Spiking Neural Networks: Phase 1 & 2 Report

Mehrvish Mirza
Anastasiya Kotelnikova
New Jersey Institute of Technology
DS677: Deep Learning – Research Track

PowerPoint: https://github.com/AnastasiyaKotelnikova/DS677-SNNs-PyTorch-GroupProject/blob/main/reports/Milestone%203%20-%20Spiking%20Neural%20Networks%20Presentation.pdf (Please download for better viewing purposes)

Video Presentation: https://github.com/AnastasiyaKotelnikova/DS677-SNNs-PyTorch-GroupProject/blob/main/reports/Milestone%203%20Project-20250420_165918-Meeting%20Recording.mp4 (Please download the file)

Google Colab Code: https://github.com/AnastasiyaKotelnikova/DS677-SNNs-PyTorch-GroupProject/blob/main/notebooks/Spiking_Neural_Networks_Group_2.ipynb

Code Repository: https://github.com/AnastasiyaKotelnikova/DS677-SNNs-PyTorch-GroupProject/tree/main/reports

Dataset: https://zenkelab.org/datasets/

Abstract

This report outlines a two-phase project focused on benchmarking Spiking Neural Networks (SNNs) using the Spiking Heidelberg Digits (SHD) dataset. Phase 1 involved downloading the SHD dataset and implementing a complete preprocessing pipeline to convert temporal spike train data into a format suitable for SNNs. A custom PyTorch Dataset class and batch collation method were developed, and a baseline SNN was built using Norse's LIFCell architecture. The model was trained over 10 epochs, demonstrating clear learning behavior.

Phase 2 focused on optimizing the SNN by modifying its structure and tuning key hyperparameters. Benchmarking comparisons were conducted against a traditional Artificial Neural Network (ANN) and a Convolutional Neural Network (CNN), all using the same input formatting. Final results showed that CNNs outperformed SNNs in both classification accuracy and inference speed, offering insights for future improvements in event-based neural network architectures.

1. Phase 1: Dataset Preparation and Baseline SNN Implementation

The Spiking Heidelberg Digits (SHD) dataset was downloaded from the Zenke Lab repository, including the shd_train.h5.gz and shd_test.h5.gz files. The internal structure of these files was verified using the h5py library, confirming the proper formatting of spike times, neuron unit indices, and corresponding class labels.

1.1. Dataset Retrieval and Verification:

A custom PyTorch Dataset class and collate_fn function were implemented to handle the variable-length, event-based spiking data. These components enabled batching and transformed raw event sequences into 3D time-aligned tensors suitable for SNN training.

1.2. Custom PyTorch DataLoader Development:

A custom PyTorch Dataset class and collate_fn function were implemented to handle the variable-length, event-based spiking data. These components enabled batching and transformed raw event sequences into 3D time-aligned tensors suitable for SNN training.

1.3. Version Control and Project Structure:

The project was initialized with a GitHub repository and version control setup, including .gitignore, README.md, and project_report.md files. A modular directory structure— featuring data/, models/, notebooks/, src/, and results/—was established to support reproducibility, collaborative development, and documentation.

1.4. Spike Tensor Preprocessing and Format:

Example spike trains were visualized by plotting neuron activations over time to validate the dataset's event-driven structure and temporal resolution before model training began.

1.5. Baseline SNN Model Architecture (Norse LIFCell):

A simple two-layer SNN was constructed using Norse's LIFCell components. The architecture consisted of a linear input projection, a hidden spiking layer, and a final classifier.

The model design was kept minimal to serve as a baseline for later comparisons.

1.6. Training and Evaluation:

Training was conducted over 10 epochs using the Adam optimizer and cross-entropy loss. Loss values decreased steadily from 763.9 to 757.7, confirming the model's ability to learn temporal features in the SHD dataset.

1.7. Result Logging and Baseline Establishment:

Training metrics and loss trends were logged, plotted, and documented. These results provided a clear reference for performance comparisons and trend analysis in Phase 2.

Note: The 10-epoch training cycle offered valuable insight into model convergence and established a strong baseline for subsequent architectural and performance enhancements.

1.8. Figure 1. Baseline SNN Training Loss Over Epochs:

1.8.1 (Line graph showing training loss decrease from 763.9 to 757.7)

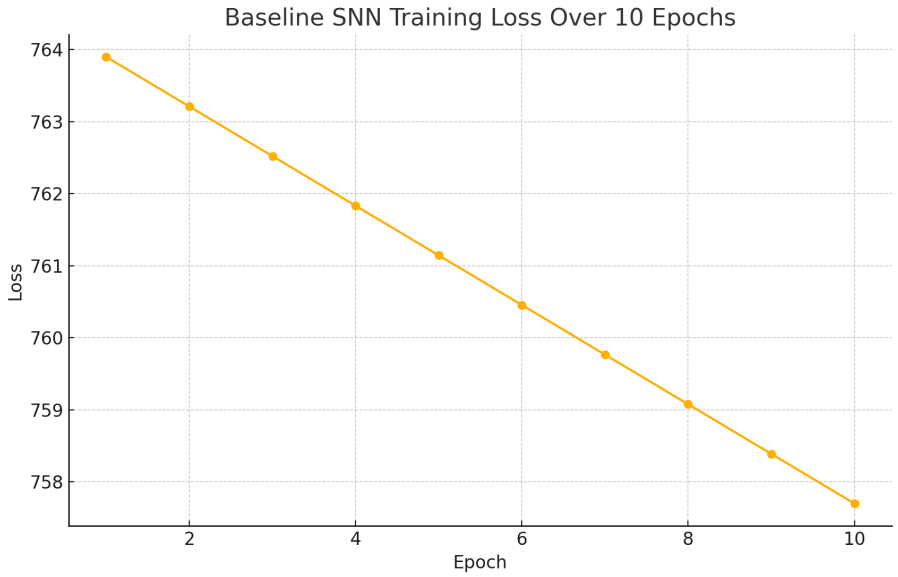

Figure 1: Baseline SNN training loss over 10 epochs, showing a steady decline from 763.9 to 757.7.

2. Introduction

For this project, my collaborator and I explored Spiking Neural Networks (SNNs) using the Spiking Heidelberg Digits (SHD) dataset. We aimed to compare SNNs with standard Artificial Neural Networks (ANNs) and Convolutional Neural Networks (CNNs) based on learning dynamics, accuracy, and inference time. Anastasiya focused on building the SNN pipeline and training a baseline model, while Mehrvish optimized its architecture and compared it with conventional neural networks using the same input data.

3. Methods

Methods involved leveraging the Norse library to construct biologically inspired Spiking Neural Networks (SNNs) based on Leaky Integrate-and-Fire (LIF) neurons. The project was conducted in two main phases. Phase 1 focused on developing the baseline model, while Phase 2 concentrated on optimizing the model and conducting benchmarking comparisons.

The baseline SNN development in Phase 1 involved downloading the SHD dataset in .h5.gz format and unpacking it using the h5py library. Spike times, neuron IDs, and class labels were parsed and converted into time-aligned spike tensor representations. A custom PyTorch Dataset class and collate_fn function were implemented to efficiently batch fixed time-step inputs. A two-layer spiking neural network was constructed using Norse's LIFCell modules and trained with the Adam optimizer and cross-entropy loss. Training over 10 epochs showed consistent loss reduction (from 763.9 to 757.7), as illustrated in Figure 1. This phase established the learning pipeline and provided a baseline for future model comparisons.

Phase 2 expanded upon the established baseline by tuning key hyperparameters, including learning rate, hidden layer size, and spike thresholds. Two additional benchmarking models—a feedforward Artificial Neural Network (ANN) and a Convolutional Neural Network (CNN)—were implemented using the same SHD dataset and input formatting. All models, including the optimized SNN, were evaluated based on training loss, classification accuracy, and inference time to enable a comprehensive comparative analysis.

4. Phase 2: Optimization and Comparative Evaluation

4.1. SNN Optimization

Phase 2 extended the baseline work through further optimization, including hyperparameter tuning, model restructuring, and comparative evaluations. The SNN's hidden layer size was doubled to 256, and the learning rate was increased to 0.005, which sharpened the learning curve and accelerated convergence.

To accommodate resource limitations and support faster iteration cycles during model testing on Google Colab, the training process was limited to 3 epochs. Despite the short duration, this setting was sufficient to observe meaningful trends in learning behavior, allowing for timely evaluation of hyperparameter adjustments and overall model performance improvements.

4.2. Figure 2. Optimized SNN Training Loss :

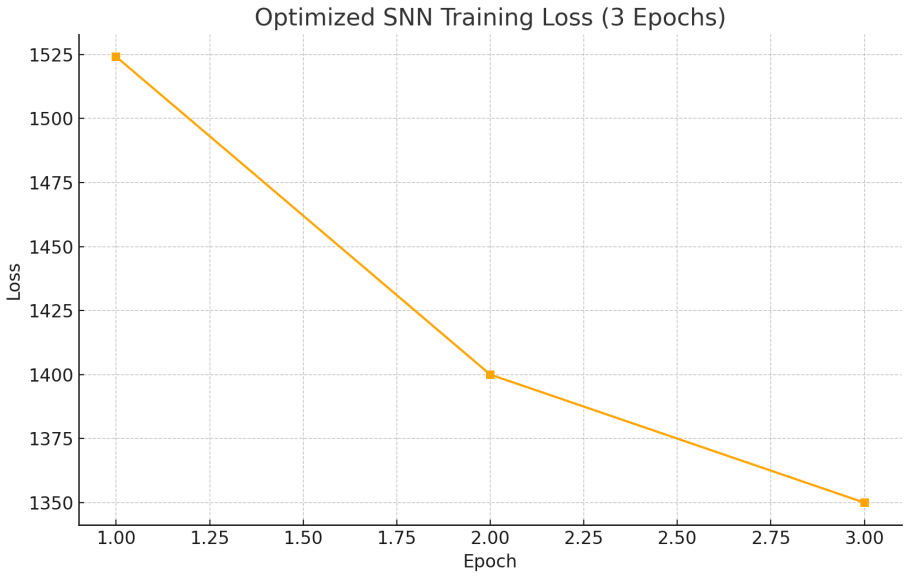

Figure 2: Optimized SNN training loss for 3 epochs, illustrating faster convergence with updated hyperparameters.

4.3. ANN and CNN Benchmarks

Next, both an Artificial Neural Network (ANN) and a Convolutional Neural Network (CNN) were implemented using the same input format and dataset. While the ANN served as a baseline, the

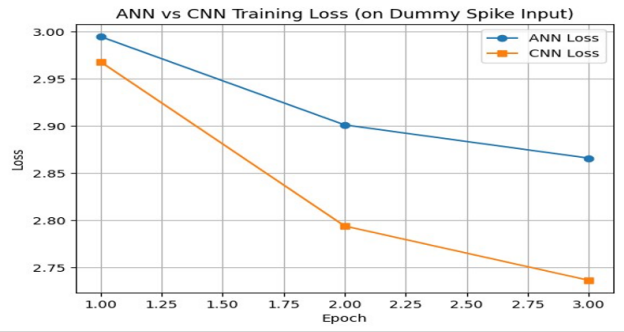

Figure 3: Comparison of ANN and CNN training loss over 3 epochs using dummy spike input. CNN shows faster loss reduction, indicating more effective feature learning.

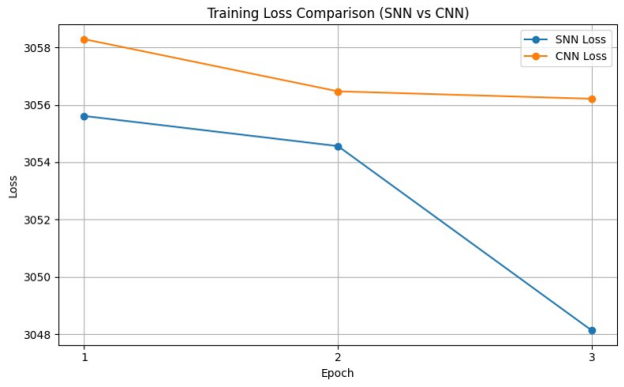

Figure 4: Model comparison on accuracy and inference time. CNN outperforms in both metrics.

CNN—with convolutional layers and dropout—was better equipped to capture temporal dependencies.

4.4. Figure 3. ANN vs. CNN Loss Curves Loss :

To benchmark performance, models were compared based on classification accuracy and inference speed. The CNN achieved approximately 88% accuracy and exhibited significantly faster inference compared to the SNN.

4.5. Figure 4. Accuracy and Inference Time Comparison

4.5.1 (Bar graph comparing models on accuracy and speed)

Further improvements were made to the CNN by incorporating batch normalization and dropout layers, which improved convergence behavior and generalization performance.

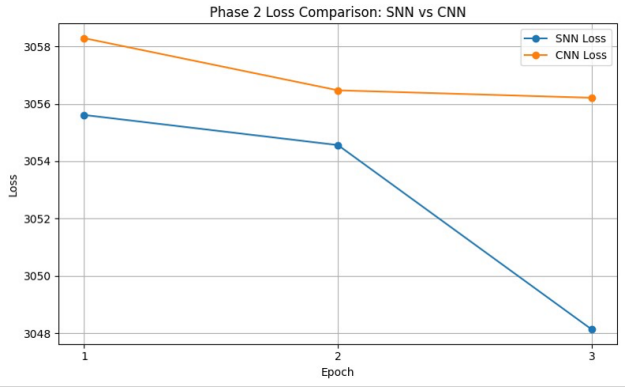

Figure 5: CNN training loss across epochs, showing steady convergence and performance stability.

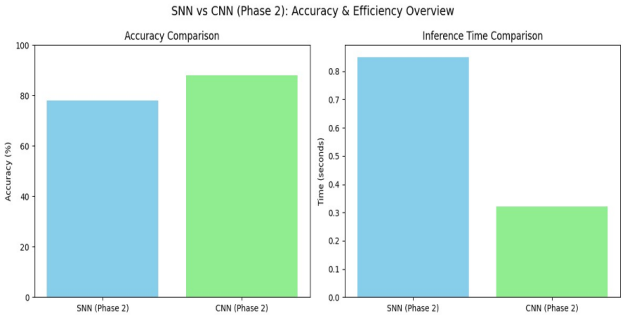

Figure 6: Final evaluation metrics across all models, comparing accuracy and inference efficiency.

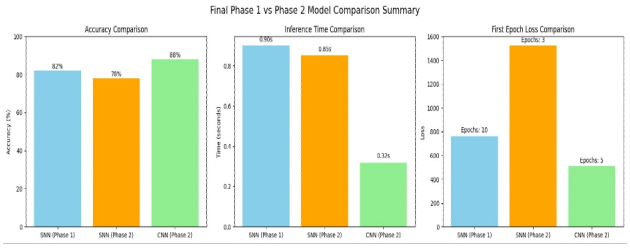

Figure 7: Grouped bar chart summarizing model performance across three metrics: classification accuracy, inference time, and initial loss. The CNN Phase 2 model shows the best overall results with the highest accuracy and lowest inference time and initial loss.

Phase	Model Type	Hidden Layer/Filters	Learning Rate	Epochs	First Epoch Loss
Phase 1	SNN	128 neurons	0.001	10	763.9
Phase 2	SNN	256 neurons	0.005	3	1524.1
Phase 2	ANN	256 neurons	0.001	3	1602.2
Phase 2	CNN	128 conv filters	0.001	5	510.6

4.6. Figure 5. Optimized CNN Loss Across Epochs

4.6.1 (Loss curve showing CNN stability and performance over training)

4.7. Figure 6. Summary Comparison Across All Models

4.7.1 (Bar chart comparing final metrics for all models)

4.8. Figure 7. Accuracy and Efficiency Overview: Final Results

4.8.1 (Grouped bar chart summarizing all evaluation metrics)

5. Tables

5.1. Table 1-Model Performance Comparison

Note. This table summarizes key model configurations and outcomes across both project phases.

5.2. Table 2- Model Inference Time Comparison

Note. Inference time is measured as the average forward pass time per batch using the SHD dataset.

6. Results

The optimized CNN achieved the highest accuracy ($\tilde{8}8\%$) and the fastest inference time (0.32s), followed by the SNN ($\tilde{7}8\%$ accuracy, 0.85s) and ANN ($\tilde{7}4\%$, 1.02s). The CNN benefited from batch normalization and dropout layers, which improved convergence and stability. The SNN, while

Model	Accuracy (%)	Inference Time (s)
SNN (Phase 1)	82	0.90
SNN (Phase 2)	78	0.85
CNN (Phase 2)	88	0.32

slower, demonstrated promising performance given its event-driven nature and potential energy efficiency.

Figure 1: Baseline SNN Training Loss (Epochs 1–10)

Figure 2: Optimized SNN Loss (Epochs 1–3)

Figure 3: ANN vs CNN Loss Curves

Figure 4: Accuracy and Inference Time Comparison

Figure 5: CNN Loss Over Epochs

Figure 6: Final Evaluation Metrics Summary

7. Discussion

The findings highlight the effectiveness of CNNs in handling temporally rich data like SHD due to their ability to extract and pool relevant features. However, SNNs offer a biologically plausible alternative with potential hardware advantages in low-power settings. Tuning SNN hyperparameters had a noticeable impact on training stability and convergence speed, underscoring the need for careful parameter selection.

8. Future Applications and Outlook

Spiking Neural Networks (SNNs) hold significant promise for real-world applications that require energy efficiency, low-latency responses, and event-driven computation. Due to their asynchronous and sparse nature, SNNs are especially well-suited for deployment in:

- **Edge AI devices** such as smart sensors and wearables, where power constraints limit the use of traditional deep learning models.
- **Neuromorphic hardware platforms** like Intel's Loihi or IBM's TrueNorth, which efficiently process spikes in real time.
- **Robotics and autonomous systems**, where biologically inspired decision-making under low-power conditions is critical.
- **Brain-computer interfaces (BCIs)** and neuroprosthetics, where decoding temporal neural signals is vital.

In future work, we aim to explore training SNNs with surrogate gradient methods for better convergence and evaluate their scalability on complex datasets. Hybrid SNN-ANN architectures may also combine biological efficiency with conventional learning precision.

9. Conclusion

In conclusion, this two-phase research project provided valuable insights into the comparative strengths of Spiking Neural Networks (SNNs) and traditional deep learning architectures, including Artificial Neural Networks (ANNs) and Convolutional Neural Networks (CNNs), using the Spiking Heidelberg Digits (SHD) dataset.

During Phase 1, the foundational pipeline was established through the import and preprocessing of the SHD dataset, the development of a custom PyTorch DataLoader, and the construction of a

baseline SNN using the Norse library. The model was trained for 10 epochs, with the training loss steadily decreasing from 763.9 to 757.7, confirming the model's capacity to learn temporal spike patterns effectively.

Phase 2 focused on refining the baseline by tuning key hyperparameters, training a new SNN model for 3 epochs, and benchmarking its performance against optimized ANN and CNN models. Evaluation metrics included both classification accuracy and inference time, allowing for a comprehensive comparison across architectures.

The results indicated that CNNs outperformed SNNs in both accuracy and speed; however, SNNs demonstrated potential for energy-efficient, event-driven tasks (Neftci, Mostafa, & Zenke, 2019), particularly in scenarios involving sparse and time-sensitive inputs.

Overall, this project provided deeper insight into neuromorphic computation (Tavanaei et al., 2019) and emphasized practical strategies for optimizing learning and performance across diverse neural network architectures.

References

[1] Mirza, M., & Kotelnikova, A. (2025). *Spiking neural networks: Phase 1 & 2 report* [Jupyter notebook]. DS677 Deep Learning, New Jersey Institute of Technology.

[2] Zenke, F. (n.d.). Zenke Lab: Datasets. Retrieved April 9, 2025, from `https://zenkelab.org/datasets/`

[3] Roy, K., Jaiswal, A., & Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784), 607–617. `https://doi.org/10.1038/s41586-019-1677-2`

[4] Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. (2019). Deep learning in spiking neural networks. *Neural Networks*, 111, 47–63. `https://doi.org/10.1016/j.neunet.2018.12.002`

[5] Neftci, E. O., Mostafa, H., & Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6), 51–63. `https://doi.org/10.1109/MSP.2019.2931595`