MORPHING WITH SPIKES

Anonymous authors
Paper under double-blind review

ABSTRACT

Net2net techniques are used to increase the capacity of ANNs without any loss in accuracy during training. This paper extends those techniques to spiking neural networks. We show that net2widernet is exact on SNNs but net2deepernet is not when the inserted layers receive non-binary inputs. We also provide a method of mitigating the accuracy loss introduced by net2deepernet on spiking networks that optimizes the inserted layer's gain via gridsearch. We demonstrate both aforementioned claims with experiments on image classification datasets CIFAR-10, CIFAR-100, CIFAR-10-DVS, and the RL task Cartpole, where our gain-searched spiking morphism techniques achieve substantially higher prediction agreement than raw insertion, allowing future work to build stronger and more efficient neural architecture search algorithms for spiking neural networks.

1 INTRODUCTION

Spiking neural networks (SNNs) utilize discrete spike events rather than continuous-valued activations, as used by classical artificial neural networks (ANNs) (Maass, 1997). Due to their event-driven nature, SNNs are able to perform significantly more efficiently, especially when used on specialized neuromorphic processors (Indiveri and Liu, 2015). Since computation occurs only when spike events are generated, SNNs avoid the dense activations of classical ANNs, leading to improved energy efficiency on neuromorphic platforms (Davies et al., 2018).

However, the search and design of these spiking architectures remains a tedious task. For classical artificial neural networks (ANNs), neural architecture search has advanced from brute force RL methods (Zoph & Le, 2017) to much more elegant methods like differentiable searches (Liu et al., 2019a), supernet searches (Pham et al., 2018), and more. One method in particular, network morphism (Chen et al., 2016; Cai et al., 2018), allows the growth of an ANN while preserving the loss, so it doesn't require retraining from scratch. Chen et al. (2016) developed ways to deepen and widen ANNs with identity layers and neuron duplication, respectively. To deepen an ANN while preserving the network loss, network morphism inserts a layer with the weight matrix initialized as the identity matrix, meaning the output of the inserted layer is identical to the output of the layer previous to it. To widen a given layer, network morphism duplicates an existing neuron and halves the output weights, meaning the layer downstream still receives an identical input, preserving the function. However, this duplication creates symmetry that is unfavorable for subsequent training, as the duplicates' gradients are identical. Further innovations like Gradmax (Evci et al., 2022) introduce function preserving growth while maximizing the gradient norms of the added neurons in widened layers as well, benefiting subsequent training dynamics.

The non-differentiable spike activation function, which requires surrogate gradients to enable back-propagation through time (Neftci et al., 2019) complicates Net2Net-style methods. The closest existing work to architecture growth in SNNs instead relies on heuristic, random neuron growth for continual learning settings (Han et al., 2023), rather than function-preserving or gradient-optimal growth operators. We are not aware of any existing work that provides a principled, theoretically grounded approach to growing spiking architectures analogous to Net2Net or GradMax for ANNs.

Thus, we propose an extension of network morphism to spiking neural networks. Specifically, our contributions are the following:

- For widening layers, we provide theoretical proof that the widening operation is exact.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

- For deepening, we show that it is exact only when input spikes are binary. We also show that when layers are inserted after average pooling layers, the inserted layers cannot replicate the non-binary input, so net2deepernet is no longer exact.
- To combat the non-preserving deepening after pooling, we introduce a workaround that scales the inserted identity kernel by a gain searched directly against prediction agreement in the spiking network, decreasing the loss from the deepen significantly.
- We provide a further extension of our methods to local learning methods like reward-modulated spike timing dependent plasticity (RSTDP).
- We provide empirical validation on image classification datasets CIFAR-10, CIFAR-100, CIFAR-10-DVS, and the RL task Cartpole.

2 METHODOLOGY

2.1 NETWORK TYPES

We test our spiking morphism techniques on two types of SNNs, namely leaky-integrate-and-fire (LIF) neuron networks and adaptive-leaky-integrate-and-fire (ALIF) networks. Every net is a stack of conv layers $l = 0, \dots, L-1$ (BatchNorm in some, applied in eval mode with frozen running statistics), each followed by a spiking neuron layer, run for T time steps over a constant, rate-coded input current. Let $\hat{z}_{l,c}^{(t)}$ denote channel c's post-BN pre-activation current into layer l's neurons at step t, and $s_{l,c}^{(t)} \in \{0,1\}$ its output spike.

2.1.1 LIF SNN

Each channel is a leaky integrate-and-fire neuron with a hard reset according to the following equations:

$$u_{l,c}^{(t)} = \alpha\, v_{l,c}^{(t-1)} + \hat{z}_{l,c}^{(t)}, \qquad s_{l,c}^{(t)} = \left[u_{l,c}^{(t)} \geq v_{th}\right], \qquad v_{l,c}^{(t)} = u_{l,c}^{(t)}\left(1 - s_{l,c}^{(t)}\right),$$

with $[\cdot] = 1$ if the enclosed condition holds, 0 otherwise and $v_{l,c}^{(0)} = 0$, leak α, and threshold v_{th} shared by every channel in the layer. We use a fast-sigmoid surrogate gradient on the backward pass (Neftci et al., 2019) to train the networks with $v_{th} = 1$ throughout.

2.1.2 ALIF SNN

The adaptive-threshold variant replaces the fixed v_{th} with a threshold that rises with recent spiking activity, in the style of Bellec et al. (2018):

$$b_{l,c}^{(t)} = \rho\, b_{l,c}^{(t-1)} + s_{l,c}^{(t-1)}, \qquad \theta_{l,c}^{(t)} = v_{th} + \beta_{\text{adapt}}\, b_{l,c}^{(t)}, \qquad s_{l,c}^{(t)} = \left[u_{l,c}^{(t)} \geq \theta_{l,c}^{(t)}\right],$$

with $b_{l,c}^{(0)} = 0$, decay $\rho = e^{-1/\tau}$, and $u_{l,c}^{(t)}$ as above. ρ and β_{adapt} are again shared by every channel in the layer, and $b_{l,c}^{(t)}$ depends on the previous spike $s_{l,c}^{(t-1)}$.

2.2 WIDEN

Net2WiderNet (Chen et al., 2016) widens a layer by duplicating units and dividing each duplicate's outgoing weights by its replication count, so the next layer's input is unchanged. We show the same construction is exactly function preserving for an SNN built from the two types of neurons of Section 2.1.

To widen conv layer l from C_l to C_l' output channels ($C_l' > C_l$), we introduce a random mapping function

$$g : \{1, \dots, C_l'\} \to \{1, \dots, C_l\}, \qquad g(j) = \begin{cases} j & j \leq C_l \\ \text{random sample from } \{1, \dots, C_l\} & j > C_l. \end{cases}$$

We introduce new weight tensors W_l' and W_{l+1}' (and, for BatchNorm, gathered $\gamma_l', \beta_l', \mu_l', \sigma_l'^2$) representing the weights of these layers in the widened network. Then the new weights are given by

$$W_l'[j] = W_l[g(j)], \qquad W_{l+1}'[:, j] = \frac{1}{|\{x : g(x) = g(j)\}|}\, W_{l+1}[:, g(j)].$$

The first C_l channels of W_l are copied directly into W'_l; channels C_l+1 through C'_l are chosen by g, with replacement, so a source channel may be duplicated many times. The corresponding slice of W'_{l+1} divides by the replication factor $n_{g(j)} := |\{x : g(x) = g(j)\}|$, so that summed back together the duplicated channels contribute exactly as much as the single original channel did.

Widening layer l leaves its input untouched, so duplicate channel j and its source $g(j)$ receive the identical filter and BN parameters on the identical input current, hence the identical pre-activation $\hat{z}'^{(t)}_{l,j} = \hat{z}^{(t)}_{l,g(j)}$ at every t. Both start from the same zero state and evolve under the same neuron equations (Section 2.1), so their membrane traces and spikes stay equal at every step: $s'^{(t)}_{l,j} = s^{(t)}_{l,g(j)}$.

Layer $l+1$'s pre-activation current is then, grouping the sum by source channel $c = g(j)$,

$$z'^{(t)}_{l+1} = \sum_{j=1}^{C'_l} \frac{1}{n_{g(j)}} W_{l+1}[:, g(j)]\, s^{(t)}_{l,g(j)} = \sum_{c=1}^{C_l} n_c \cdot \frac{1}{n_c} W_{l+1}[:, c]\, s^{(t)}_{l,c} = \sum_{c=1}^{C_l} W_{l+1}[:, c]\, s^{(t)}_{l,c} = z^{(t)}_{l+1}.$$

So layer $l+1$ receives exactly its pre-widening input at every t; since no other layer's weights changed, every layer downstream, and the final output, is unchanged. The same argument applies unchanged when l is the last conv and $l+1$ is the FC readout, with $W_{l+1}[:, c]$ replaced by channel c's block of the flattened FC weight.

Unlike Net2DeeperNet's identity-layer (Section 2.3), this proof never relies on the neuron equations being idempotent, so it holds for the LIF and ALIF neurons alike, and composes step-by-step across an arbitrary sequence of widened layers.

2.3 DEEPEN

Net2DeeperNet (Chen et al., 2016) inserts a new layer whose weight is the identity matrix, so it computes $h^{(i+1)} = \phi(Ih^{(i)}) = \phi(h^{(i)})$. This equals the original activation exactly when ϕ is idempotent ($\phi(\phi(x)) = \phi(x)$), which is true of ReLU following ReLU, since $h^{(i)}$ is already non-negative, but false in general, as other functions like the sigmoid or tanh are not idempotent.

We insert the identity the same way, by inserting conv with input weights initialized to identity, right after conv l, with its BatchNorm calibrated to the identity map: setting $\gamma = \sqrt{\sigma^2 + \varepsilon}$, $\beta = \mu$, running mean μ, running variance σ^2 (the layer's measured pre-BN statistics) gives $\mathrm{BN}(x) = \gamma \frac{x-\mu}{\sqrt{\sigma^2+\varepsilon}} + \beta = x$ exactly, ε canceling.

Thus, in theory, net2deepernet is exact for LIF neurons with binary inputs, as in purely binary, the arriving current is exactly 1 when the source fires and 0 otherwise. The network fixes $v_{th} = 1$ (Section 2.1), so for a LIF neuron each input spike yields an identical output spike, and $\mathcal{N}_{l+1}$ reproduces $s^{(t)}_l$ bit-for-bit at every t, meaning the insertion is exact.

However, using classical net2deepernet, we run into some problems in the realm of SNNs that don't exist in the ANN world. First, one can only guarantee exactness in ALIF networks if a non-adaptive LIF layer is inserted. The identity layer relies on the inserted neuron having a fixed v_{th}, which holds for a LIF neuron but not an ALIF one. If an ALIF layer is inserted, once the inserted neuron fires, its threshold rises to $v_{th} + \beta_{\text{adapt}} b^{(t)}$, and the next arriving spike still delivers a current of exactly 1, which is now below the threshold, so the receiving neuron no longer spikes. Thus, only non-adaptive LIF layers were inserted in this study. Though, this can be resolved by gradually increasing the adaptability of an inserted non-adaptive LIF layer over time if one really desires a fully ALIF network.

The second problem is the pooling problem. Let $\mathcal{N}_{l+1}$ denote the inserted neuron, viewed as the map from a current sequence to the spike sequence it fires (LIF or ALIF, Section 2.1). The insertion is exact if and only if $\mathcal{N}_{l+1}$, driven by input $a^{(t)}_l$ as current, reproduces output $a^{(t)}_l$ as its own identical output.

If the input to the inserted layer is purely binary, net2deepernet is exact, as explained earlier.

If layer l is not strictly binary (followed by average pooling, $a^{(t)}_l = \mathrm{AvgPool}(s^{(t)}_l) \in \{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}$, for example), the arriving current takes fractional values strictly between 0 and 1, but $\mathcal{N}_{l+1}$'s output

at a single time step is binary, $s \in \{0,1\}$. No choice of gain or BN calibration can make a binary output equal a fractional-current input, so the insertion cannot be exact at a pooling boundary.

Furthermore, even for the theoretically sound binary inputs, rounding error becomes significant in practice. The proof needs $1 \geq v_{th}$, so a spike survives only if floating-point rounding of that 1 happens to fall the right way; in the five-layer chained insertions of Section 3, about 54% of a layer's spikes fail to survive by the last of the five layers, as this rounding error compounds across the chain.

2.3.1 DEEPEN BY GAIN SEARCH

To combat this property of SNN net2net, we propose a method of deepening that scales the inserted identity kernel by a gain c chosen directly in the spiking network. Both problems above are due to arriving current relative to the threshold. Pooling drops the current below 1, and float32 rounding can move it under as well. Thus, we search for the best possible multiplicative factor for the current, the gain c.

We insert an identity layer scaled by c, with its BatchNorm calibrated to the identity as before, and choose c per inserted layer to maximize the network's prediction agreement with the pre-growth net over a held-out calibration batch B from the training set. Writing $\hat{k}(x) = \arg\max_j \hat{y}_j(x)$ for the predicted class on input x (the greedy action, for CartPole),

$$c^\star = \arg\max_c \frac{1}{|B|} \sum_{x \in B} \left[\hat{k}_{\text{grown}}(c,x) = \hat{k}_{\text{base}}(x)\right],$$

using the same bracket notation as Section 2.1, over a grid $c \in [1.1, 4.0]$. Layers are chained front to back, so layer k's search uses layer $k-1$'s searched gain.

To prevent the loss due to rounding error that hid some spikes, the grid excludes $c < 1.1$.

2.4 BENCHMARK SPECIFICS

Specific hyperparameters and network configurations across all four evaluated benchmarks are summarized in Table 1.

Table 1: Hyperparameter configurations and network architectures across the evaluated benchmarks.

Hyperparameter / Setting	**CIFAR-10 / CIFAR-100**	**CIFAR-10-DVS**	**CartPole**
Input Type	Static RGB image	Dynamic DVS frames	4D State Vector
Input Dimension	$3 \times 32 \times 32$	$2 \times 48 \times 48$	4 (encoded to 32)
Number of Classes / Outputs	10 / 100	10	2 Actions / 1 Value
Base Network Architecture	5-Conv VGG	5-Conv VGG	2-Layer MLP
Base Widths / Channels	64–128–192–256–256	64–128–192–256–256	64–32
Grown Widths	64–192–320–448–256	64–192–320–448–256	96–48 (Actor)
Time Steps (T)	8	10 (equal-event binning)	16
Threshold (v_{th})	1.0	1.0	1.0
Leak (α)	0.9	0.9	–
Surrogate Gradient (β)	Fast-sigmoid ($\beta = 10.0$)	Fast-sigmoid ($\beta = 10.0$)	–
Adaptation ($\beta_{\text{adapt}}, \tau_{\text{adapt}}$)	0.5, 20.0 (ALIF variant)	0.5, 20.0 (ALIF variant)	–
Normalization / Calibration	BatchNorm (tdBN)	Per-sample norm + BatchNorm	Homeostatic Rate/LR
Learning Algorithm	Backprop (Surrogate)	Backprop (Surrogate)	R-STDP with TD(λ)
Optimizer & Learning Rate	Adam (1×10^{-3})	Adam (1×10^{-3})	Local Trace (1×10^{-3})
Batch Size	128	128	1 (per step)
Training Budget	15 + 15 epochs	15 + 15 epochs	1200 + 1200 episodes
Data Augmentation	RandomCrop (pad 4), HorizontalFlip, RandAugment (2,9), RandomErasing	HorizontalFlip, Shift ($\pm$6 px), Frame Repeat, Event Dropout ($\leq$10%)	None

2.4.1 CIFAR-10

Each static 32×32 RGB image is normalized and injected as a constant rate-coded input current $a_0^{(t)} = x$ into the first convolutional layer across all $T = 8$ discrete time steps. Intermediate layers

integrate current into leaky integrate-and-fire (LIF) or adaptive LIF (ALIF) membrane potentials $u_l^{(t)} = \alpha v_l^{(t-1)} + \hat{z}_l^{(t)}$ and fire binary Heaviside spikes $s_l^{(t)} \in \{0,1\}$ when membrane potentials cross $v_{\text{th}} = 1.0$, followed by a hard reset to zero. The final class prediction is obtained by accumulating the non-spiking linear readout layer's output over all $T = 8$ time steps: $\hat{y} = \frac{1}{T}\sum_{t=1}^{T} W_{\text{fc}} a_{\text{final}}^{(t)}$.

2.4.2 CIFAR-100

CIFAR-100 follows an identical rate-coded spike encoding scheme as CIFAR-10. Normalized 32×32 images are held constant over $T = 8$ time steps to input to a 5-conv VGG backbone. Binary spikes propagate through surrogate-gradient LIF and ALIF units, and class logits across 100 fine-grained categories are computed by accumulating the outputs of a 100-dimensional linear head Linear(4096, 100) over the $T = 8$ steps.

2.4.3 CIFAR-10-DVS

CIFAR-10-DVS (Li et al., 2017) evaluates network growth on dynamic event streams captured by a DVS128 sensor. Raw asynchronous events are partitioned into $T = 10$ temporal frames using equal-event-count binning (dividing the total event count per recording equally across 10 temporal frames to avoid empty frames) and accumulated into $2 \times 48 \times 48$ spatial grids. Frame t is directly injected as the time-varying input current $x^{(t)}$ into Conv 0 at time step t, driving dynamic spike generation across $T = 10$ steps. Logits are computed by averaging the output of a 10-dimensional linear readout Linear(9216, 10) over the 10 time steps.

2.4.4 CARTPOLE

CartPole-v1 evaluates spiking network growth under local reinforcement learning without back-propagation. The continuous 4D state vector $s = (x, \dot{x}, \theta, \dot{\theta})$ is squashed via $z = \tanh(s/s_{\text{limits}})$ and encoded into continuous input firing rates via $n_{\text{pop}} = 8$ Gaussian tuning curves per state variable (32 channels total tiled over $[-1, 1]$), injected as constant input current over $T = 16$ time steps per environment step.

The actor network ($32 \to 64 \to 32 \to 2$, taking the 32 encoded channels as input) employs stochastic escape-noise LIF neurons, where the post-membrane potential $v_l^{(t)}$ determines an instantaneous firing probability $\rho_l^{(t)} = \sigma((v_l^{(t)} - v_{\text{th}})/\Delta)$ and spikes are sampled $s_l^{(t)} \sim \text{Bernoulli}(\rho_l^{(t)})$, providing exploration for action selection via spike-count argmax. The critic network ($32 \to 64 \to 32 \to 1$) uses deterministic LIF neurons ($s_l^{(t)} = [v_l^{(t)} \geq v_{\text{th}}]$) to predict the state value $V(s)$ as the average readout current over $T = 16$ steps. Synapses update via reward-modulated STDP (R-STDP) with local eligibility traces $e_{ij} = \frac{1}{T}\sum_{t=1}^{T}(s_i^{(t)} - \rho_i^{(t)}) \cdot \bar{P}_j^{(t)}$ modulated by a global scalar TD error $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$, balanced by per-neuron homeostatic firing rate and learning rate gain calibrations.

3 RESULTS

Every run trains a base network for the first half of the budget, applies exactly one growth operation at the dashed vertical line, then trains the grown network for an equal second half. The `base` curve in Figures 1–4 therefore stops at the growth point: it is the pre-growth network every operation starts from, not a train-longer control. All numbers are mean ± std over 5 seeds. We report *agreement* at the growth point (Table 2), providing an empirical measure of preservation, and describe how each operation's accuracy or reward tracks base over the post-growth budget in Figures 1–4.

Operations. Each operation is named for what it does. The base is the 5-conv stack of Table 1, 64–128–192–256–256, pooled after convs 0–2. `widen` (Section 2.2) duplicates channels in the three interior convs $1, 2, 3$ at once, adding $\min(C_{\text{out}}^{l+1}, C_{\text{in}}^{l})$ to each, giving 64–192–320–448–256 (1.33M → 2.99M conv parameters). `deepen` (Section 2.3) chain-inserts 5 identity conv (with batchnorm) layers after the pooling after conv 1 (+0.74M) directly into the SNN. `deepen gridsearch` inserts the same layers with each one's gain searched as in Section 2.3.1. `deepen gridsearch nopool` and `deepen nopool` insert layers the same way as their aforementioned counterparts, except their site of

insertion is moved to conv 3 (+2.95M), where no pooling intervenes and the input is binary. `base` is the pre-growth network. `random grow` makes the same architecture change but initialises the new units randomly, rescaled to the base weights' standard deviation, separating added capacity from preservation; and `reinit grow` discards the base and trains the grown architecture from scratch on the post-growth budget alone.

For CartPole, the actor is grown, with the critic fixed at 32–64–32 → 1. `widen` gives 32–64–32–2 → 32–96–48–2, and `deepen` inserts 2 identity layers after the first hidden layer, giving 32–64–64–64–32–2. It has no pooling anywhere, so there is no `nopool` variant.

3.1 AGREEMENT

Table 2: Agreement between the pre- and post-growth network immediately after growth (%), before any post-growth training, mean ± std over 5 seeds. Prediction agreement for CIFAR-10/100/CIFAR-10-DVS, greedy action agreement for CartPole (with stochasticity turned off when measuring); both are percentages, with 100% indicating agreement on all evaluated samples.

	CIFAR-10		CIFAR-100		CIFAR-10-DVS		CartPole
Growth method	LIF	ALIF	LIF	ALIF	LIF	ALIF	
`widen`	100.00 ± 0.00	100.00 ± 0.00	99.99 ± 0.01	99.99 ± 0.01	100.00 ± 0.00	99.98 ± 0.04	100.00 ± 0.00
random grow (widen)	77.26 ± 7.89	71.23 ± 5.17	57.00 ± 2.80	50.90 ± 4.27	77.86 ± 3.98	74.24 ± 10.09	82.85 ± 5.57
`reinit grow (widen)`	9.95 ± 0.84	9.87 ± 1.20	0.95 ± 0.18	0.97 ± 0.22	6.16 ± 1.39	9.62 ± 3.49	46.99 ± 6.89
`deepen gridsearch nopool`	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	–
`deepen gridsearch`	90.27 ± 0.10	85.99 ± 0.93	77.95 ± 0.50	73.39 ± 1.35	83.77 ± 1.17	72.63 ± 2.58	100.00 ± 0.00
`deepen nopool`	94.38 ± 0.41	89.14 ± 2.96	80.30 ± 2.61	76.89 ± 2.15	79.42 ± 6.46	75.92 ± 5.20	–
`deepen`	33.56 ± 6.56	21.49 ± 2.83	7.84 ± 1.33	5.21 ± 0.94	39.10 ± 6.22	30.42 ± 5.88	100.00 ± 0.00
random grow (deepen)	9.50 ± 1.62	9.41 ± 1.37	1.05 ± 0.07	1.62 ± 0.38	16.54 ± 6.67	10.88 ± 3.55	71.76 ± 18.74
`reinit grow (deepen)`	9.95 ± 0.84	9.87 ± 1.20	0.95 ± 0.18	0.97 ± 0.22	6.16 ± 1.39	9.62 ± 3.49	45.90 ± 19.11

Table 2 confirms widen's exactness empirically. `widen` transfers at 100.00% agreement on every benchmark and both neuron models; the 99.98–99.99% entries on CIFAR-100 and CIFAR-10-DVS can be attributed to float32 rounding at the zero-margin threshold $v_{th} = 1$ flipping a few spikes. random grow widening retains only 51–83% of the base network's predictions, and full reinitialisation performance collapses. Note that the `reinit grow (widen)` and `reinit grow (deepen)` rows are identical to two decimal places on every image benchmark. The reinitialised network's BatchNorm layers still carry their default running statistics at this pre-training measurement point, which collapses spiking to zero within the first two conv layers regardless of the grown architecture's width or depth. With every logit pinned at exactly 0, the argmax tie-breaks to class 0 for every input and both grow variants, resulting in the identical behavior. CartPole's `reinit grow` rows differ (46.99 vs. 45.90) because its actor network has no BatchNorm.

Direct insertion at a pooling boundary (`deepen`) retains only 5.2–39.1% agreement, confirming that the inserted binary neuron cannot reproduce its fractional avg-pooled input. Moving the identical operation to a non-pooling site (`deepen nopool`), where the input is binary, recovers 75.9–94.4%.

This loss can be attributed to compounded rounding error. BatchNorm's identity reconstruction is not bit-exact in float32,[1] so the arriving current is 1 ± 10^{-7} rather than exactly 1. About 54% of the original spikes fail to survive, cumulatively, by the output of the fifth chained layer, as the loss compounds at each stage.

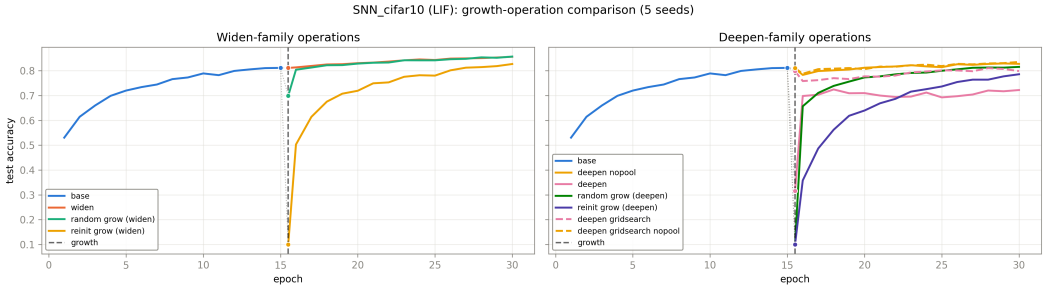
This rounding error, however, does not apply to the gain search, which selects $c^\star > 1$ by construction. At non-pooling sites, the search returns $c^\star = 1.1$, on every layer, seed, and dataset. As shown in Table 2, `deepen gridsearch nopool` scores an agreement of 100%, as any gain above 1 clears the threshold with margin, so the inserted layer reproduces its binary input bit-for-bit.

For insertions directly after pooling, only the first inserted layer takes a large gain (1.39–2.32), as it is the only layer that receives a fractional input. Every subsequent layer in the chain sees binary spikes again and falls back to the 1.1 floor. `deepen gridsearch` achieves agreements between 72.6% and 90.3% across datasets, substantially higher than `deepen`'s 5.2% to 39.1% range, though still not 100%, as the impossibility argument of Section 2.3 showed. For Cartpole, since the MLP actor–

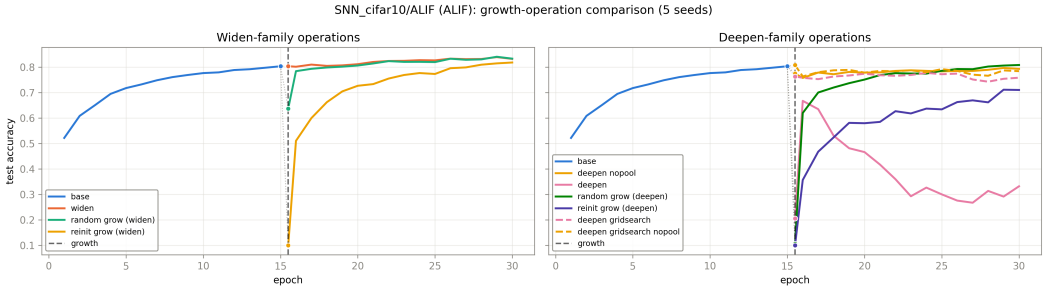
[1] Measured on the trained CIFAR-10 LIF base; with an input of exactly 1 received, the calibrated BN outputs it as 0.99999994 instead of 1 on 38 of 256 channels at the first of the five chained identity-BN layers.

critic has no pooling or Batchnorm, `widen`, `deepen`, and `deepen gridsearch` all reach 100.00% greedy-action agreement (stochasticity turned off during agreement measurement).

3.2 CIFAR-10



(a) LIF



(b) ALIF

Figure 1: **CIFAR-10 test accuracy.** Each color is an average over 5 seeds.

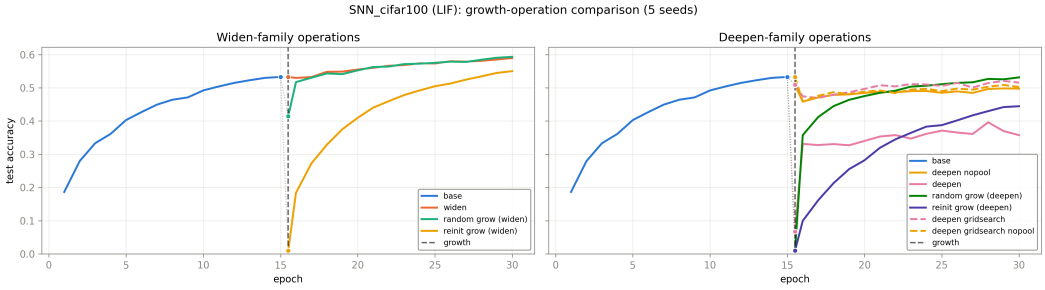
On CIFAR-10, `widen` never dips below its pre-growth accuracy and keeps climbing past it for the rest of the budget on both neuron models. `reinit grow` instead restarts near chance and spends the whole post-growth budget climbing back, never catching up. `random grow` recovers almost as cleanly as `widen` despite not being exact, so on this budget exactness buys a small head start rather than a better ceiling.

For deepening, direct insertion at the pooling boundary drops accuracy significantly at the growth point and decays further for the ALIF SNN. Searching the inserted gain recovers most of that gap. At non-pooling sites, both `deepen gridsearch nopool` and `deepen nopool` perform well, preserving significantly more accuracy and recovering faster than `random grow`, `reinit grow`, and `deepen` for both network types.

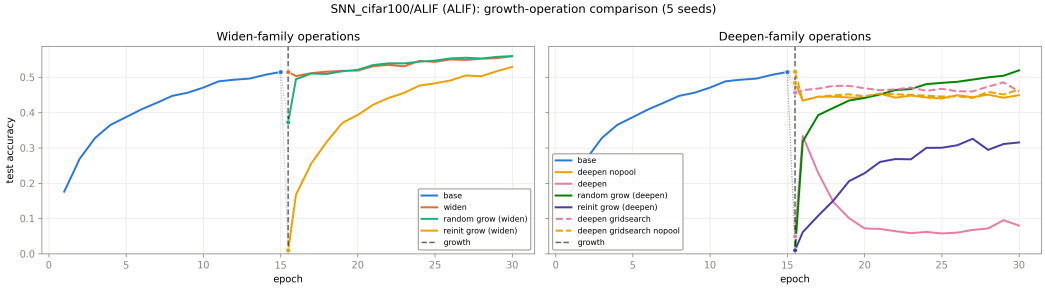
3.3 CIFAR-100

CIFAR-100 had much of the same results as CIFAR-10. For both network types, `widen` again recovers similarly to `random grow` and keeps climbing, while `reinit grow` is still climbing back toward base when the budget ends (Figure 2).

When deepening, the ALIF network once again does not recover at all, its accuracy declining significantly, well below where it started, inviting further investigation as to the effects of adaptive thresholds on growth methods. `deepen gridsearch nopool`, `deepen gridsearch`, and `deepen nopool` remain the most accuracy-preserving for both networks.



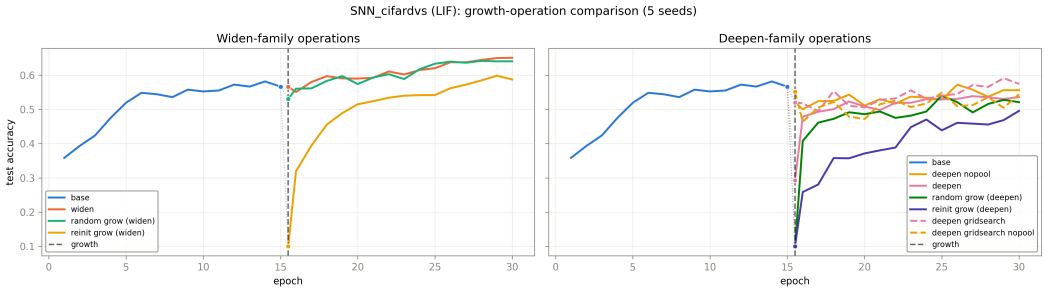
(a) LIF



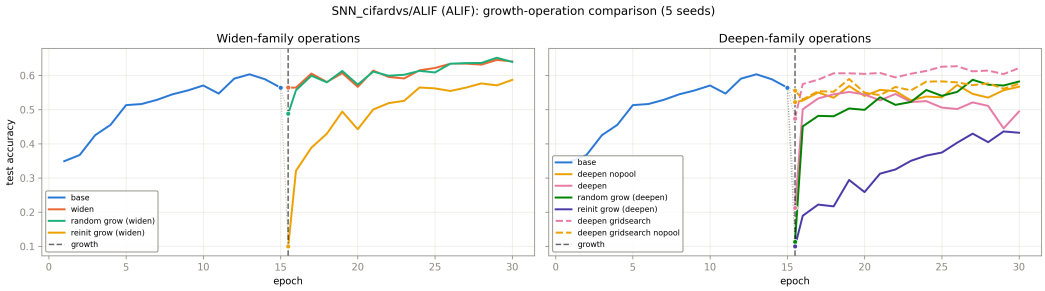
(b) ALIF

Figure 2: **CIFAR-100 test accuracy.** Each color is an average over 5 seeds.

3.4 CIFAR-10-DVS



(a) LIF



(b) ALIF

Figure 3: **CIFAR-10-DVS test accuracy.** Each color is an average over 5 seeds.

On event data, `widen` and `random grow` once again recover almost all of the accuracy and keep improving.

3.5 CARTPOLE

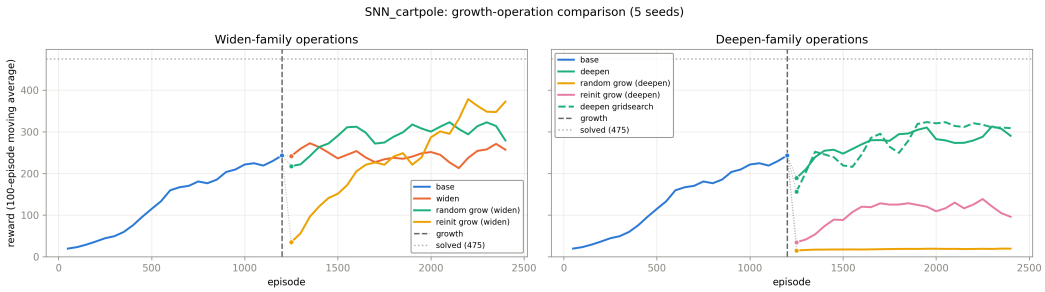


Figure 4: **CartPole.** Moving-average reward (100-episode window) for the R-STDP actor-critic SNN, with the dotted reference line at the CartPole-v1 solve threshold (475). Each color is an average over 5 seeds.

CartPole tests the operators under purely local, reward-modulated learning with no backpropagation. Because the actor–critic is an MLP with no pooling, every insertion site receives binary spikes, and `widen`, `deepen`, and `deepen gridsearch` are all fully agreed upon the evaluated samples (100.00 $\pm$ 0.00% greedy-action agreement).

For deepening, interestingly, default deepening begins with a slightly higher reward than deepen gridsearch. `random grow (deepen)` collapses at the growth point and never recovers, and `reinit grow (deepen)` climbs but stays far below the preserving operators. The collapse of the `random grow` operator can be attributed to the local nature of R-STDP. R-STDP has only a scalar TD error and local eligibility traces, so once a perturbation destroys the policy there is no gradient to undo it.

The widen family starts similarly to the image benchmarks, with `widen` performing similarly to `random grow`, `reinit grow` far behind, recovering much slower.

Note that our method of measurement, greedy-action agreement, only tests whether the membrane potential crosses the threshold, so a gain greater than 1 can leave this at 100 percent while still shifting the firing probability away from its pre-growth value, meaning agreement alone does not establish that the stochastic policy was preserved.

4 CONCLUSION

This paper extends net2net to spiking network architectures, for both LIF neurons and ALIF neurons. We show that widen is exact in SNN architectures, and deepen is only exact in theory under certain conditions. By searching the inserted layer's gain directly against prediction agreement, our `deepen gridsearch` method makes deepening exactly function preserving at non-pooling sites and mitigates the error derived from deepening right after a pooling layer, shown empirically to have much higher agreement to the original network after growth.

4.1 FUTURE WORK

First, future work can utilize these generalized operations in neural architecture search algorithms for SNNs, allowing stronger SNN architectures to be found. Second, the `deepen gridsearch` method is still not perfectly function preserving at a pooling boundary. Perhaps future work could improve on this method and further reduce the error from post-pooling growth. Third, more diverse types of SNNs, such as recurrent SNNs or spiking transformer architectures have yet to be explored by this paper. Future work may extend spiking morphism further to those architecture types.

REFERENCES

W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997, doi: 10.1016/S0893-6080(97)00011-7.

G. Indiveri and S.-C. Liu, "Memory and information processing in neuromorphic systems," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1379–1397, 2015, doi: 10.1109/JPROC.2015.2444094.

M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018, doi: 10.1109/MM.2018.112130359.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*, 2018.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2Net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations (ICLR)*, 2016.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI Conference on Artificial Intelligence*, 2018.

Utku Evci, Bart van Merriënboer, Thomas Unterthiner, Max Vladymyrov, and Fabian Pedregosa. GradMax: Growing neural networks using gradient information. In *International Conference on Learning Representations (ICLR)*, 2022.

Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. 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, 2019.

Bing Han, Feifei Zhao, Yi Zeng, Wenxuan Pan, and Guobin Shen. Enhancing efficient continual learning with dynamic structure development of spiking neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2023.

Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *International Conference on Machine Learning (ICML)*, volume 48 of *PMLR*, pp. 564–572, 2016.

Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

Qiang Liu, Lemeng Wu, and Dilin Wang. Splitting steepest descent for growing neural architectures. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. Firefly neural architecture descent: a general approach for growing neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Youngeun Kim, Yuhang Li, Hyoungseob Park, Yeshwanth Venkatesha, and Priyadarshini Panda. Neural architecture search for spiking neural networks. In *European Conference on Computer Vision (ECCV)*, 2022.

Byunggook Na, Jisoo Mok, Seongsik Park, Dongjin Lee, Hyeokjun Choe, and Sungroh Yoon. AutoSNN: Towards energy-efficient spiking neural networks. In *International Conference on Machine Learning (ICML)*, 2022.

Kaiwei Che, Luziwei Leng, Kaixuan Zhang, Jianguo Zhang, Qinghu Meng, Jie Cheng, Qinghai Guo, and Jianxing Liao. Differentiable hierarchical and surrogate gradient search for spiking neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Bing Han, Feifei Zhao, Yi Zeng, and Guobin Shen. Developmental plasticity-inspired adaptive pruning for deep spiking and artificial neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Yanqi Chen, Zhaofei Yu, Wei Fang, Tiejun Huang, and Yonghong Tian. Pruning of deep spiking neural networks through gradient rewiring. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

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 (NeurIPS)*, 2018.

Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. CIFAR10-DVS: An event-stream dataset for object classification. *Frontiers in Neuroscience*, 11:309, 2017.

Eugene M. Izhikevich. Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex*, 17(10):2443–2452, 2007.

A Related Work

A.1 Function-Preserving Growth in Artificial Neural Networks

Net2Net (Chen et al., 2016) introduced the two operators this paper extends, net2widernet and net2deepernet. Wei et al. (2016) generalised them into network morphism, extending net2net to cover changes in depth, width, and kernel size. Naturally, NAS followed. EAS (Cai et al., 2018) trains a meta-controller that emits Net2Net actions and NASH (Elsken et al., 2017) hill-climbs over morphisms, both avoiding the from-scratch evaluation of earlier search methods (Zoph & Le, 2017; Liu et al., 2019a; Pham et al., 2018). Furthermore, splitting steepest descent (Liu et al., 2019b) and Firefly (Wu et al., 2020) select and split neurons by a second-order functional criterion, and GradMax (Evci et al., 2022) initialises the new units' outgoing weights to maximise the gradient norm, addressing the zero gradients of new units whose outgoing weights are initialized to zero. Each of these methods, however, only applies to ANNs. Thus, this paper extends them to the realm of SNNs.

A.2 Architecture Search and Structural Plasticity in Spiking Networks

Neural architecture search has been explored with SNNs, but growth methods are still uncommon. SNASNet (Kim et al., 2022) scores untrained cells and shows that backward connections are necessary, AutoSNN (Na et al., 2022) searches a supernet under a joint accuracy and spike-count objective, and SpikeDHS (Che et al., 2022) makes both the cell and layer search differentiable, mirroring the ANN predecessor, DARTS (Liu et al., 2019a). However, each is forced to train the selected architecture from a random initialization, something that growth operators like net2net enable algorithms to avoid.

The closest work to ours changes SNN structure during training under biologically inspired heuristics. DSD-SNN (Han et al., 2023) grows neurons randomly for each incoming task and prunes redundant ones, DPAP (Han et al., 2024) applies a developmental pruning algorithm with regeneration, and gradient rewiring (Chen et al., 2021) prunes deep SNNs by jointly optimizing weights and connectivity. However, in all of them, newly added units start from random weights, so the network's function changes discontinuously at the moment of growth. We are not aware of prior work that gives an SNN growth operator with function-preservation.

A.3 Local and Reward-Modulated Learning

The CartPole networks are grown under a rule that uses no backpropagation at all. R-STDP comes from Izhikevich (2007), who solved the distal reward problem by pairing a slowly decaying eligibility trace with a global dopamine-like signal. Growth under such rules has not been examined. Our results suggest that successful deepening is only possible with function preserving methods. In R-STDP trained networks, there is no gradient to repair a policy that a non-preserving deepening step has destroyed (Section 3).

B AI Statement

In this work, we used generative AI tools (Claude) to assist with implementing code for our experiments and methods, to give feedback on those methods, and to help polish the writing of the manuscript. We have not used generative AI tools for generating synthetic datasets, developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical

ingredients for proofs, assisting in the writing of proofs, proposing or refining hypotheses, assisting with translation, cleaning or reformatting datasets, supporting qualitative or thematic data analysis, or interpreting results. We have reviewed all AI-assisted work; all AI-assisted code was tested and verified for correctness by the authors, and all AI-edited text was reviewed and revised to ensure it accurately reflects our own analysis and conclusions. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.