1. Additional Figures with Other Settings

Figure 1 illustrates the upper-bound performance of SNNs across all eight settings.

2. Visualization of Dynamic Strategies

Dynamic strategies can be visualized in a three-dimensional space with three axes representing the number of layers, the number of channels, and precision, see Figure 2. The majority of research into dynamic strategies, such as BlockDrop [20], feature boosting and suppression [8], and Dynamic Convolution [1], dynamically skip either layers (see Figure 2.b.) or channels (see Figure 2.c.), optimizing dimensions one and two respectively.

Some work on dynamic strategies (see Figure 2.d.) in dimension three, the optimization of precision at runtime, has been carried out. Song et al. proposed dynamic region-based quantization (DRQ) in which a predictor classified regions in an input image as requiring 8 bit or 4 bit weights [19]. Huang et al. present a similar method in which the less significant bits of an activation representation can be dynamically masked to allow on-the-fly changes to the multiplications carried out [14]. So far dynamic strategies for precision are based on selecting a precision for a certain value from a discrete set of integer precisions. This is in contrast with the work presented in this paper where the effective activation precision can be varied smoothly based on the inference confidence by virtue of using an SNN model.

3. Spiking Neurons and ANN-to-SNN Conversion

The neuronal dynamics of integrate-and-fire neuron in SNNs is controlled by these two equations below:

$$\boldsymbol{u}_t^l = \boldsymbol{u}_{t-1}^l(1 - \boldsymbol{z}_{t-1}^l) + \widetilde{\boldsymbol{W}}^l \boldsymbol{z}_t^{l-1} + \widetilde{\boldsymbol{B}}^l \tag{1}$$

$$\boldsymbol{z}_t^l = \Theta(\boldsymbol{u}_t^l - \boldsymbol{th}^l) \tag{2}$$

In these equations, $\boldsymbol{u}_t^l$ is the membrane potential of spiking neurons in layer l at time t. $\boldsymbol{u}_{t-1}^l$ is the membrane potential at the previous time step $t-1$. $\widetilde{\boldsymbol{W}}^l$ is the weight and $\widetilde{\boldsymbol{B}}^l$ is the bias in layer l. $\Theta(\cdot)$ denotes the Heaviside step function, and it returns 1 if the value in the bracket is higher than zero; otherwise it returns 0. $\boldsymbol{th}^l$ is the threshold in layer l, and $\boldsymbol{z}_t^l$ is the output spike in this layer at time t.

ANN-to-SNN conversion is using spiking neurons to achieve the computations that happen in artificial neurons. The information processing in the artificial neurons in layer l is

$$\boldsymbol{y}^l = \sigma(\boldsymbol{W}^l \boldsymbol{y}^{l-1} + \boldsymbol{B}^l) \tag{3}$$

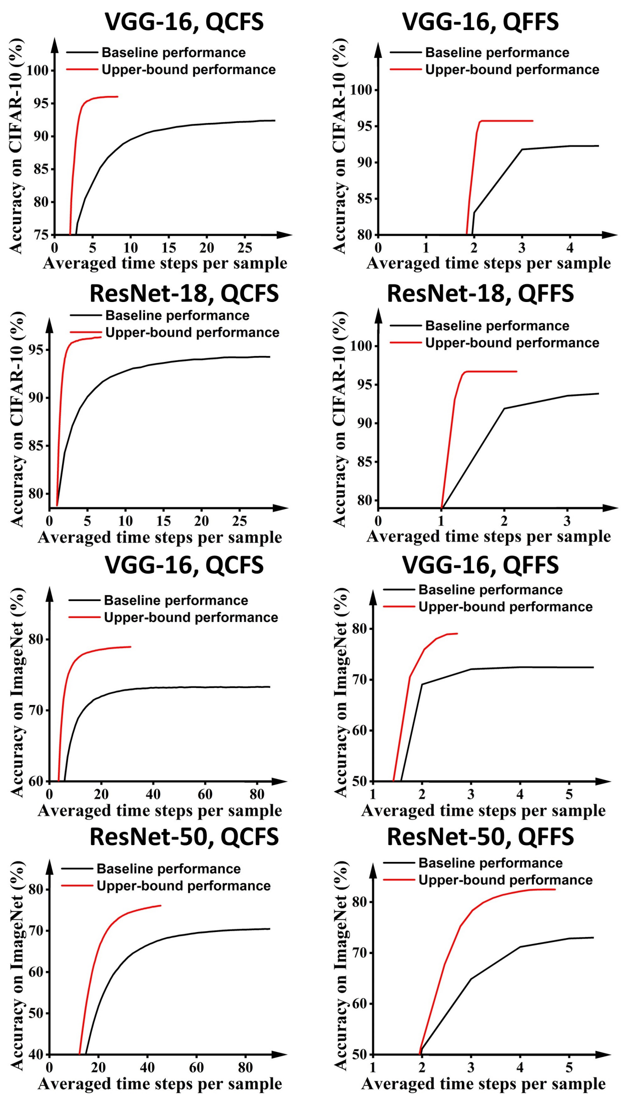

Figure 1. The upper-bound performance of SNNs when fully utilizing dynamic strategies at runtime, as shown by the red curves. The black curves represent baseline SNN performance without dynamic strategies.

where $\sigma(\cdot)$ is the ReLU activation function, $\boldsymbol{W}^l$ and $\boldsymbol{B}^l$ denote the weight and the bias in layer l, $\boldsymbol{y}^l$ is the output of layer l, and $\boldsymbol{y}^{l-1}$ is the output in the previous layer.

The original method to conduct ANN-to-SNN conversion is called data-based normalization [4], where the SNN parameters are calculated by

$$\widetilde{\boldsymbol{W}}^l = \frac{\lambda^{l-1}\boldsymbol{W}^l}{\lambda^l} \tag{4}$$

$$\widetilde{\boldsymbol{B}}^l = \frac{\boldsymbol{B}^l}{\lambda^l} \tag{5}$$

$$\boldsymbol{th}^l = 1 \tag{6}$$

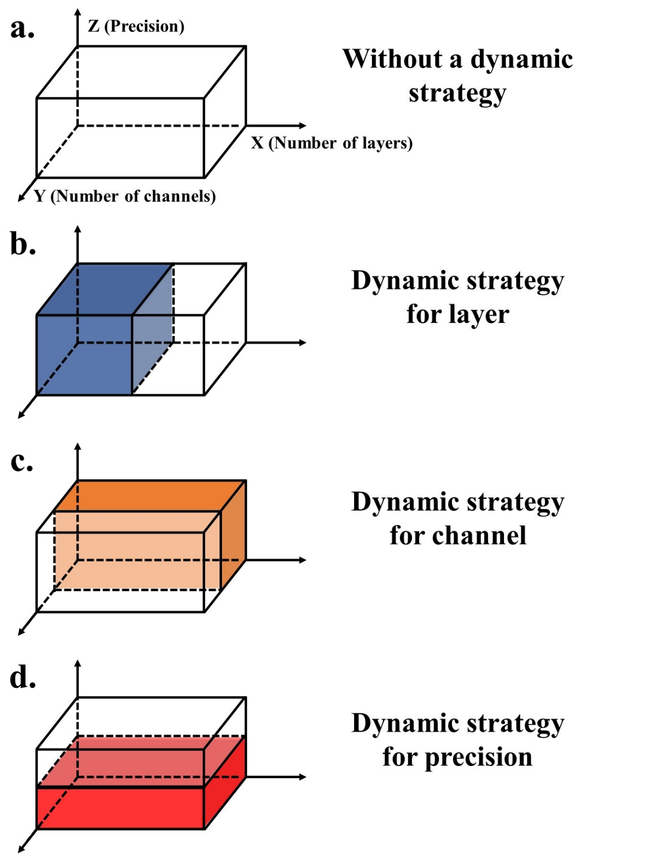
a.
Z (Precision)
X (Number of layers)
Y (Number of channels)
Without a dynamic strategy
b.
Dynamic strategy for layer
c.
Dynamic strategy for channel
d.
Dynamic strategy for precision

Figure 2. The visualization of dynamic strategies in a three-dimensional space. The cuboid in **a.** represents a neural network model without applying any dynamic strategies. For each example, the whole model capability will be used to generate outputs. **b. c. d.** apply dynamic strategies in the dimension of layer, channel, and precision, respectively. These dynamic strategies can adapt to different examples. Instead of the full model capacity, only a portion of computational resources, just enough to generate reliable outputs (The colored part of the cuboid), will be allocated for a given example. Each example has heterogeneous resource allocation.

In these equations, λ^l and λ^{l-1} are the maximum ANN activation value in layer l and the previous layer $l-1$ respectively.

4. LSQ

LSQ, or Learned Step Size Quantization [6], is a method to build low-precision ANNs. Compared with other quantization-aware training approaches, the quantization scale factors in LSQ are trainable, which benefits to better mapping from full-precision values to quantized values.

A quantization operator is defined by

$$\hat{v} = s\lfloor clip(\frac{v}{s}, -Q_N, Q_P)\rceil, \tag{7}$$

where v is the full-precision value before quantization, s is a quantization scale factor. $clip(z, -Q_N, Q_P)$ clips z to the range of $(-Q_N, Q_P)$. For example, in 2-bit activation quantization, this range is $(0, 3)$. $\lfloor z\rceil$ is rounding operation that rounds z to the nearest integer.

During training, the gradient of $\frac{\partial \hat{v}}{\partial v}$ is defined by applying straight-through estimators [18], which is

$$\frac{\partial \hat{v}}{\partial v} = \begin{cases} 1 & \text{if } -Q_N < \frac{v}{s} < Q_P \\ 0 & \text{if } \frac{v}{s} \leq -Q_N \\ 0. & \text{if } \frac{v}{s} \geq Q_P \end{cases} \tag{8}$$

The gradient of $\frac{\partial \hat{v}}{\partial s}$ is

$$\frac{\partial \hat{v}}{\partial s} = \begin{cases} -\frac{v}{s} + \lfloor \frac{v}{s} \rceil & \text{if } -Q_N < \frac{v}{s} < Q_P \\ -Q_N & \text{if } \frac{v}{s} \leq -Q_N \\ Q_P, & \text{if } \frac{v}{s} \geq Q_P \end{cases} \tag{9}$$

which is also calculated by applying straight-through estimators. The detailed calculations are shown below.

$$\begin{aligned} \frac{\partial \hat{v}}{\partial s} &= s\lfloor \frac{v}{s} \rceil \qquad\qquad\qquad \text{if } -Q_N < \frac{v}{s} < Q_P \\ &= s^{'}\lfloor \frac{v}{s} \rceil + s\lfloor \frac{v}{s} \rceil^{'} \\ &= \lfloor \frac{v}{s} \rceil + s(\lfloor \cdot \rceil^{'} (\frac{v}{s})^{'}) \\ &= \lfloor \frac{v}{s} \rceil + s(\frac{v}{s})^{'} \\ &= \lfloor \frac{v}{s} \rceil + s(-\frac{v}{s^2})^{'} \\ &= \lfloor \frac{v}{s} \rceil - \frac{v}{s} \end{aligned} \tag{10}$$

5. QCFS and QFFS

In this Section, we first formulate the target ANN computations to be simulated in SNNs by QCFS and QFFS. After replacing ReLU in Equation 3 with the activation quantization operator defined in Equation 7, the computation in a layer of ANN becomes

$$\boldsymbol{y}^l = s^l \lfloor clip(\frac{\boldsymbol{W}^l \boldsymbol{y}^{l-1} + \boldsymbol{B}^l}{s^l}, -Q_N, Q_P)\rceil. \tag{11}$$

Our experiments use 2-bit activation quantization where Q_N is 0 and Q_P is 3, so this equation becomes

$$\boldsymbol{y}^l =^l \lfloor clip(\frac{\boldsymbol{W}^l \boldsymbol{y}^{l-1} + \boldsymbol{B}^l}{s^l}, 0, 3)\rceil. \tag{12}$$

QCFS. The spiking neuronal model used in QCFS is the soft-reset integrate-and-fire neuron, which is

$$\boldsymbol{u}_t^l = \boldsymbol{u}_{t-1}^l + \widetilde{\boldsymbol{W}}^l \boldsymbol{z}_t^{l-1} \boldsymbol{th}^{l-1} + \widetilde{\boldsymbol{B}}^l - \boldsymbol{z}_{t-1}^l \boldsymbol{th}^l, \tag{13}$$

Table 1. Latency advantages brought by Dynamic Confidence in 8 different experimental settings. The accuracy is compromised for better latency.

Dataset	Architecture	Method	Acc(ANN)(%)	Acc(SNN)(%)	Averaged time steps	Latency saving(%)
CIFAR-10	VGG-16	QCFS	92.41	92.31	27	
		QCFS + Dynamic Confidence	92.41	92.31	10.85	60%
		QFFS	92.41	92.31	6	
		QFFS + Dynamic Confidence	92.41	92.31	2.87	52%
	ResNet-18	QCFS	93.79	94.00	19	
		QCFS + Dynamic Confidence	93.79	**94.00**	8.03	63%
		QFFS	93.79	93.79	4	
		QFFS + Dynamic Confidence	93.79	93.79	**2.16**	46%
ImageNet	VGG-16	QCFS	72.40	73.00	33	
		QCFS + Dynamic Confidence	72.40	**73.00**	27.90	15%
		QFFS	72.40	72.52	4	
		QFFS + Dynamic Confidence	72.40	72.40	**2.78**	31%
	ResNet-50	QCFS	72.60	70.50	94	
		QCFS + Dynamic Confidence	72.60	70.50	67.22	28%
		QFFS	72.60	72.56	6	
		QFFS + Dynamic Confidence	72.60	72.56	3.79	37%

$$\boldsymbol{z}_t^l = \Theta(\boldsymbol{u}_t^l - \boldsymbol{th}^l). \tag{14}$$

There are three parameters ($\widetilde{\boldsymbol{W}}$, $\widetilde{\boldsymbol{B}}^l$, and $\boldsymbol{th}^l$) that need to be calculated according to ANN parameters. The equations are provided below:

$$\widetilde{\boldsymbol{W}}^1 = \boldsymbol{W}^l \tag{15}$$

$$\widetilde{\boldsymbol{B}}^1 = \boldsymbol{B}^l \tag{16}$$

$$\boldsymbol{th}^l = 3s^l. \tag{17}$$

Note that, unlike QFFS, QCFS does not have a maximum spike count limitations per neuron (Z_max in Equation 19) to model the clipping operation on Q_P in the target ANN (Equation 12). As a result, after the simulation of QFFS stops, QCFS can continue to simulate and at a time point, its accuracy will surpass that of QFFS.

QFFS. The spiking neuronal model used in QFFS is the soft-reset integrate-and-fire neuron with negative spikes, which is defined by the equations below:

$$\boldsymbol{u}_t^l = \boldsymbol{u}_{t-1}^l + \widetilde{\boldsymbol{W}}^l \boldsymbol{z}_t^{l-1} \boldsymbol{th}^{l-1} + \widetilde{\boldsymbol{B}}^l - \boldsymbol{z}_{t-1}^l \boldsymbol{th}^l \tag{18}$$

$$\boldsymbol{z}_t^l = \Theta(\boldsymbol{u}_t^l - \boldsymbol{th}^l)\Theta(Z_{max} - \boldsymbol{Z}_{t-1}^l) - \Theta(-\boldsymbol{u}_t^l)\Theta(\boldsymbol{Z}_{t-1}^l) \tag{19}$$

$$\boldsymbol{Z}_t^l = \boldsymbol{Z}_{t-1}^l + \boldsymbol{z}_t^l. \tag{20}$$

Z_{max} is the maximum spike count limitation, and $\boldsymbol{Z}_t^l$ records the accumulated spike counts. The ANN-to-SNN conversion is achieved according to the equations below:

$$\widetilde{\boldsymbol{W}}^l = \boldsymbol{W}^l \tag{21}$$

$$\widetilde{\boldsymbol{B}}^l = \boldsymbol{B}^l \tag{22}$$

$$\boldsymbol{th}^l = 3s^l \tag{23}$$

$$Z_{max} = 3 \tag{24}$$

6. Configuration Overhead of Dynamic Confidence

The configurations of Dynamic Confidence have three steps as illustrated in main sections. In step 1 an ANN model is run for one epoch on a small valid set for calibration, this cost is about 600 times lower than the whole training cost and it can be faithfully ignored. Step 2 calculates a scale factor α to use in inference, which does not bring any overhead. In step 3 the confidence threshold th_c is calculated by Pareto Front. Generally, Pareto Front can be very expensive since Pareto Front replies on high-resolution search space to ensure the performance of the Pareto-optimal one. However, Dynamic Confidence is very power efficient as our proposed method is highly robust to the search resolution of th_c when applying Pareto Front. This robustness may come from step 2 which softens the confidence value before computing Pareto Front. Second, we emphasize that even applying a high search resolution of th_c such as 0.0001, the computational complexity of the Pareto Front is still trivial, only if exploiting the monotonicity of thresholded classifications, similar to the trick used when calculating ROC curve in machine learning[7]. Specifically, any input samples that are terminated by Dynamic Confidence with respect to a given th_c will be terminated for all lower thresholds. Leveraging this property, a high-search-resolution Pareto Front can be created by running an SNN once on a valid set, and then, on the collected SNN outputs conducting a linear scan of th_c (which can be done in a few seconds on a laptop). In summary, the main overhead of Dynamic Confidence at the configuration phase is running an ANN and an SNN on a valid set, which is trivial and can be ignored.

7. More Results with Compromised Accuracy

Table 1 displays the results of compromising the accuracy of SNNs to achieve better latency.

8. More Related Work

Model Compression. In order to deploy neural network algorithms on resource-constrained edge devices and to meet the edge restraints on model size, inference latency, and power cost, several model compression techniques have been proposed such as quantization [10, 2, 16, 6]; structured and unstructured pruning [11, 10]; compact network architecture design [12, 13, 15]; neural architecture search (NAS) [5, 17]; and algorithm-hardware co-design [9].

The model compression techniques introduced above optimize the neural network model in an offline manner and will not adapt to inputs during execution. This paper, on the other hand, is focused on runtime dynamic optimization methods (also called dynamic strategies or adaptive computation) that can choose different computing strategies for different input images. Our method is orthogonal to these model compression techniques and could be combined with them to further improve SNN performance.

9. More Discussions

Except for Pareto Front, th_c can be calculated by such as applying reinforcement learning or SGD learning (SGD learning needs to define the exact form of the gradients of th_c, where surrogate gradients may be desirable). These methods may be more effective to find better thresholds, but this paper emphasizes the efficiency of our method, so does not involve learning and keep it post-hoc.

Rate-coding is generally considered inefficient in its use of spikes [3]. However, the method we have developed exploits the smoothly variable precision available with rate-coding to process incoming data incrementally, opening the door to saving compute resources by early exiting from the inference pipeline. The efficiency of this kind of approach in terms of the number of operations is dependent on the number of input examples that are 'easy' and allow for early exiting. In other words, Applying Dynamic Confidence on less challenging datasets such as MNIST and CIFAR-10 can render greater improvements than on ImageNet.

In applications where model input data are streams of events directly from sensors, such as with event camera input or event-based scintillation detectors, Dynamic Confidence can provide a reduction in the latency between event generation and model output. This makes Dynamic Confidence a particularly attractive candidate algorithm for exploration in low-power low-latency edge computing.

References

[1] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11030–11039, 2020. 1

[2] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015. 4

[3] Simon Davidson and Steve B Furber. Comparison of artificial and spiking neural networks on digital hardware. *Frontiers in Neuroscience*, 15:651141, 2021. 4

[4] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. ieee, 2015. 1

[5] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019. 4

[6] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019. 2, 4

[7] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006. 4

[8] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*, 2018. 1

[9] Song Han. *Efficient methods and hardware for deep learning*. PhD thesis, Stanford University, 2017. 4

[10] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 4

[11] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017. 4

[12] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 4

[13] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 4

[14] Kai Huang, Bowen Li, Dongliang Xiong, Haitian Jiang, Xiaowen Jiang, Xiaolang Yan, Luc Claesen, Dehong Liu, Junjian Chen, and Zhili Liu. Structured Dynamic Precision for Deep Neural Networks Quantization. *ACM Transactions on Design Automation of Electronic Systems*, page 3549535, July 2022. 1

[15] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 4

[16] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256kb memory. *arXiv preprint arXiv:2206.15472*, 2022. 4

[17] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018. 4

[18] Emre O Neftci, Charles Augustine, Somnath Paul, and Georgios Detorakis. Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in neuroscience*, 11:324, 2017. 2

[19] Zhuoran Song, Bangqi Fu, Feiyang Wu, Zhaoming Jiang, Li Jiang, Naifeng Jing, and Xiaoyao Liang. DRQ: Dynamic Region-based Quantization for Deep Neural Network Acceleration. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 1010–1021, May 2020. 1

[20] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8817–8826, 2018. 1