Accelerating Distributed Spike-Timing-Dependent-Plasticity Learning

Yuga Hanyu s1290224 Supervised by Khanh N. Dang

Abstract

This work introduces a novel ensemble Spike Timing Dependent Plasticity (EnsembleSTDP) approach for implementing Spiking Neural Networks (SNNs) with on-chip, online (in-situ) unsupervised learning to accelerate training. EnsembleSTDP trains multiple sub-models concurrently on separate data subsets, followed by a merging and compression process at a centralized node, achieving faster learning. We explore various merging and compression methods by identifying overlapping neurons post-training to reduce model size. Experimental results show that EnsembleSTDP, with two and five remote nodes, accelerates training on the MNIST dataset by approximately 1.98x and 5.02x, respectively, with a 0.41% accuracy gain and a 3.45% accuracy loss relative to the baseline. The source code of EnsembleSTDP learning method is available at https://github.com/klab-aizu/EnsembleSTDP.

1 Introduction

Spiking Neural Networks (SNNs) represent the third generation of neural networks, offering more biologically realistic models by using action potentials (or spikes) as fundamental elements for communication and computation [1]. Due to the natural sparsity of spikes and their event-driven operation, SNNs are considered to be more energy-efficient compared to conventional neural networks.

There are several approaches for training SNNs, including: (1) *training an equivalent ANN and converting it to an SNN* [2], (2) *training directly on spikes using backpropagation* [3], and (3) *bio-plausible learning* methods such as Spike Timing Dependent Plasticity (STDP) or Spike-Driven Synaptic Plasticity (SDSP) [4]. While the first two approaches often achieve higher accuracy, *bio-plausible* learning methods are better suited for online, on-chip (in-situ) learning, as they support lightweight computation and greater energy efficiency. In this work, we focus on the *bio-plausible* learning approach. Although lightweight on-chip learning is feasible with STDP, several challenges must be addressed:

- **Distributed datasets**: On-chip, online learning often involves distributed datasets, meaning each model is trained on different data subsets. This results in multiple neural network models with varying performance based on their respective local datasets.
- **Centralized training limitations**: Gathering data and training at a centralized node could address the distributed dataset issue, but it eliminates the benefits of on-chip and online learning, such as updating models with new data in real-time. Additionally, transferring data introduces two significant issues: (1) *privacy concerns*, as data would no longer be retained locally; and (2) *increased time and power requirements for data transfers*, particularly for low-power nodes. Furthermore, *frequent data updates do not necessarily improve overall accuracy*.
- **Existing ensemble and federated learning methods**: are potential solutions to the above challenges [5]. However, existing methods either overlook pure STDP implementations [6] or rely on different supervised learning techniques, such as Batch Normalization Through Time [5]. These approaches introduce two major issues: (1) *inefficiency for on-chip learning* due to high complexity, and (2) *infeasibility of adjusting network size during training* because of the rigid structure of these combined methods.

These challenges underscore the need for a lightweight approach to ensemble STDP learning that maintains: (1) low complexity for efficient on-chip and online learning, and (2) flexibility to adapt to different network sizes without requiring retraining at local nodes. Therefore, this work presents an approach to tackle the aforementioned challenges. The contributions of this work are as follows:

- A novel ensemble Spike Timing Dependent Plasticity (STDP) learning process that accelerates training time without requiring access to the entire dataset, thereby preserving privacy.
- An efficient model compression technique based on neuron similarity that reduces the size of the merged model while retaining key performance characteristics.

The organization of the thesis is as follows: Section 2 discusses related work. Section 3 introduces the background components of EnsembleSTDP. Section 4 details the application of ensemble learning for SNNs. Section 5 presents a comparison of various compression methods and results from merging different numbers of sub-models. Finally, Section 6 provides the conclusion.

2 Related Works

In this section, we discuss STDP learning and ensemble/federated learning for SNNs.

The effectiveness of stacked SNN ensembles has been demonstrated, achieving up to an 8% accuracy improvement over state-of-the-art techniques using only one pretrained network for transfer learning [7]. This principle is also common in ensemble learning for Neural Networks (NNs), where sub-models are trained on different dataset subsets and combined to improve overall performance and training efficiency. For example, distributed assistive STDP learning has achieved a 2.8× increase in efficiency with an accuracy improvement of approximately 2.5% [8]. In [6], the authors presented an ensemble learning approach for convolutional networks where each layer is trained using the STDP rule, while *Venkatesha et al.* [5] implemented federated SNNs, training each node with Batch Normalization Through Time.

Ensemble learning also enhances Spiking Neural Networks (SNNs) with Spike Timing Dependent Plasticity (STDP) learning. SNNs differ from other neural networks as they learn and make inferences based on spike timing, requiring only 1-bit precision for spikes, which significantly improves energy efficiency. For instance, in SNN experiments for unsupervised learning [9], SNNs reduced energy consumption by 51% on average for training and by 37% for inference compared to the state-of-the-art. STDP, a biologically plausible learning method, allows SNNs to process values such as pixel intensity over time (represented as Spike Trains), which are presented as inputs for learning [10].

By merging several sub-models trained on different dataset subsets, ensemble learning improves overall performance. This merging process combines the learned weights, enabling the final model to respond more accurately to diverse inputs.

3 Background

This section will cover the background such as ensemble learning, Spiking Neural Network, and Spike-timing-dependent plasticity learning rule.

3.1 Ensemble learning

The concept of ensemble learning is illustrated in Fig. 1 [11]. In this approach, separate models are trained in parallel on different data subsets and are then combined at a centralized system to form the final model. Unlike federated learning, where individual learners are typically homogeneous, ensemble learning allows for heterogeneity both among individual learners and in the final combined model.

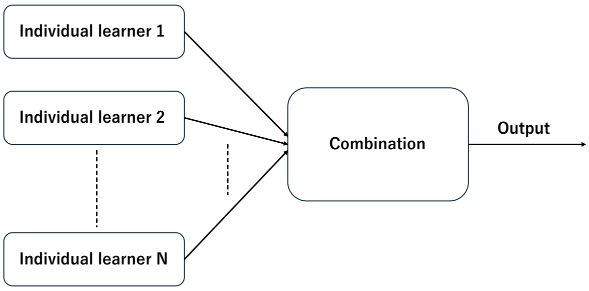

Figure 1: Ensemble learning of Neural Networks.

3.2 Spiking Neural Network

The structure of the SNN used in this work is illustrated in Fig. 2, adapted from [12]. This SNN is designed for unsupervised learning on the MNIST dataset, as discussed in Section 3.3. The network consists of three layers: an input layer, an excitatory layer, and an inhibitory layer. The input layer contains neurons corresponding to each input spike train, while the excitatory layer includes a customizable number of excitatory neurons.

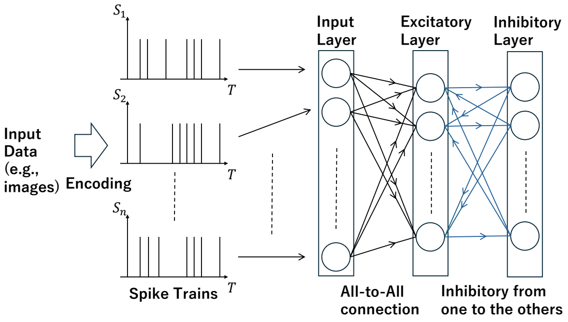

Figure 2: High-level view of spiking neural network architecture.

In the excitatory layer, spikes from the input layer are weighted by the connections between the input and excitatory neurons. These weighted values are then added to the membrane potential of each excitatory neuron. If an excitatory neuron's membrane potential exceeds a certain threshold, it emits an output spike.

The output spike from an excitatory neuron is transmitted to a corresponding inhibitory neuron (with the same index), which then generates a lateral inhibition signal to suppress activity in all other excitatory neurons. The excitatory neuron then enters a refractory state, preventing it from firing for several time steps to avoid dominating the network and allowing other neurons to activate.

3.3 Spike-timing-dependent plasticity

In the SNN model described above, STDP learning is applied during the training process [12].

STDP, or Spike Timing Dependent Plasticity, refers to the process where the timing of spikes influences synaptic strength, in this case, the weight [13–15]. Fig. 3 illustrates the basic pre- and post-synaptic model of STDP learning. The detailed relationship between spike timings and weight updates are also shown in Fig. 4.

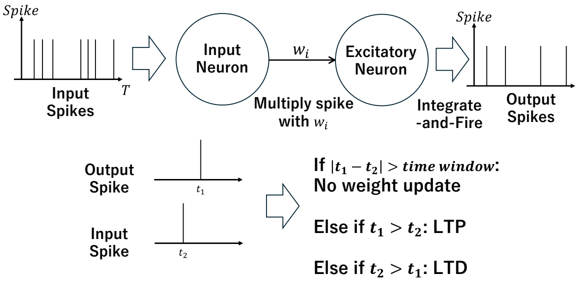

Figure 3: Learning model with relative timing.

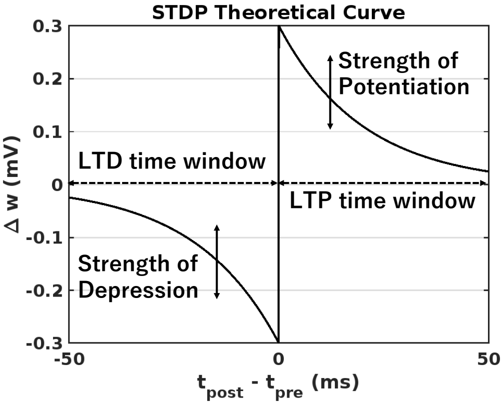

Figure 4: Timing Dependent Plasticity Learning (STDP) learning

4 Ensemble STDP Learning in Spiking Neural Networks

In this section, we describe the proposed ensemble STDP learning method. We begin with an overview of the system, followed by a detailed explanation of its operation, including concurrent learning, concatenation, and compression processes.

4.1 Proposed Ensemble Learning

Fig. 5 illustrates the proposed ensemble learning process for SNNs. The dataset is first divided into subsets, with each subset being fed into a sub-model for concurrent training. Once the sub-models are trained, they are merged into a single model through concatenation. At this stage, overlapping neurons that perform similar tasks may appear in the merged model. To address this, we apply compression techniques to remove redundant neurons, retaining only the most useful ones, thereby reducing the size of the final model while preserving its performance.

To compress the concatenated model, neuron pairs are compared using various similarity metrics, such as Mean Squared Error (MSE), Manhattan Distance, Cosine Similarity, and the Correlation Coefficient of the weights. These metrics help identify redundant neurons, which can be removed to optimize the model's size without sacrificing accuracy. Ensemble learning provides several advantages for developing Spiking Neural Networks by distributing the learning task across multiple sub-models and employing efficient merging techniques.

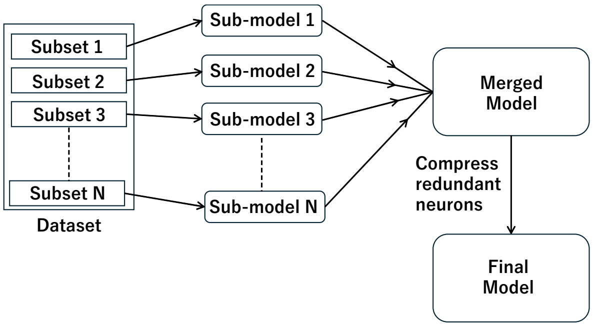

Figure 5: Proposed ensemble learning model.

4.2 Proposed merging approach of Spiking Neural Network sub-models

In this section, we first propose a concatenation method to combine the sub-models into a single model. Then, we illustrate the issue of merging and how to merge.

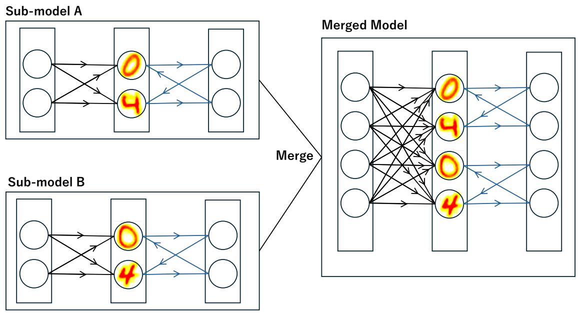

Figure 6: Principle of merging sub-models

4.2.1 Proposed concatenation method

Once several sub-models are trained, the next step is to merge them into a single SNN model. Merging sub-models involves modifications to all layers and connections, including data such as membrane potentials, spike occurrences, and refractory period records. However, the most crucial aspect of merging is the concatenation of excitatory neurons.

The principle of merging sub-models is illustrated in Fig. 6. In this simple example using a handwritten digits dataset, sub-models A and B each contain neurons labeled 0 and 4. The neurons are represented by their corresponding weight matrices. Although the label assignments are the same in both models, the weight distributions differ due to the inherent variations in the representation of handwritten digits. As a result, a single sub-model may not be able to correctly classify all variations of a given number. In the merged model, however, different variations of digits 0 and 4 from both sub-models A and B are included. This increases the likelihood that the merged model will perform better than the individual sub-models.

4.2.2 Issues of merged model

When merging sub-models, two main issues arise: the merged model may become too large, and redundant neurons may remain after merging. To address these issues, we generate neuron pairs and measure the similarity of each pair after concatenating the models. To preserve the classification accuracy of the final model, the neurons to be compressed are selected such that only one of the similar neurons is retained, ensuring that the compression does not negatively impact the model's performance.

4.2.3 Preliminary result

To verify that removing one of the similar neurons preserves classification accuracy, we manually compressed two 64-neuron models and then merged them into a single model. Fig. 7 illustrates the 2D receptive fields of the two trained 64-neuron sub-models. The 2D receptive field is a visual representation of the learned weights of a neuron. The training sub-datasets consisted of different sets of 3000 images for each sub-model.

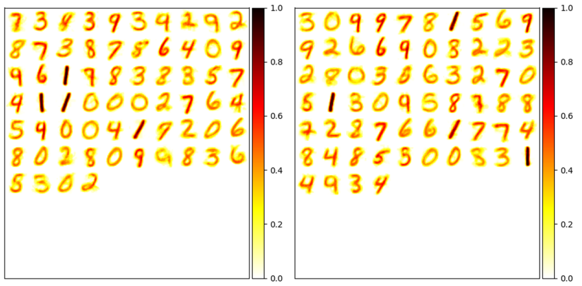

Figure 7: Examples of 2D receptive fields of 2 models.

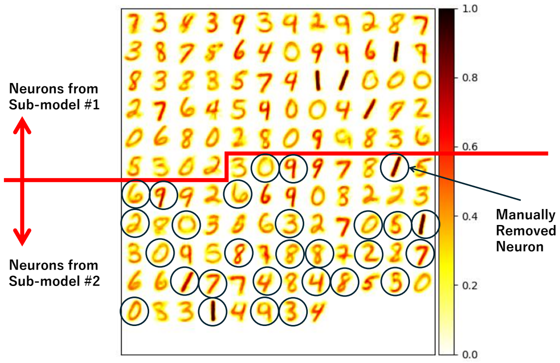

Figure 8: 2D receptive field of merged model and compressed neurons.

As shown in Fig. 7, there are similar neurons between the two models. For simplicity, 28 neurons were removed from model #2 based on human observation. The 2D receptive field of the merged model, along with the manually removed neurons, is shown in Fig. 8. Each removed neuron from sub-model #2 corresponds to a similar neuron in sub-model #1. Table 1 presents a comparison of the classification accuracies of the merged models.

As shown in Table 1, compressing 28 neurons out of 128 resulted in only a 0.76% decrease in classification accuracy. Removing one of the similar neurons retains classification accuracy while reducing the number of neurons. Building on this observation, we aim

Table 1: Preliminary result of no compression (merge only) and compression under merging two 64 neurons models.

Model	sub-model #1	sub-model #2	Merged Model	Manually Compressed Model
No. of neurons	64	64	128	100
Accuracy	75.38%	75.54%	77.60%	76.84%

to automate the compression process using a similarity measurement, allowing it to scale to a large number of neurons and models. The automatic neuron compression (using MSE) achieved a classification accuracy of 77.08% under the same conditions as the manually compressed model, as shown in Table 1.

4.3 Automatic Neuron Compression

The major steps of automatically compressing models are as follows:

1) Separate the model and dataset into smaller sub-models and sub-datasets, then train each sub-model on a different sub-dataset concurrently.

2) Concatenate the trained sub-models. The label assignments for the neurons in each sub-model are also concatenated at this step.

3) Generate all possible neuron pairs from the concatenated model and assign a similarity value to each pair. These similarity pairs form the basis for the compression process. The number of possible pairs and how we generate them is as follows:

    - Number of pairs: $nC_2 = \frac{n!}{2!(n-2)!}$ where n is the total number of neurons.
    - Algorithm 1 illustrates the pair generation and the similarity value calculation process.
    - The `calculate_similarity(i, j)` function can be computed using one of the following methods: MSE, Manhattan Distance, Cosine Similarity, or Correlation Coefficient.

4) Sort the pairs in descending order of similarity.

5) Given the number of compressions, determine which neurons to compress using the sorted similarity pairs. The compression priority is determined by the order of similarity values, starting from the highest. Continue down the sorted list of similarity pairs until the required number of neurons to compress is reached.

6) Remove the selected neurons from the final model.

Algorithm 1 Generate pairs with similarity values.

```
Input: $n$
Output: pairs
pairs ← {}
for $i \leftarrow 0$ to $n - 1$ do
    for $j \leftarrow i + 1$ to $n$ do
        similarity ← calculate_similarity($i, j$)
        pairs.append(($i, j$, similarity))
    end for
end for
```

There are several neuron similarity measurements that can be used to calculate the similarity value for a pair. Each measurement has its unique characteristics when comparing two neurons. Details about each measurement are provided in the following sections.

4.3.1 Mean Squared Error

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$$

Where:

- N represents the total number of weights in each neuron.
- x_i and y_i are the corresponding elements in the weight matrices.

Mean Squared Error (MSE) calculates the average of the squared differences between the weight matrices of two neurons. It is sensitive to large differences due to squaring the differences, which is beneficial in similarity measurement. The main advantage of MSE lies in its simplicity and ease of interpretation. However, its sensitivity to outliers can sometimes lead to misleading similarity results.

4.3.2 Manhattan Distance

$$\text{Manhattan Distance} = \sum_{i=1}^{N}|x_i - y_i|$$

Manhattan Distance, also known as the L1 norm, measures the absolute differences between weight matrices and sums them up. This distance metric is more robust to outliers compared to MSE and provides a straightforward measure of similarity based on individual element differences.

Its main weakness is that it might not capture the overall similarity between neurons if the differences are spread across many dimensions. For instance, if each element in the matrix is slightly but consistently different, this measurement will calculate a large value, even though the matrices may be similar overall.

4.3.3 Cosine Similarity

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||\,||\mathbf{y}||}$$

Where:

- $\mathbf{x} \cdot \mathbf{y}$ is the dot product of weight matrices $\mathbf{x}$ and $\mathbf{y}$.
- $||\mathbf{x}||$ is the magnitude of matrix $\mathbf{x}$, calculated as $||\mathbf{x}|| = \sqrt{\sum_{i=1}^{N} x_i^2}$.
- $||\mathbf{y}||$ is the magnitude of matrix $\mathbf{y}$, calculated as $||\mathbf{y}|| = \sqrt{\sum_{i=1}^{N} y_i^2}$.

Cosine Similarity measures the cosine of the angle between weight matrices, comparing their direction rather than magnitude. This metric is particularly useful for determining similarity when the magnitude of the weights is less important than their direction. It is less sensitive to differences in magnitude, making it robust in scenarios where the scale of neuron weights varies.

However, it may not be suitable in cases where the differences in magnitude are meaningful, as it disregards magnitude differences and focuses solely on directional similarity.

4.3.4 Correlation Coefficient

$$r = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

Where:

- $\bar{x}$ is the mean of the x values, calculated as $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$.
- $\bar{y}$ is the mean of the y values, calculated as $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$.

The Correlation Coefficient measures the degree to which two weight matrices move together. A value close to 1 indicates a strong positive correlation, -1 indicates a strong negative correlation, and 0 indicates no correlation. This metric is advantageous for identifying linear relationships between neurons.

However, it may not fully capture non-linear relationships between elements, and can be influenced by extreme values, which may impact the similarity measurement.

4.4 Nature of Ensemble Learning and Trade-Offs

Ensemble learning accelerates the training process by concurrently training sub-models on different subsets of the dataset. However, splitting the dataset and models, and then concatenating them after training, inherently results in a performance loss compared to training a single model on the entire dataset. Moreover, as the performance of each sub-model improves, the benefit of concatenating them diminishes. This is because merging becomes less advantageous as the pre-trained sub-models become more complete on their own.

Taking these factors into account, we aim to maximize the benefits of ensemble learning. The test cases are designed based on the following considerations:

- *Keep the number of sub-models manageable:* While it is possible to use 10, 20, or more sub-models, the performance loss compared to a single benchmark model increases as more sub-models are merged. For example, with 30K training images and a total of 500 neurons, concatenating 2 (250×2 neurons), 5 (100×5 neurons), and 10 (50×10 neurons) sub-models results in classification accuracies of 90.34%, 86.83%, and 82.61%, respectively. Additionally, since the complexity of merging and compressing models is $O(n^2)$, processing time becomes a bottleneck when the total number of neurons approaches 1000.
- *Increase the learning ratio:* We observed that neurons in sub-models tend to remain undertrained due to the dataset being split into smaller subsets. Therefore, increasing the learning ratio for the sub-models provides a more significant advantage in ensemble learning than training a single model on the whole dataset. To ensure fair comparison, the learning ratio is kept the same for both sub-models and the benchmark model.
- *Compress the model to match the benchmark's neuron count:* It is important to note that the number of neurons has less impact on training time compared to the size of the training dataset. This allows us to use more neurons during training and then compress the model afterward to match the neuron count of the benchmark, ensuring a comparable final model size.

5 Evaluation Results

5.1 Evaluation Methodology

In this evaluation, we model our Ensemble STDP learning using the BindsNet library [16]. The models were trained and tested on Ubuntu 22.04 machines equipped with an Intel Core i7-13700K processor, 64 GB DDR5 RAM, and an Nvidia RTX 4070 GPU. We trained several sub-models on different portions of the MNIST dataset and subsequently merged them into a single model, followed by neuron compression. The number of training images and neurons in the merged model was kept consistent with the benchmark model. First, we compare the performance of each compression method on the benchmark model. Then, we train and merge 5 sub-models, followed by 2 sub-models, and compare their performance with the benchmark.

5.2 Accuracy of assemble STDP learning

Fig. 9 shows the neuron compression results for a 300-neuron SNN model using four different similarity measurements. The model without compression achieves a classification accuracy of 88.87%, and as neurons are compressed, the performance gradually decreases. In this case, despite the minimal differences between the methods up to 50 compressed neurons, the MSE method consistently performed the best in terms of retaining classification accuracy.

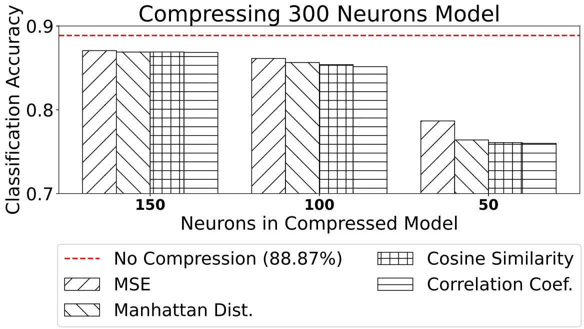

Figure 9: Neuron compression from a 300-neuron SNN model using four different similarity measurements.

5.3 Execution time of assemble STDP learning

The average merging and compressing times for each method are shown in Fig. 10. We observe that using the Correlation Coefficient results in significantly longer processing times compared to the other three methods. Additionally, the time difference across varying numbers of compressions is negligible. This is because the majority of the merging time is consumed by the concatenation of sub-models and the generation of neuron pairs.

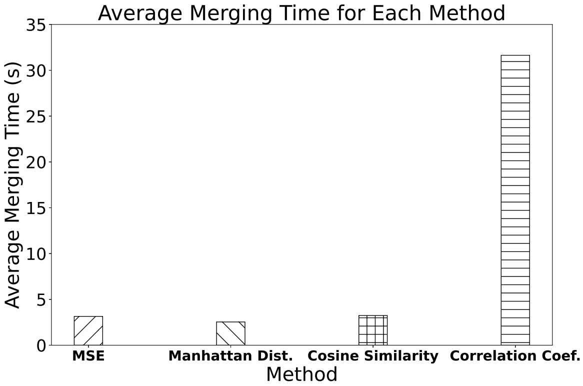

Figure 10: Average Merging Times in Merging 5×100 Neurons Sub-models

5.4 Comparison to the baseline

To assess the learning efficiency, we compare our approach with the baseline where the entire dataset is centralized for training. Our results are contrasted with the BindsNet implementation [16] of the method described in [12].

Following are comparison tables of our merging method and single benchmark model under the same number of neurons and dataset.

Table 2: Comparison between one 300-neuron model and merging 5×100-neuron sub-models, followed by 200-neuron compression. Trained on 30K images.

Model	[12]	**Ours**
#neurons	300	300 (5×100-200)
Training Time (minutes)	53.13	10.58
Classification Accuracy	88.87%	85.42%

Table 2 compares the highest-performing method (Manhattan Distance) in this specific case. Our method achieved a 5x speedup in training, with a trade-off of a 3.45% classification accuracy loss. Table 3 presents another comparison between our merging method and the benchmark model.

The compression was performed using the highest-performing method (MSE) in this specific case. Our merging method slightly outperformed the single benchmark model.

We have tested the basic cases of ensemble learning and compared them with single benchmark models. As expected, we observed trade-offs when training multiple sub-models and merging them into a single model. However, in some use cases, this approach yielded ben-

Table 3: Comparison between a single 300-neuron model and merging 2 × 250-neuron sub-models, followed by compression to 200 neurons. Trained on 30K images.

Model	[12]	**Ours**
#neurons	300	300 (2×250-200)
Training Time (minutes)	53.13	26.8
Classification Accuracy	88.87%	89.28%

eficial results.

6 Conclusion

In this paper, we proposed a method to train SNN sub-models on different datasets, merge them, and compress neurons. Our approach achieved a 5× speedup in training with only a 3.45% accuracy loss. Merging two sub-models outperformed the single benchmark model, and compression retained performance while reducing model size. This method is particularly useful for large datasets, speeding up training while maintaining a reasonable model size.

Future work could focus on improving communication between sub-models during training and exploring collaborative learning, where sub-models trained in different locations are merged without sharing data.

References

[1] P.A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," Science, vol.345, no.6197, pp.668–673, 2014.

[2] B. Rueckauer *et al.*, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," Frontiers in neuroscience, vol.11, p.682, 2017.

[3] M. Dampfhoffer *et al.*, "Backpropagation-based learning techniques for deep spiking neural networks: A survey," IEEE Transactions on Neural Networks and Learning Systems, 2023.

[4] M.R. Azghadi *et al.*, "Spike-based synaptic plasticity in silicon: design, implementation, application, and challenges," Proceedings of the IEEE, vol.102, no.5, pp.717–737, 2014.

[5] Y. Venkatesha, Y. Kim, L. Tassiulas, and P. Panda, "Federated learning with spiking neural networks," IEEE Transactions on Signal Processing, vol.69, pp.6183–6194, 2021.

[6] Q. Fu and H. Dong, "An ensemble unsupervised spiking neural network for objective recognition," Neurocomputing, vol.419, pp.47–58, 2021.

[7] M. Mohammadi and S. Das, "Snn: Stacked neural networks," 2016.

[8] P. Panda *et al.*, "Ensemblesnn: Distributed assistive stdp learning for energy-efficient recognition in spiking neural networks," International Joint Conference on Neural Networks (IJCNN), pp.2629–2635, 2017.

[9] R.V.W. Putra and M. Shafique, "Spikedyn: A framework for energy-efficient spiking neural networks with continual and unsupervised learning capabilities in dynamic environments," CoRR, vol.abs/2103.00424, 2021.

[10] H. Markram, W. Gerstner, and P.J. Sjöström, "Spike-timing-dependent plasticity: A comprehensive overview," Frontiers in Synaptic Neuroscience, vol.4, 2012.

[11] Z.H. Zhou, Ensemble methods: foundations and algorithms, CRC press, 2012.

[12] P.U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," Frontiers in computational neuroscience, vol.9, p.99, 2015.

[13] C.C. Bell, V.Z. Han, Y. Sugawara, and K. Grant, "Synaptic plasticity in a cerebellum-like structure depends on temporal order of input," Nature, vol.387, no.6633, pp.278–281, 1997.

[14] H. Markram, J. Lubke, M. Frotscher, and B. Sakmann, "Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps," Science, vol.275, no.5297, pp.213–215, 1997.

[15] G.Q. Bi and M.M. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," The Journal of Neuroscience, vol.18, no.24, pp.10464–10472, 1998.

[16] H. Hazan *et al.*, "Bindsnet: A machine learning-oriented spiking neural networks library in python," Frontiers in neuroinformatics, vol.12, p.89, 2018.