Inference-Scale Complexity in ANN-SNN Conversion for High-Performance and Low-Power Applications

Supplementary Material

1. Proof for Error Bound

Theorem 1 *The layer-wise conversion error can be divided into intra-layer and inter-layer errors:*

$$e^l \leqslant \overbrace{\|\mathrm{S}(\hat{\boldsymbol{z}}^l;\theta^l) - \mathrm{A}(\hat{\boldsymbol{z}}^l)\|_2}^{\text{intra-layer error}} + \overbrace{\|\boldsymbol{w}^l\|_2 e^{l-1}}^{\text{inter-layer error}}. \tag{S1}$$

Given that both ANN and SNN models receive the same input in the first layer, leading to $e^0 = 0$, the upper bound for the conversion error between ANN and SNN models in an L-layer fully-connected network is given by

$$e_{\text{model}} = e^L \leqslant \sum_{l=1}^{L}\left(\prod_{k=l+1}^{L}\|\boldsymbol{w}^k\|_2\right)\left\|\mathrm{S}(\hat{\boldsymbol{z}}^l;\theta^l) - \mathrm{A}(\hat{\boldsymbol{z}}^l)\right\|_2 \tag{S2}$$

proof 1 (error bound) *According to the definition of the conversion error (Equation (7)), we have*

$$\begin{aligned} e^l &= \|\mathrm{S}(\hat{\boldsymbol{z}}^l) - \mathrm{A}(\boldsymbol{z}^l)\|_2 \\ &= \|\mathrm{S}(\hat{\boldsymbol{z}}^l) - \mathrm{A}(\hat{\boldsymbol{z}}^l) + \mathrm{A}(\hat{\boldsymbol{z}}^l) - \mathrm{A}(\boldsymbol{z}^l)\|_2 \\ &\leqslant \|\mathrm{S}(\hat{\boldsymbol{z}}^l) - \mathrm{A}(\hat{\boldsymbol{z}}^l)\|_2 + \|\mathrm{A}(\hat{\boldsymbol{z}}^l) - \mathrm{A}(\boldsymbol{z}^l)\|_2 \end{aligned} \tag{S3}$$

Here the ANN activation function is defined a s $\mathrm{A}(\cdot) = \mathrm{R}(\cdot) = \mathrm{ReLU}(\cdot)$, where $\mathrm{ReLU}(\cdot)$ is ReLU function. We first prove that $\|\mathrm{R}(\hat{\boldsymbol{z}}^l) - \mathrm{R}(\boldsymbol{z}^l)\|_2 \leqslant \|(\hat{\boldsymbol{z}}^l - \boldsymbol{z}^l)\|_2$. To do so, we analyze four possible cases for z_i^l and $\hat{z}_i^l$, which are individual elements of $\boldsymbol{z}^l$ and $\hat{\boldsymbol{z}}^l$, respectively.

$$\begin{aligned} &\text{if } \hat{z}_i^l \geqslant 0,\ z_i^l \geqslant 0, \\ &\qquad \text{then } (\mathrm{R}(\hat{z}_i^l) - \mathrm{R}(z_i^l))^2 = (\hat{z}_i^l - z_i^l)^2 \\ &\text{if } \hat{z}_i^l \geqslant 0,\ z_i^l \leqslant 0, \\ &\qquad \text{then } (\mathrm{R}(\hat{z}_i^l) - \mathrm{R}(z_i^l))^2 = (\hat{z}_i^l - 0)^2 \leqslant (\hat{z}_i^l - z_i^l)^2 \\ &\text{if } \hat{z}_i^l \leqslant 0,\ z_i^l \geqslant 0, \\ &\qquad \text{then } (\mathrm{R}(\hat{z}_i^l) - \mathrm{R}(z_i^l))^2 = (0 - z_i^l)^2 \leqslant (\hat{z}_i^l - z_i^l)^2 \\ &\text{if } \hat{z}_i^l \leqslant 0,\ z_i^l \leqslant 0, \\ &\qquad \text{then } (\mathrm{R}(\hat{z}_i^l) - \mathrm{R}(z_i^l))^2 = (0 - 0)^2 \leqslant (\hat{z}_i^l - z_i^l)^2 \end{aligned} \tag{S4}$$

Therefore, for each element in vector $\boldsymbol{z}^l$ and $\hat{\boldsymbol{z}}^l$, we can conclude that $\forall i, (\mathrm{A}(\hat{z}_i^l) - \mathrm{A}(z_i^l))^2 \leqslant (\hat{z}_i^l - z_i^l)^2$. From this, we can further derive

$$\|\mathrm{R}(\hat{\boldsymbol{z}}^l) - \mathrm{R}(\boldsymbol{z}^l)\|_2 \leqslant \|(\hat{\boldsymbol{z}}^l - \boldsymbol{z}^l)\|_2. \tag{S5}$$

Back to the main theorem, we further rewrite the conversion error bound as

$$\begin{aligned} e^l &\leqslant \|\mathrm{S}(\hat{\boldsymbol{z}}^l) - \mathrm{A}(\hat{\boldsymbol{z}}^l)\|_2 + \|(\hat{\boldsymbol{z}}^l - \boldsymbol{z}^l)\|_2 \\ &\leqslant \|\mathrm{S}(\hat{\boldsymbol{z}}^l) - \mathrm{A}(\hat{\boldsymbol{z}}^l)\|_2 + \|\boldsymbol{w}^l(\mathrm{S}(\hat{\boldsymbol{z}}^{l-1}) - \mathrm{A}(\boldsymbol{z}^{l-1}))\|_2 \end{aligned} \tag{S6}$$

$$\leqslant \|\mathrm{S}(\hat{\boldsymbol{z}}^l) - \mathrm{A}(\hat{\boldsymbol{z}}^l)\|_2 + \|\boldsymbol{w}^l\|_2\|\mathrm{S}(\hat{\boldsymbol{z}}^{l-1}) - \mathrm{A}(\boldsymbol{z}^{l-1})\|_2 \tag{S7}$$

$$\leqslant \overbrace{\|\mathrm{S}(\hat{\boldsymbol{z}}^l) - \mathrm{A}(\hat{\boldsymbol{z}}^l)\|_2}^{\text{intra-layer error}} + \overbrace{\|\boldsymbol{w}^l\|_2 e^{l-1}}^{\text{inter-layer error}}. \tag{S8}$$

Note that $\|\boldsymbol{w}^l\|_2$ in Equation S7 represents the matrix norm (p=2) or spectral norm of the weight matrix $\boldsymbol{w}^l$, and the derivation from Equation S6 to S7 holds true because of the property of the spectral norms. From the inequality above, we can find that the layer-wise conversion error is bounded by two components: the intra-layer error, which is layer-wise error when both the analog and spiking neurons receive the same input, and the inter-layer error, which is proportional to the layer-wise error in the previous layer.

We further derive the conversion error between models, which corresponds to the conversion error in the last output layer. For simplicity, we define the intra-layer error in each layer as ε^l. According to Equation S8, we get

$$\begin{aligned} e^L &\leqslant \|\mathrm{S}(\hat{\boldsymbol{z}}^L) - \mathrm{A}(\hat{\boldsymbol{z}}^L)\|_2 + \|\boldsymbol{w}^L\|_2 e^{L-1} \\ &= \varepsilon^L + \|\boldsymbol{w}^L\|_2 e^{L-1}. \end{aligned} \tag{S9}$$

Also, since we use direct input coding for SNNs, there is no conversion error in the 0-th layer, and the conversion error in the first layer is given by $e^1 = \varepsilon^1 = \|S(\hat{\boldsymbol{z}}^1) - A(\hat{\boldsymbol{z}}^1)\|_2$. By iteratively applying this relationship across layers, we can derive the error bound for arbitrary layer. The error bound for the final output should be

$$\begin{aligned} e^L &\leqslant \varepsilon^L + \|\boldsymbol{w}^L\|_2 e^{L-1} \\ &\leqslant \varepsilon^L + \|\boldsymbol{w}^L\|_2\varepsilon^{L-1} + \|\boldsymbol{w}^L\|_2\|\boldsymbol{w}^{L-1}\|_2 e^{L-2} \\ &\leqslant \varepsilon^L + \|\boldsymbol{w}^L\|_2\varepsilon^{L-1} + \ldots + \|\boldsymbol{w}^L\|_2 \ldots \|\boldsymbol{w}^2\|_2\varepsilon^1 \\ &= \sum_{l=1}^{L}\left(\prod_{k=l+1}^{L}\|\boldsymbol{w}^k\|_2\right)\varepsilon^l, \quad \left(\textit{Define} \prod_{k=L+1}^{L}\|\boldsymbol{w}^k\|_2 = 1\right) \\ &= \sum_{l=1}^{L}\left(\prod_{k=l+1}^{L}\|\boldsymbol{w}^k\|_2\right)\left\|\mathrm{S}(\hat{\boldsymbol{z}}^l;\theta^l) - \mathrm{A}(\hat{\boldsymbol{z}}^l)\right\|_2 \end{aligned} \tag{S10}$$

2. Proof for Update Rule

The final update rule for the local threshold balancing algorithm at each step is:

$$\Delta\theta^l = -\sum_{i=1}^{N} 2(\hat{z}_i^l - \theta^l)H(\hat{z}_i^l - \theta^l), \tag{S11}$$

$$\theta^l \leftarrow \theta^l - \eta\Delta\theta^l. \tag{S12}$$

proof 2 *As we have mentioned in the main text, our goal is to optimize the following equation:*

$$\forall\, l,\ \arg\min_{\theta^l} \left(\prod_{k=l+1}^{L} \|\boldsymbol{w}^k\|_2\right) \left\|\mathrm{C}(\hat{\boldsymbol{z}}^l;\theta^l) - \mathrm{A}(\hat{\boldsymbol{z}}^l)\right\|_2^2. \tag{S13}$$

We can apply the gradient descent method to iteratively update the threshold value by subtracting the first-order derivative with respect to the threshold, given by:

$$\Delta\theta^l = \frac{\partial \left(\prod_{k=l+1}^{L}\|\boldsymbol{w}^k\|_2\right) \left\|\mathrm{C}(\hat{\boldsymbol{z}}^l;\theta^l) - \mathrm{A}(\hat{\boldsymbol{z}}^l)\right\|_2^2}{\partial\theta^l}. \tag{S14}$$

Considering each element $\hat{z}_i^l$ in the vector $\hat{\boldsymbol{z}}^l$, for each i, we have:

$$\begin{aligned}
&\frac{\partial \left(\prod_{k=l+1}^{L}\|\boldsymbol{w}^k\|_2\right) \left(\mathrm{C}(\hat{\boldsymbol{z}}_i^l;\theta^l) - \mathrm{A}(\hat{\boldsymbol{z}}_i^l)\right)^2}{\partial\theta^l} \qquad \text{(S15)}\\
&= \begin{cases} -\left(\prod_{k=l+1}^{L}\|\boldsymbol{w}^k\|_2\right)\cdot 2(\hat{z}_i^l - \theta^l) & \text{if } \hat{z}_i^l > \theta^l \\ 0 & \text{if } \hat{z}_i^l \leqslant \theta^l \end{cases}\\
&= -\left(\prod_{k=l+1}^{L}\|\boldsymbol{w}^k\|_2\right)\cdot 2(\hat{z}_i^l - \theta^l)H(\hat{z}_i^l - \theta^l)
\end{aligned}$$

Therefore, consider the derivative for θ^l over the whole vector with N elements in total, we have

$$\begin{aligned}
\Delta\theta^l &= \frac{\partial \left(\prod_{k=l+1}^{L}\|\boldsymbol{w}^k\|_2\right) \sum_{i=1}^{N} \left(\mathrm{C}(\hat{\boldsymbol{z}}_i^l;\theta^l) - \mathrm{A}(\hat{\boldsymbol{z}}_i^l)\right)^2}{\partial\theta^l} \qquad \text{(S16)}\\
&= -\left(\prod_{k=l+1}^{L}\|\boldsymbol{w}^k\|_2\right) \sum_{i=1}^{N} 2(\hat{z}_i^l - \theta^l)H(\hat{z}_i^l - \theta^l).
\end{aligned}$$

Since $\left(\prod_{k=l+1}^{L}\|\boldsymbol{w}^k\|_2\right)$ is a constant with fixed weight matrix, we incorporate this term into the learning rate parameter η. Consequently, the final update rule can be derived as:

$$\Delta\theta^l = -\sum_{i=1}^{N} 2(\hat{z}_i^l - \theta^l)H(\hat{z}_i^l - \theta^l), \tag{S17}$$

$$\theta^l \leftarrow \theta^l - \eta\Delta\theta^l. \tag{S18}$$

3. Proof for Pre-Neuron Max pooling Layer

Theorem 2 *The order of max pooling layer and ReLU activation layer does not affect the output results.*

$$\max \mathrm{R}(\boldsymbol{z}) = \mathrm{R}(\max \boldsymbol{z}),\ \text{when}\ \max(\boldsymbol{z}) > 0. \tag{S19}$$

proof 3 *Since $\mathrm{R}(x) = \max(\boldsymbol{x}, 0)$, we can rewrite the left hand side as $\max(\mathrm{R}(\boldsymbol{z})) = \max(\max(\boldsymbol{z}, \boldsymbol{0})) = \max(\boldsymbol{z})$. Similarly, the right-hand side can be written as $\max(\max(\boldsymbol{z}), 0) = \max(\boldsymbol{z})$, which is equal to the left-hand side.*

4. Details for Experiments

4.1. Pseudo-code for Full Conversion Pipeline

In this section, we present the pseudo-code of the full conversion pipeline in Algorithm 1. At the start of the conversion process, the model is initialized by replacing all activation layers with clipping function $\mathrm{C}(\cdot;\boldsymbol{\theta}^l)$ and the initial threshold values $\boldsymbol{\theta}^l$ for each layer are set to 0. Additionally, all max pooling layers are replaced with pre-neuron max pooling layers, and all other modules are set to inference mode.

During local threshold balancing process, input images are sampled from the training dataset at each iteration and fed into the model. The threshold values can be optimized during forward propagation without global backpropagation. After threshold value optimization, the delayed time is calculated by running another iteration of the model with sampled images from the dataset.

The pseudo-code of the SNN inference process with delayed evaluation technique is presented in Algorithm 2. The delayed time is determined based on the given inference time and the estimated delay time. After t_0, the outputs of SNN model are accumulated and the average output value is used as the final prediction.

4.2. Image Classification

When conducting experiments on the ImageNet dataset, we use the pre-trained models from TorchVision. During both the threshold balancing process and inference, we normalize the image to standard Gaussian distribution and Crop the image to size 224×224. The iteration step number of the threshold balancing process is 1000 unless mentioned.

4.3. Semantic Segmentation

For the experiments of Pascal VOC 2012 dataset, the weights of the original ANN models are from open-source Github

Algorithm 1 Efficient ANN-SNN Conversion Algorithm

```
Input:
ANN model pre-trained weight $\boldsymbol{w}$;
Training dataset $\mathcal{D}$;
Iteration steps $K$;
Learning rate $\eta$;
Output:
SNN$(\cdot;\boldsymbol{w},\boldsymbol{\theta})$
Delayed time $t_0$;
 1: // Initialize model
 2: for $l = 1$ to $L$ do
 3:     Set all activation layer as C$(\cdot;\theta^l)$
 4:     Set pre-neuron max-pooling layer
 5:     Set the initial value of threshold $\theta^l = 0$
 6:     Set initial weights for SNN as ANN pre-trained weights $\boldsymbol{w}$
 7: end for
 8:
 9: // Threshold balancing algorithm
10: step = 0
11: while step++ $< K$ do
12:     Sample input images $\boldsymbol{x}^0$ in Dataset $D$
13:     for $l = 1$ to $L$ do
14:         $\boldsymbol{z}^l = \boldsymbol{w}^l\boldsymbol{x}^{l-1}$
15:         $\boldsymbol{x}^l$ = C$(\boldsymbol{z}^l;\theta^l)$
16:         $\Delta\theta^l = -\sum_{i=1}^{N} 2(z_i^l-\theta^l)H(z_i^l-\theta^l)$
17:         $\theta^l \leftarrow \theta^l - \eta\Delta\theta^l$
18:     end for
19: end while
20:
21: // Delayed time calculation
22: Sample input images $\boldsymbol{x}^0$ in Dataset $D$
23: $t_0 = 0$
24: for $l = 1$ to $L$ do
25:     $\boldsymbol{z}^l = \boldsymbol{w}^l\boldsymbol{x}^{l-1}$
26:     $t_0 = t_0 + \left(\theta^l - v^l(0)\right) / \left(\max_i\left(\overline{\mathrm{R}(\boldsymbol{z}^l)}\right)\right)$
27:     $\boldsymbol{x}^l$ = C$(\boldsymbol{z}^l;\theta^l)$
28: end for
29:
30: // Initialize SNN model
31: for $l = 1$ to $L$ do
32:     $\boldsymbol{v}^l(0) \leftarrow \theta^l/2$
33: end for
34: return SNN$(\cdot;\boldsymbol{w},\boldsymbol{\theta}), t_0$
```

repositories. The data preprocessing operations during both the threshold balancing process and inference process include resizing the data into 256×256 image and normalizing the data value. The delayed evaluation step length is set to half of the inference step length. The iteration step number of the threshold balancing process is set to 4 traversals of the training set for FCN and 5 traversals of the training set for DeepLab. Moreover, Pascal VOC 2012 is augmented by the extra annotations provided by SBD, resulting in 10582 training images.

Algorithm 2 SNN Inference with Delayed Evaluation Strategy

```
Input:
SNN model SNN$(\cdot;\boldsymbol{w},\boldsymbol{\theta})$;
Input Image $\boldsymbol{x}^0$;
Inference steps $T$;
Delayed time $t_0$;
Output:
Prediction $\boldsymbol{o}$;
 1: // SNN inference
 2: step = 0
 3: if $t_0 > T-4$ then
 4:     $t_0 = T-4$
 5: end if
 6: $\boldsymbol{o} = \mathbf{0}$
 7: for $t = 1$ to $T$ do
 8:     for $l = 1$ to $L$ do
 9:         $\boldsymbol{z}^l = \boldsymbol{w}^l\boldsymbol{x}^{l-1}$
10:         $\boldsymbol{x}^l$ = S$(\boldsymbol{z}^l;\theta^l)$
11:     end for
12:     if $t > t_0$ then
13:         $\boldsymbol{o} = \boldsymbol{o} + \boldsymbol{x}^l$
14:     end if
15: end for
16: $\boldsymbol{o} = \boldsymbol{o}/(T-t_0)$
17:
18: // Reset SNN model
19: for $l = 1$ to $L$ do
20:     Reset S$(\cdot;\theta^l)$
21:     $\boldsymbol{v}^l(0) \leftarrow \theta^l/2$
22: end for
23: return $\boldsymbol{o}$
```

For the experiments of the MS COCO 2017 dataset, the weights are directly downloaded from TorchVision. When performing data preprocessing, We first resize the input data into 400×400 images and normalize the images. The iteration step number of the threshold balancing process is set to 3 traversals of the training set. Note that these weights were trained on a subset of MS COCO 2017, using only the 20 categories that are present in the Pascal VOC dataset. This subset contains 92518 images for training.

4.4. Object Detection

For our object detection experiments, we utilized pre-trained weights from TorchVision. During the threshold balancing process, we use similar dataset augmentation as SSD [2]. The input images are augmented by RandomPhotometricDistort, RandomZoomOut, RandomIoUCrop, and RandomHorizontalFlip. The iteration steps are set to 5000 for each model. During the evaluation of converted models, we directly use normalized images as inputs.

4.5. Video Classification

The pre-trained weights for video classification tasks are directly downloaded from TorchVision. We split the original validation set of Kinetics-400 into a new training set and a new test set. The resulting training set contains 12000 videos and the new test set contains 7881 videos. The accuracies are estimated on video-level with parameters frame_rate=15, clips_per_video=5, and clip_len=16. The frames are resized to 128×171, followed by a central crop resulting into 112×112 normalized frames.

5. Visualizations on Object Detection and Semantic Segmentation

In Figure S1 and Figure S2 we present the visualization of the semantic segmentation and object detection results. In each row of the figure, we illustrate the visualization of ground truth, original image (only for semantic segmentation), results from original ANN and results from converted SNN at different time-steps.

6. Energy Consumption Analysis

Since the low-power consumption is one of the advantages of SNNs, we calculate the average energy consumption of the converted SNNs and compare it with the energy consumption of the ANN counterparts. We employed a method similar to previous work, estimating the energy consumption of the SNN by calculating the number of Synaptic Operations (SOP). Since the total spike activity of the SNN increases proportionally with the inference time, we define SOP90 and SOP95 as the metrics for converted SNNs on ImageNet. The SOP90/95 denotes the average synaptic operation per image when the accuracy of the converted SNN exceeds 90%/95% of the original ANN while SNN90-FPS/W denotes the total number of frames per joule when the performance of the converted SNN exceeds 90%. In order to further estimate the energy consumption, we utilize the average energy efficiency of 77fJ/SOP for SNN and 12.5pJ/FLOP for ANN [3] to calculate the required energy for one frame. The detailed comparison is demonstrated in the table below. For

Architecture	SOPs90	SOPs95	ANN-FPS/W	SNN90-FPS/W
ResNet-34	20.86	33.42	22	622

Table S1. Energy consumption estimation on ImageNet dataset

ImageNet classification tasks, using the same ResNet-34 architecture, the SNN is over 28 times more energy efficient than the original ANN while maintaining 90% performance of the original ANN, achieving an estimation of 622 FPS/W energy efficiency while deployed on neuromorphic hardware. It is worth noticing the SNN can be easily obtained by converting open-source pre-trained ANN models with negligible training cost and then deploy on specific hardware for energy-saving purpose.

7. Detailed Discussion on Training Cost

Besides general discussion of the inference-scale complexity of the overall conversion framework, we also demonstrate the efficiency of our method by comparing the total time required for three different post-training conversion methods, including LCP, ACP [1], and our method. For all methods, we used the same environment with a 4090 GPU for the evaluation. Here we present the results using ResNet-34 architecture on ImageNet in the below table.

Method	T=32	T=64	T=128	T=256	T=512
Ours	92.73	92.73	92.73	92.73	92.73
LCP	70.23	114.19	201.82	376.98	727.66
ACP	829.49	1218.93	2018.56	3731.53	7059.09

Table S2. Comparison of conversion time with different post-training methods

It is worth noting that one only needs to run the threshold optimization algorithm once in our method and the obtained SNN can be applied with any simulation time-steps. While in LCP/ACP, calibration is required for every simulation steps. Therefore, in the table above, our algorithm only takes 93 seconds to get the converted SNN and is applicable to any time-steps. Although LCP has an advantage at 32 steps, the required conversion time still increases with the total time-steps and loses its advantage at 64 steps. Moreover, ACP takes much more time because it involves weight updates. This is because our method only requires a similar training cost as ANN inference, which is lower than the computational cost of SNN inference and weight update required in LCP/ACP.

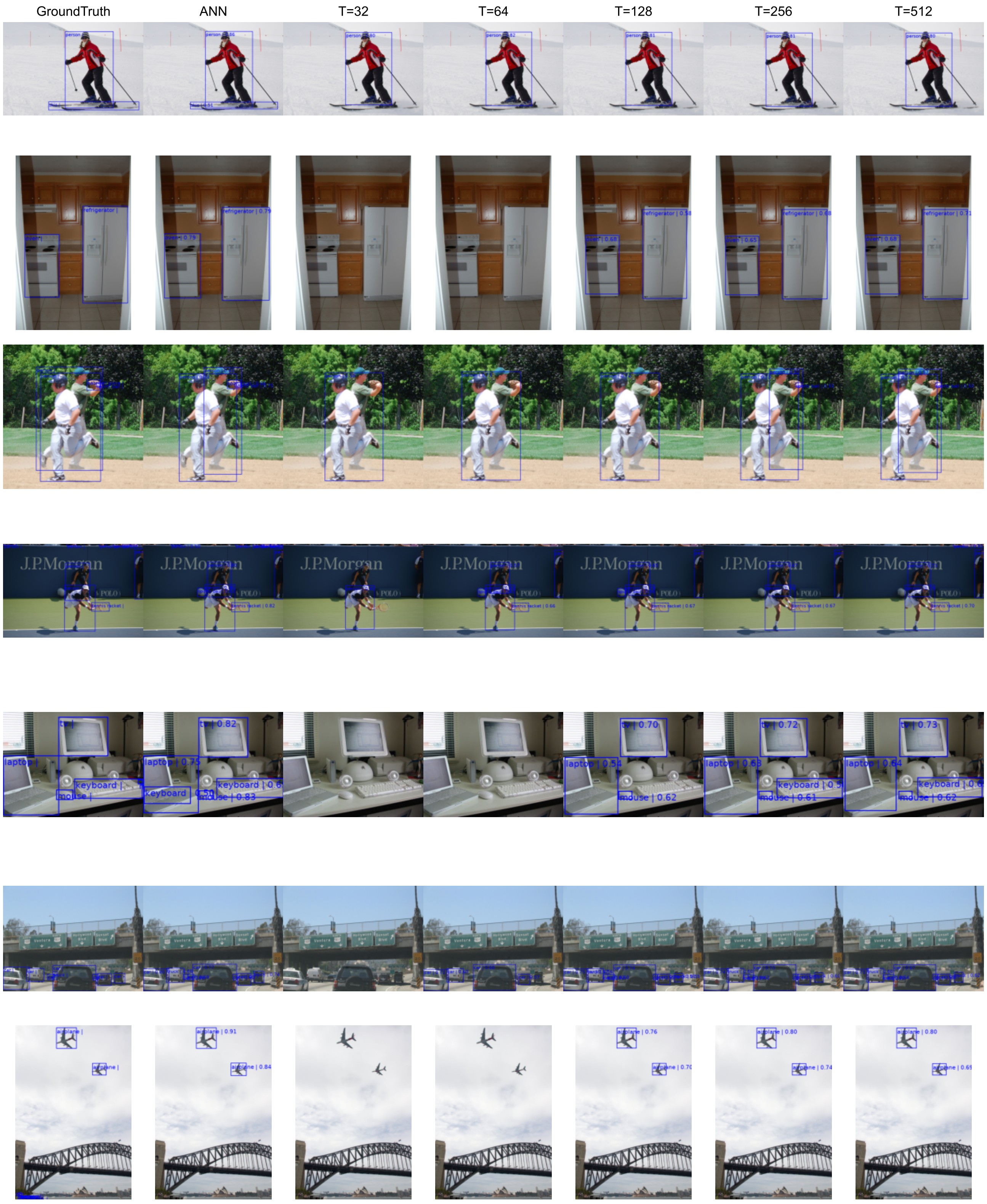

Figure S1. Illustration for detection examples of SNNs on different inference steps

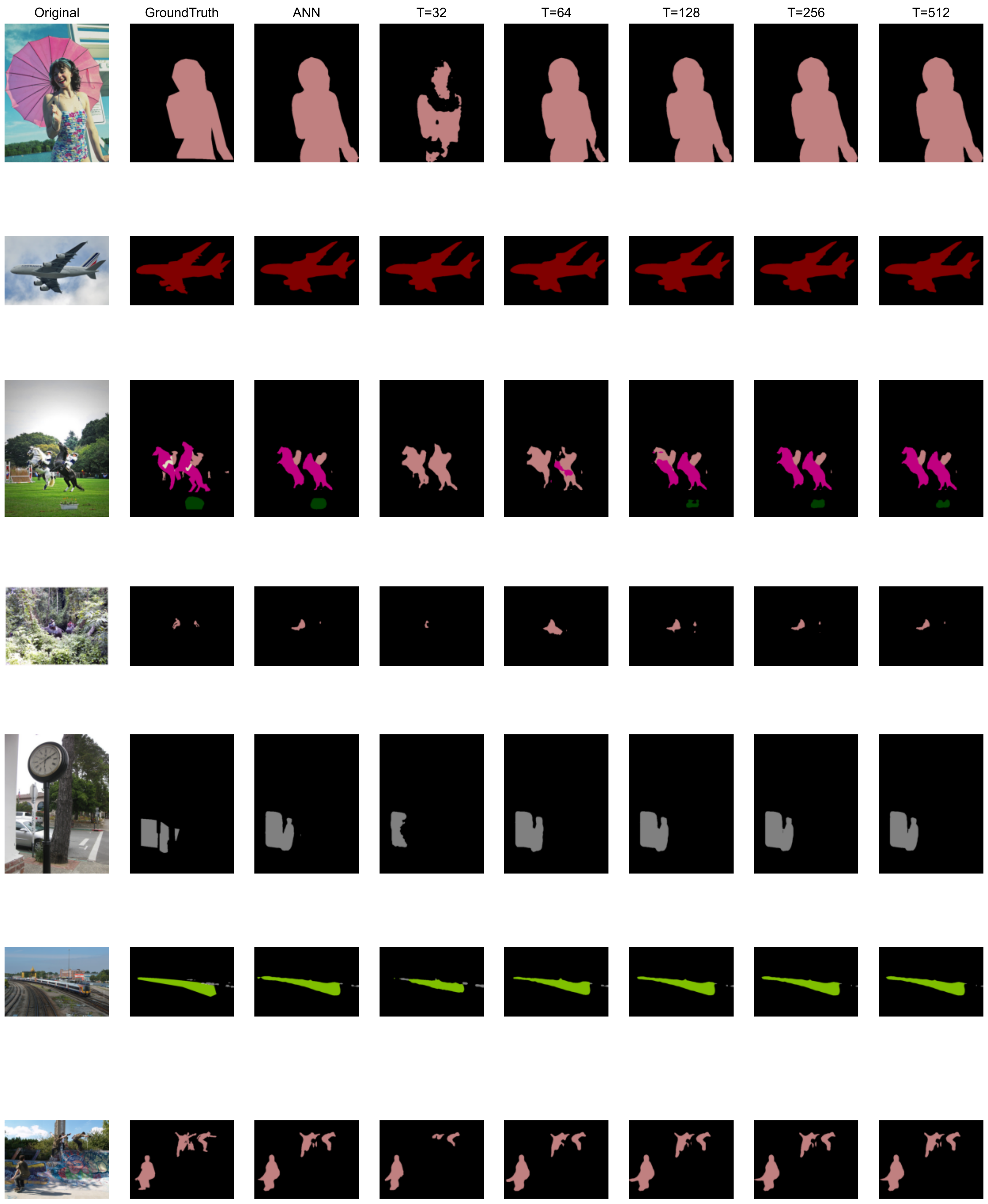

Figure S2. Illustration for segmentation examples of SNNs on different inference steps

References

[1] Yuhang Li, Shikuang Deng, Xin Dong, and Shi Gu. Error-aware conversion from ann to snn via post-training parameter calibration. *International Journal of Computer Vision*, 2024. 4

[2] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *European Conference on Computer Vision*, 2016. 3

[3] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Frontiers in Neuroscience*, 2015. 4