

## APPENDIX

**Anonymous authors**

Paper under double-blind review

### A. ENERGY ANALYSIS

This section analyzes SPARTA’s energy consumption, including calculation methodology, performance comparison, and component-wise breakdown.

#### A.1. CALCULATION METHODOLOGY

The energy consumption is calculated based on the operation type and spike activity Panda et al. (2020). The initial layer processing continuous data uses Multiply–Accumulate (MAC) operations, while subsequent layers processing spike-based data use Accumulate (AC) operations, scaled by their input firing rates ( $S_{in}$ ). Our estimation adopts the standard 45 nm CMOS values, where one MAC costs  $E_{MAC} = 4.6$  pJ and one AC costs  $E_{AC} = 0.9$  pJ Panda et al. (2020). Peak activation memory is estimated by summing the output tensor sizes of each layer.

| Block      | Layer           | Energy Consumption   |
|------------|-----------------|--|
| STEN       | First Conv      | $E_{MAC} \cdot F_{Conv} \cdot T$                           |
|            | Other Convs     | $E_{AC} \cdot F_{Conv} \cdot T \cdot S_{in}$               |
| MSP & STSG | All Conv/Linear | $E_{AC} \cdot F_{Conv/Linear} \cdot T \cdot S_{in}$        |
| SC         | Q, K, V Proj.   | $3 \cdot E_{AC} \cdot F_{Linear} \cdot T \cdot S_{in}$     |
|            | Self-attn Ops   | $E_{AC} \cdot T \cdot k \cdot D^2 \cdot (S_Q + S_K + S_V)$ |
|            | FFN (MLP)       | $E_{AC} \cdot F_{Linear} \cdot T \cdot S_{in}$             |

Table 1: Energy-consumption equations for SPARTA computational layers.  $T$ : time steps,  $S_{in}$ : input firing rate.

The sparse nature of spike-based computation enables energy consumption to scale with actual neural activity rather than theoretical computational capacity, resulting in significant energy savings during inference.

#### A.2. PERFORMANCE RESULTS

We evaluated SPARTA’s computational efficiency on DVSGesture. Energy cost for one DVSGesture samples are summarized in Table 2.

| Method               | Prec.       | T         | Mem (MB)      | Energy (mJ)  |
|----------------------|-------------|-----------|---------------|--------------|
| ViT-B/16             | 32/32       | 1         | 351.8         | 254.84       |
| Spikformer           | 32/1        | 16        | 186.73        | 28.57        |
| SpikingReformer-s    | 32/1        | 16        | 286.84        | 15.16        |
| SeWResnet-18         | 32/1        | 16        | 289.11        | 30.06        |
| <b>SPARTA (Ours)</b> | <b>32/1</b> | <b>16</b> | <b>323.26</b> | <b>17.97</b> |
| <b>SPARTA (Ours)</b> | <b>32/1</b> | <b>20</b> | <b>342.15</b> | <b>19.21</b> |

Table 2: Comparison of precision, timestep, memory, and energy per DVSGesture sample. T indicates timesteps.

SPARTA achieves the lowest energy consumption among all methods. At T=16, SPARTA consumes 17.97 mJ compared to Spikformer’s 28.57 mJ. The theoretical computational requirement for

SPARTA is 80.67 GFLOPs when implemented as an ANN-based method. However, through the SNN architecture, the actual computational load is reduced to 1.23 GFLOPs.

### A.3 ENERGY CONSUMPTION OF COMPONENTS

Table 3 presents the component-wise energy breakdown of the SPARTA model, revealing the effectiveness of hierarchical information processing in SNN architectures.

| Block        | Params (M)  | (%)         | Energy (mJ)  | (%)          |
|--------------|-------------|-------------|--------------|--------------|
| STEN         | 2.91        | 21.1%       | 15.04        | <b>83.7%</b> |
| MSP          | 5.29        | 38.4%       | 1.01         | 5.6%         |
| STSG         | 1.04        | 7.5%        | 1.89         | <b>10.5%</b> |
| SC           | 4.55        | 33.0%       | 0.03         | 0.2%         |
| <b>Total</b> | <b>13.8</b> | <b>100%</b> | <b>17.97</b> | <b>100%</b>  |

Table 3: Component-wise breakdown of parameters and energy consumption for the SPARTA model. The total parameter count of 10.94M reflects the model at inference time

The STEN module dominates energy consumption (83.7%) despite having only 21.1% of parameters due to temporal processing complexity. The SC module shows extreme efficiency, consuming 0.2% of energy while holding 33.0% of parameters, demonstrating effective sparse activation patterns. Energy consumption decreases dramatically through the pipeline (STEN: 83.7%  $\rightarrow$  MSP: 5.6%  $\rightarrow$  SC: 0.2%), with total consumption of 17.97 mJ per inference.

## B. HETEROGENEITY GRANULARITY (LAYER, CHANNEL, AND NEURON-WISE)

While LSNN Bellec et al. (2018) and Diet-SNN Rathi & Roy (2023) use neuron-wise or layer-wise random  $\tau/V_{th}$ , our HI-LIF introduces channel-wise Gaussian priors, scaling heterogeneity to convolutional/ViT features and empirically widening temporal bandwidth.

**Experimental design.** We compare three granularity levels for  $\tau$  and  $V_{th}$ :

1. *Layer-wise*: all neurons in a layer share  $(\tau, V_{th})$  pair.
2. *Channel-wise (ours)*: each feature channel has own pair.
3. *Neuron-wise*: every spatial location maintains independent parameters.

### B.1 MOTIVATION

Prior work randomised  $(\tau, V_{th})$  either *layer-wise* (low diversity) or *neuron-wise* (high cost). We test an intermediate *channel-wise* setting: one parameter pair per feature channel.

| Granularity  | Parameter Count      | Memory Overhead |
|--------------|----------------------|-----------------|
| Layer-wise   | $2L$                 | baseline        |
| Channel-wise | $2 \sum C_l$         | +13.5 %         |
| Neuron-wise  | $2 \sum C_l H_l W_l$ | +52.8 %         |

Table 4: Parameter scaling and memory overhead at each granularity ( $L$ : layers).

All models use Section E hyper-parameters and are trained for 300 epochs on DVS-Gesture, varying only granularity.

### B.2 RESULTS & DISCUSSION

Channel-wise heterogeneity achieves the highest accuracy (98.46%), outperforming both layer-wise (91.00%) and neuron-wise (96.60%) approaches. This marks a 7.46 percentage point gain over

| Granularity         | Acc. (%)     | Memory (MB)  |
|---------------------|--------------|--------------|
| Layer-wise          | 91.00        | 285.2        |
| <b>Channel-wise</b> | <b>98.46</b> | <b>323.6</b> |
| Neuron-wise         | 96.60        | 435.8        |

Table 5: Accuracy and memory usage on DVS-Gesture for different parameter granularities.

| Granularity         | Sparsity (%) | Peak Step      | Active Pixels |
|---------------------|--------------|----------------|---------------|
| Layer-wise          | 77.5         | 16 / 16        | 8,548         |
| <b>Channel-wise</b> | <b>65.4</b>  | <b>13 / 16</b> | <b>10,026</b> |
| Neuron-wise         | 61.5         | 9 / 16         | 10,842        |

Table 6: Sparsity and spike activity metrics on DVS-Gesture for different parameter granularities.

layer-wise and 1.86 percentage point over neuron-wise, with only 13.5% more memory than layer-wise.

The results indicate channel-wise granularity balances model capacity and efficiency: it avoids the limited expressiveness seen in high sparsity layer-wise models (77.5%) and the complexity of neuron-wise models with lower sparsity (61.5%) and higher memory costs. With 65.4% sparsity and 323.6 MB memory, channel-wise achieves effective feature learning. Peak spike activity occurs at step 13 for channel-wise, between layer-wise (step 16) and neuron-wise (step 9), suggesting balanced temporal processing. Thus, channel-wise is adopted as the default.

## C. DETAILS FOR COMPONENT BREAKDOWN

This section provides detailed analysis of key SPARTA components, focusing on internal mechanisms and decision-making processes.

### C.1 DETAILS FOR DYNAMIC SPARSITY PREDICTOR

The dynamic sparsity predictor in STSG operates through a three-layer MLP that transforms input characteristics into adaptive sparsity ratios.

The predictor transforms a 3-dimensional input vector through the following sequence:

$$\mathbf{s}_{\text{input}} = [\bar{f}, \sigma_t, \bar{i}] \quad (1)$$

$$\mathbf{h}_1 = \text{GELU}(\mathbf{W}_1 \text{LN}(\mathbf{s}_{\text{input}}) + \mathbf{b}_1) \quad (2)$$

$$\mathbf{h}_2 = \text{GELU}(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2) \quad (3)$$

$$r_{\text{sparse}} = \sigma(\mathbf{w}_3^T \mathbf{h}_2 + b_3) \quad (4)$$

where  $\bar{f}$  is mean firing rate across tokens,  $\sigma_t$  is timing map standard deviation, and  $\bar{i}$  is mean inter-spike interval.

**Decision Process Analysis.** The predictor exhibits three behavioral regimes:

- **High Activity Regime** ( $\bar{f} > 0.6$ ): The predictor increases sparsity ratio to 0.7–0.8 to suppress noisy spikes.
- **Temporal Coherence Regime** ( $\sigma_t < 0.3$ ): When temporal variance is low, sparsity is reduced to 0.4–0.5 to preserve temporally synchronized events.
- **Mixed Regime**: The predictor balances sparsity based on both activity level and temporal structure.

### C.2 DETAILS FOR MSP WEIGHT LEARNING METHODOLOGY

The Multi-Scale Processing (MSP) module employs learnable scale fusion weights to combine information across different temporal scales.

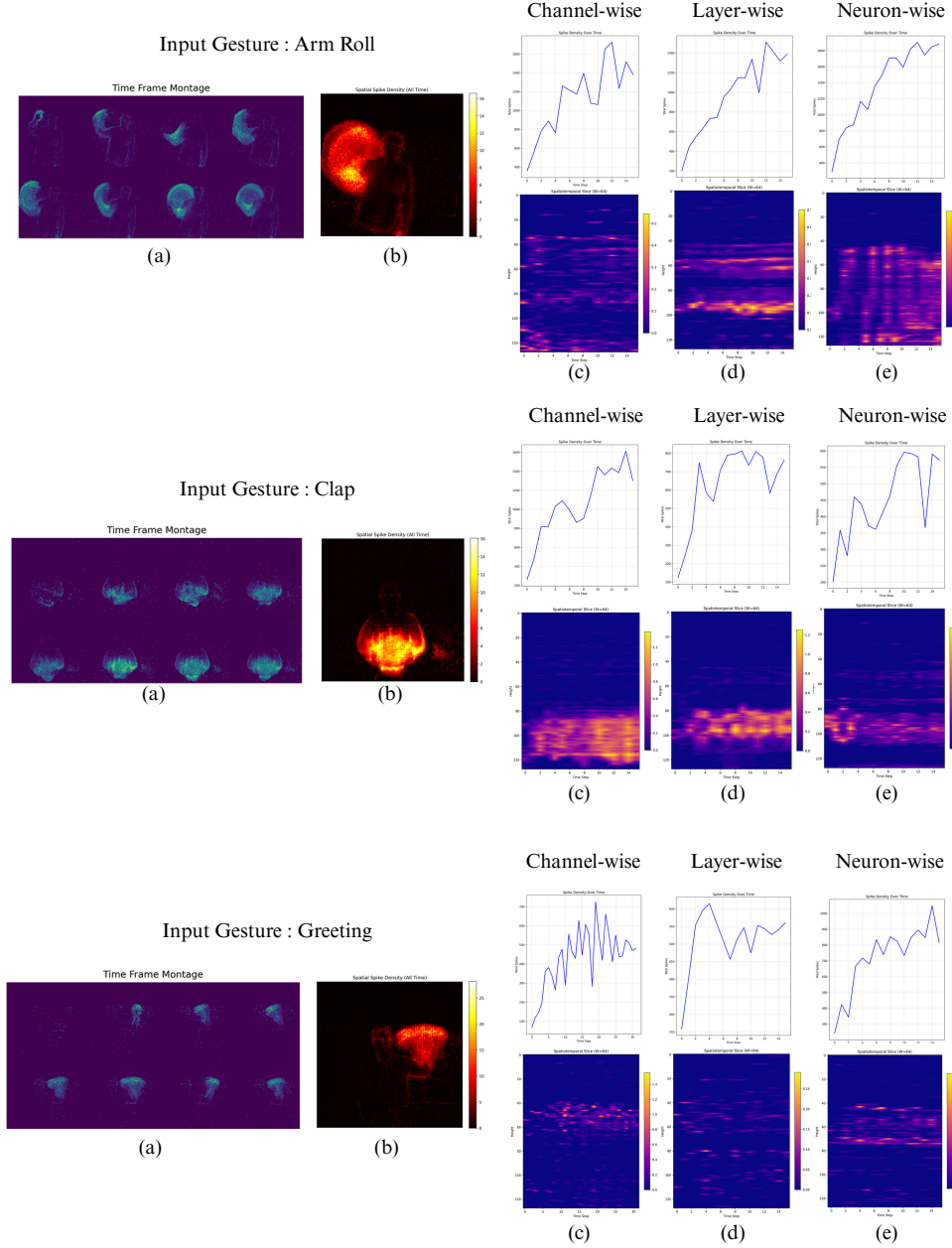


Figure 1: Heterogeneity visualisation on three DVS-Gesture samples. (a) Input frames, (b) Firing rate map, (c–e) Spike distribution under different granularities.

**Scale Fusion Mathematics:** During training, the scale fusion operates through:

$$\alpha_{\text{soft}} = \text{Softmax}(\alpha_{\text{raw}}) \quad (5)$$

$$\mathbf{F}_{\text{fused}} = \sum_{s=1}^3 \alpha_s \cdot \mathbf{F}_s \odot \mathbf{M}_s \quad (6)$$

where  $\mathbf{F}_s$  represents features from scale  $s$ ,  $\mathbf{M}_s$  is the scale mask, and  $\alpha_{\text{raw}}$  are learnable parameters.

**Scale-Specific Processing:** Each scale  $s \in \{4, 8, 12\}$  processes features through dedicated convolution paths with stride mapping  $\{4 : 1, 8 : 2, 12 : 3\}$ .

**Weight Evolution Analysis:** Empirical analysis shows:

- Scale 4 (fine temporal details):  $\alpha_1 = 0.42 \pm 0.08$
- Scale 8 (intermediate patterns):  $\alpha_2 = 0.35 \pm 0.06$
- Scale 12 (coarse structure):  $\alpha_3 = 0.23 \pm 0.05$

### C.3 DETAILS FOR SPARSE ATTENTION LAYER ARCHITECTURE

The Sparse Attention Layer implements attention computation through specialized neuromorphic components.

**Core Attention Computation:** The sparse attention mechanism operates through:

$$\mathbf{Q}, \mathbf{K}, \mathbf{V} = \mathbf{XW}_Q, \mathbf{XW}_K, \mathbf{XW}_V \quad (7)$$

$$\mathbf{S} = \text{RMS}(\mathbf{QK}^T) \cdot \frac{\gamma}{\sqrt{d_k}} \quad (8)$$

$$\mathbf{A} = \text{Softmax}(\mathbf{S}) \quad (9)$$

$$\mathbf{O} = \mathbf{AV} \odot \text{SE}(\mathbf{AV}) \quad (10)$$

**RMS Normalization:** RMS normalization provides numerical stability:

$$\text{RMS}(\mathbf{x}) = \frac{\mathbf{x}}{\sqrt{\text{Var}(\mathbf{x}) + \epsilon}} \quad (11)$$

where variance is clamped to  $[\epsilon, 50.0]$  for spike-based input stability.

**Temperature Scaling:** The learnable temperature parameter  $\gamma$  sharpens attention distributions:

$$\gamma_{\text{learned}} = 5.0 \pm 0.8 \quad (12)$$

**Squeeze-and-Excitation Gating:** Channel-wise attention modulation follows:

$$\mathbf{g} = \sigma(\mathbf{W}_2 \text{GELU}(\mathbf{W}_1 \text{GlobalPool}(\mathbf{F}))) \quad (13)$$

$$\mathbf{F}_{\text{enhanced}} = \mathbf{F} \odot \text{clamp}(\mathbf{g}, 0.1, 0.9) \quad (14)$$

### C.4 ANALYSIS OF PERFORMANCE VARIATIONS ACROSS DATASETS

This subsection analyzes SPARTA’s performance characteristics across different dataset types.

**Temporal Information Dependency.** SPARTA achieves state-of-the-art performance on DVS-Gesture while showing performance variations on CIFAR10-DVS, CIFAR-10, and CIFAR-100. This performance pattern correlates with temporal information availability.

**DVS-Gesture Performance.** The attention score distribution (Figure 2c, leftmost) shows a bimodal pattern with distinct peaks, indicating clear discrimination between gesture features and background.

**CIFAR10-DVS Performance.** CIFAR10-DVS maintains event camera characteristics that preserve data sparsity. The attention distribution shows intermediate selectivity with broader spread across the range.

**Rate Coding Considerations.** In RGB datasets converted through rate coding, background regions with high pixel intensities result in elevated firing rates, which affects attention distribution patterns.

While SPARTA demonstrates strong performance on temporal-rich dataset, opportunities remain for improvement in rate-coded RGB applications.

## D. PARAMETER DETAILS

This appendix provides comprehensive implementation details for SPARTA, including hyperparameter configurations, experimental setup, and reproducibility information.

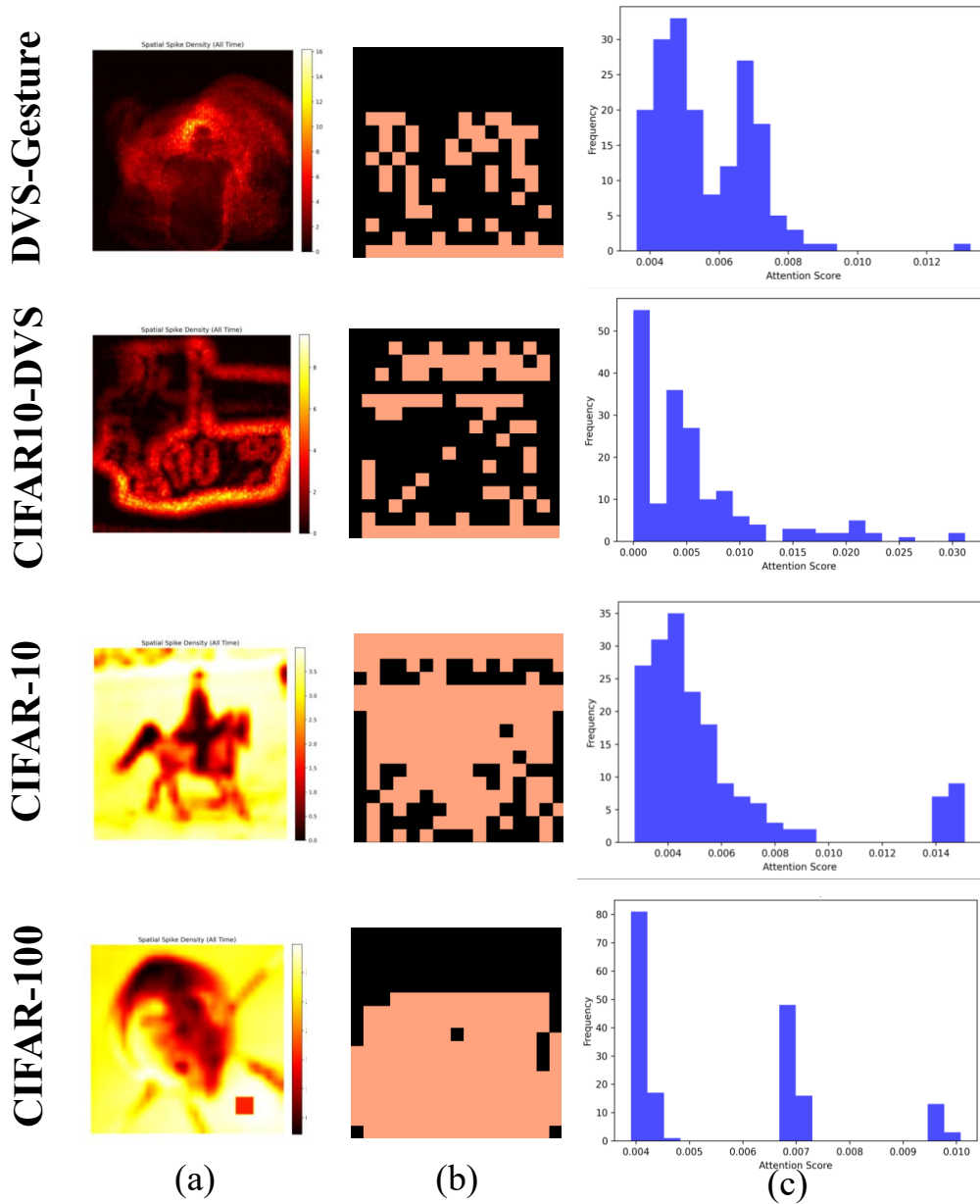


Figure 2: Performance analysis of SPARTA’s sparse attention mechanism across datasets. (a) Spatial firing rate density maps showing dataset-specific activation patterns. (b) Selected top-k attention masks (black: masked tokens, orange: selected tokens). (c) Sparse attention score distributions showing varying token selection patterns across different input modalities.

#### D.1 QUICK-START RESOURCES

- **Code** A GitHub repository will be opened upon acceptance containing (i) data download scripts, (ii) training/inference launch files, (iii) log-to-table conversion utilities, and (iv) figure generation notebooks.
- **Background reading** For readers new to SNNs we recommend: Maass (1997), Neftci & Schrauwen (2019) for surrogate gradients, and Akbari et al. (2021) for event-camera datasets.

## D.2 HYPER-PARAMETER SEARCH

All tuning was grid search on the validation set; Table 7 lists ranges, trials, and the selected value.

| Parameter     | Range                        | Trials | Chosen             |
|---------------|------------------------------|--------|--------------------|
| Learning rate | $\{1, 2, 4\} \times 10^{-4}$ | 3      | $2 \times 10^{-4}$ |
| Batch size    | $\{8, 16, 20\}$              | 3      | 8                  |
| $\tau$ mean   | $\{1.5, 1.7, 2.0\}$          | 3      | 2.0                |
| $\tau$ std    | $\{0.3, 0.5, 0.7\}$          | 3      | 0.3                |
| Dropout       | $\{0.0, 0.1, 0.2\}$          | 3      | 0.1                |

Table 7: Grid-search details used during development.

## D.3 EXECUTION ENVIRONMENT

|           |   |
|-----------|---|
| GPU       | NVIDIA A100 80GB and RTX 4090                     |
| CPU       | AMD EPYC 7763 64-Core Processor                   |
| RAM       | 1 TB  |
| OS        | Ubuntu 22.04 LTS                                  |
| Framework | Spikingjelly 0.0.0.0.15, PyTorch 2.0.1, CUDA 12.4 |

Table 8: Hardware and software used for every reported run.

Every score in the main paper is the mean of three independent runs.

## D.4 FINAL HYPER-PARAMETER LIST

Table 9 enumerates every train-time parameter excluding data augmentation, which is detailed separately in Section D.6.

| Category     | Name                | Value              |
|--------------|---------------------|--------------------|
| Optimizer    | Type                | AdamW              |
|              | Learning rate       | $2 \times 10^{-4}$ |
|              | $\beta$             | (0.9, 0.98)        |
|              | Weight decay        | $5 \times 10^{-5}$ |
| Training     | Epochs              | 300                |
|              | Batch size          | 8                  |
|              | Scheduler           | Cosine             |
|              | Warm-up epochs      | 10                 |
| Spike params | $\tau$ mean / std   | 2.0 / 0.3          |
|              | $V_{th}$ mean / std | 1.0 / 0.2          |
|              | Detach reset        | False              |

Table 9: Complete hyper-parameter configuration used in all results (excluding augmentation).

## D.5 AUGMENTATION

We employ the Neuromorphic Data Augmentation (NDA) Transform framework (Li et al., 2022), which preserves temporal characteristics of event data and provides robust regularization. The key parameters used are summarized in Table 10.

| NDA Category      | Parameter             | Value        |
|-------------------|-----------------------|--------------|
| Safe Spatial      | Horizontal flip prob. | 0.3          |
|                   | Probability threshold | 0.15         |
| Conservative Zoom | Scale range           | [0.95, 1.05] |
|                   | Application prob.     | 0.2          |
|                   | Timing preservation   | True         |
| NDA Geometric     | Roll range (pixels)   | [-4, 4]      |
|                   | Rotation (degrees)    | 20           |
|                   | Shear range (degrees) | [-20, 20]    |
| NDA Advanced      | Cutout length         | 14 pixels    |
|                   | Cutout prob.          | 0.3          |
|                   | CutMix $\alpha$       | 1.0          |
|                   | CutMix prob.          | 0.4          |

Table 10: NDA Transform augmentation parameters used in our experiments.

## REFERENCES

- Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in neural information processing systems 31*, 2018.
- Yuhang Li, Youngeun Kim, Hyungseob Park, Tamar Geller, and Priyadarshini Panda. Neuro-morphic data augmentation for training spiking neural networks. In *European Conference on Computer Vision (ECCV)*, pp. 283–300. Springer, 2022.
- Priyadarshini Panda, Sai Aparna Aketi, and Kaushik Roy. Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization. *Frontiers in neuroscience*, 14:653, 2020. doi: 10.3389/fnins.2020.00653.
- Nandan Rathi and Kaushik Roy. Diet-SNN: A low-cost spiking neural network with low-rank-based temporal-step-wise attention. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 6457–6467, 2023.