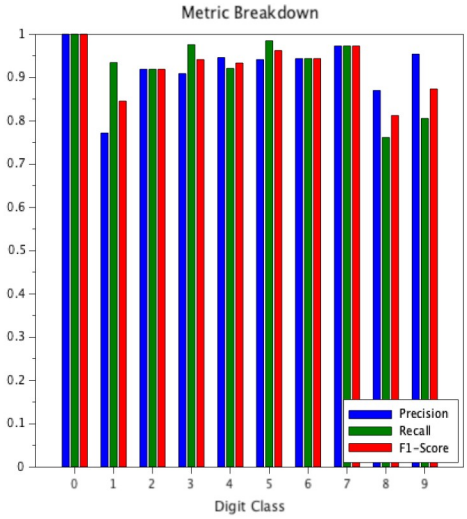


Figure 1: Per-Class Metric Breakdown comparing Precision, Recall, and F1-Scores across Digits 0-9.

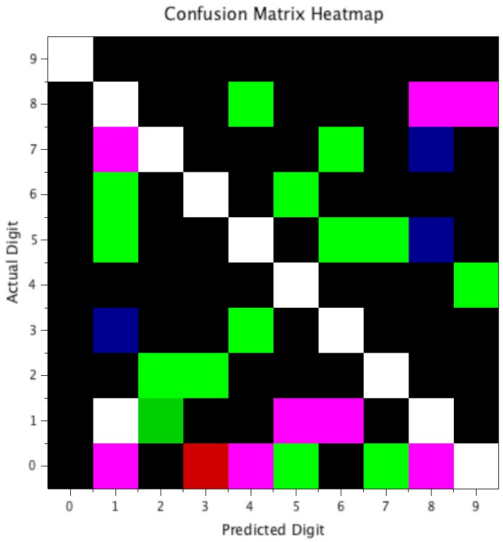


Figure 2: White squares on diagonal represents correct labels. The colored boxes (like red, blue, or green) represent a mistake. The position of these colored squares tells exact confusion. For example, if a colored square in the row for '1' but in the column for '5', it means the model looked at '1' and incorrectly guessed '5'.

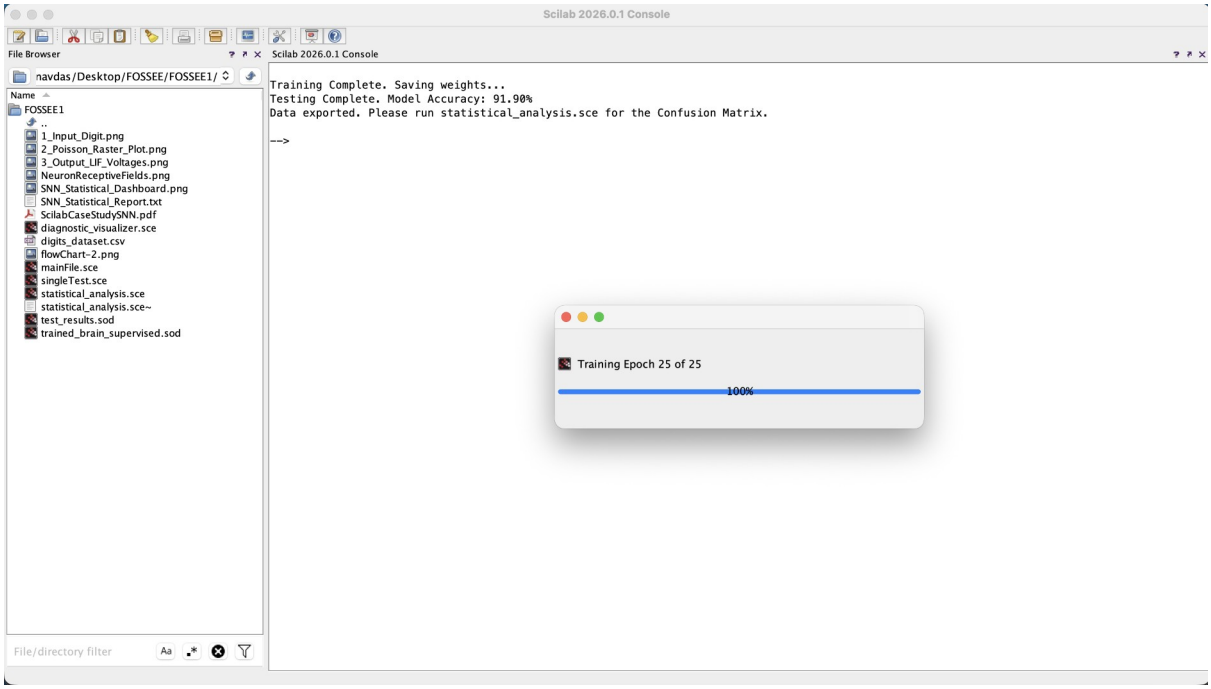


Figure 3: Screenshot of execution log in console

Graphs of Single Test:

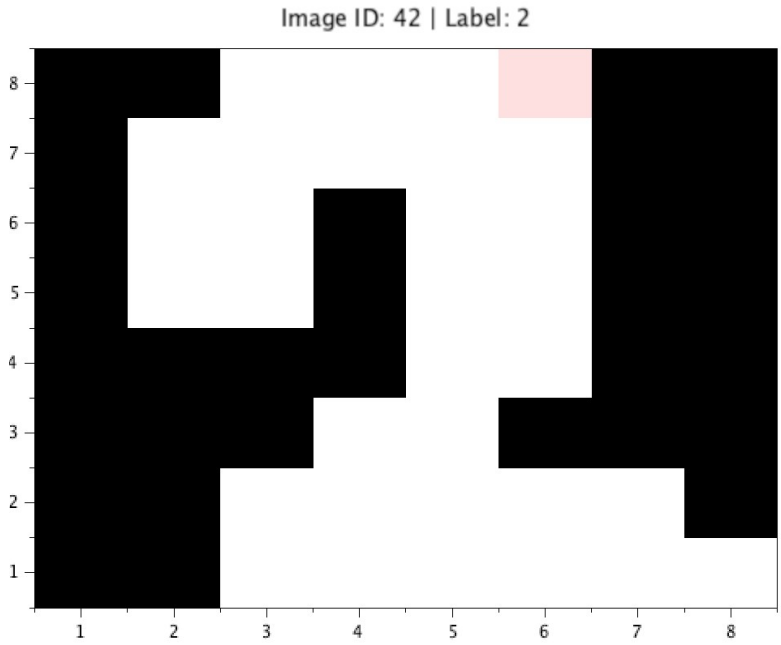


Figure 4: 2D matrix of dark and bright pixel intensities. The white pixels represent the hand-written digits.

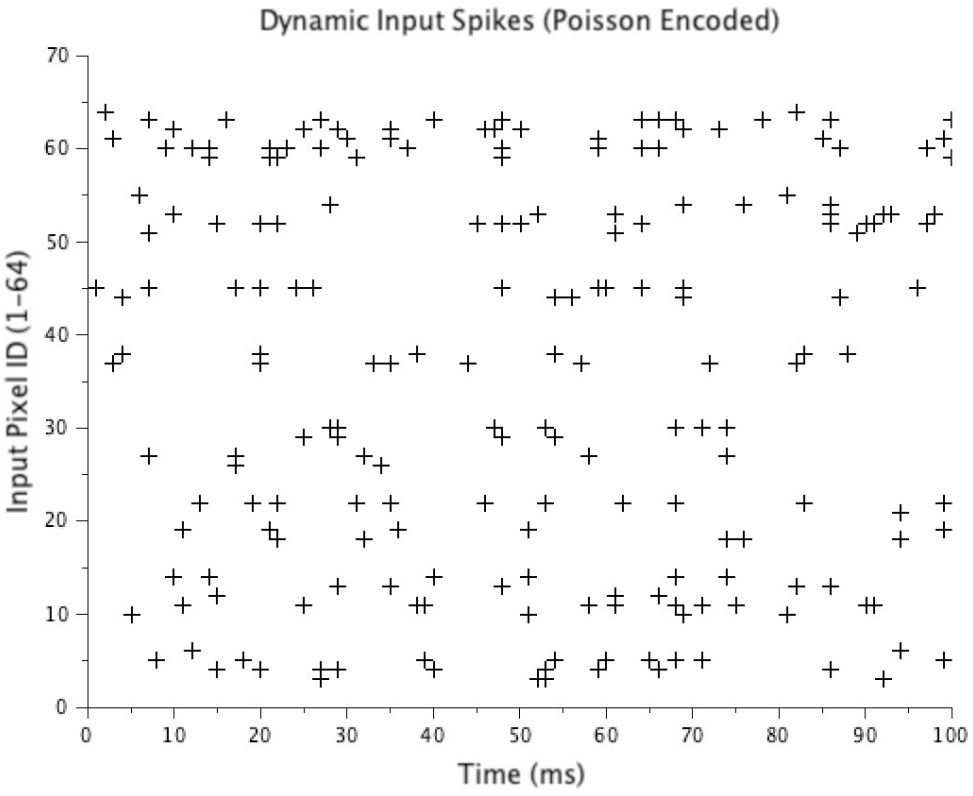


Figure 5: Raster plot of the rate of fire of input spikes

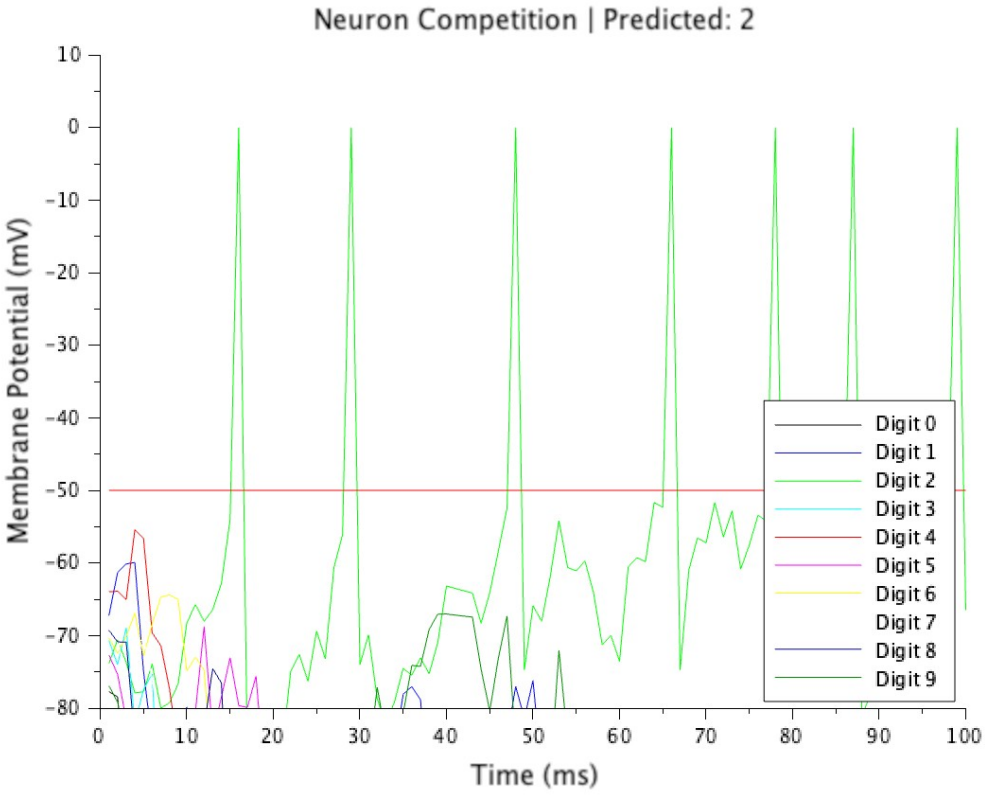


Figure 6: The test label ‘2’ is firing the most

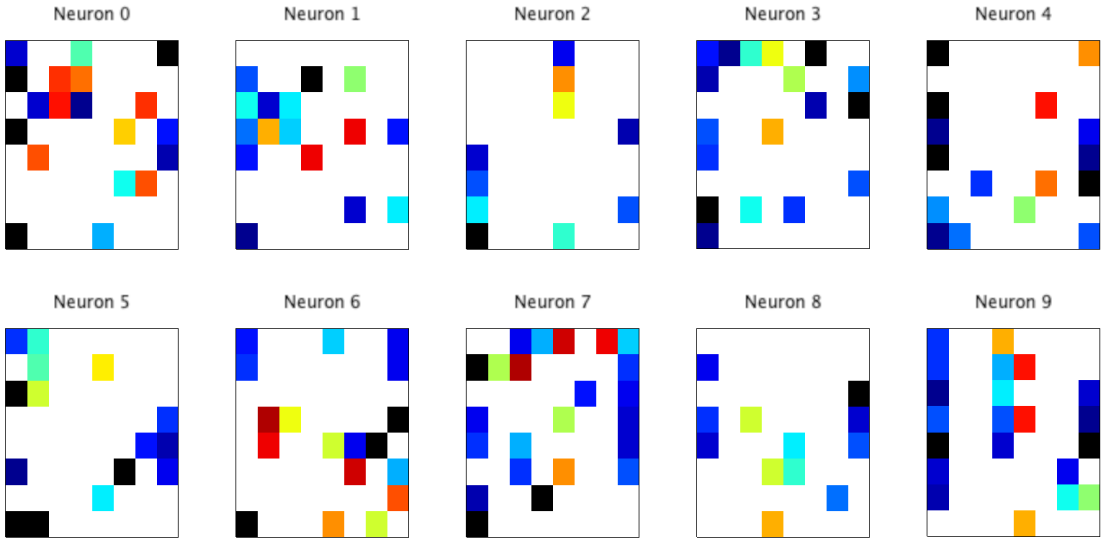


Figure 7: Hot colors (Red, Orange, Yellow) represent the "must-have" features. If input digit has a pixel in these locations, it pushes LIF neuron closer to threshold (−50mV) and triggers a spike. Cool colors (Blue, Black, Dark Blue) represent if an input digit has a pixel here, it subtracts from the neuron's membrane potential.

8. References

[1] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 7, pp. 1659-1671, 1997, doi: 10.1016/S0893-6080(97)00011-7.

[2] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, art. no. 99, 2015.

[3] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, U.K.: Cambridge University Press, 2002, doi: 10.1017/CBO9780511815706.

[4] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467-510, 2010.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533-536, 1986.

[6] G. G. Turrigiano, "The self-tuning neuron: synaptic scaling of excitatory synapses," *Cell*, vol. 135, no. 3, pp. 422-435, 2008.