ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

Where to grow: a surprisingly straightforward criterion to detect under-expressive layers

Yifan Wang[1], Julien Mille[1,2] and Moncef Hidane[1]

1- INSA Centre Val de Loire, Blois, France
Laboratoire d'Informatique Fondamentale et Appliquée de Tours, Tours, France
2- Université Paris-Saclay
Laboratoire Interdisciplinaire des Sciences du Numérique
Gif-sur-Yvette, France

Abstract. The idea of gradually increasing the capacity of a neural network, during or after training, has gained attention in recent years. Online network growing raises three fundamental questions: *when*, *where*, and *how* to expand. Among all questions, the present paper focuses on the *where*. We introduce the *Normalized Activation Gradient Norm* (NAGN), a lightweight criterion to detect under-expressive layers using standard backpropagation signals. Experiments on image classification demonstrate that this approach consistently discovers compact architectures that match larger static baselines with reduced training costs.

1 Growing networks

Traditional deep learning models often require large computational resources and datasets, and they usually contain many redundant parameters. Online growing neural networks were introduced as a way to lower these computational demands and/or to find a more efficient architecture for a given task. These methods start from a small seed architecture and gradually expand it during training [1, 2, 3], either by adding units to existing layers or by introducing new layers. A key challenge is to decide when to trigger the growth, where to apply it, and how many units/layers to add [2].

Early online methods keep adding units to pre-specified hidden layers until the model reaches some performance target [4, 5]. Later work uses training signals to guide network growth. Some methods maximize gradient norms when initializing new neurons under fixed growth schedules [6], others search a small functional neighborhood to select additional neurons or layers [7], and some increase depth when validation accuracy stops improving [8]. These approaches mainly address *when* to expand and *how much* capacity to add, but rely on hand-designed or restricted choices for *where* growth can occur.

Only a few methods directly target this question. Liu et al. [1] use the splitting matrix, a truncated Hessian whose minimum eigenvalues identify which neurons, and therefore which layers, should be expanded. Maile et al. [2] propose the NORTH* framework, which monitors the numerical rank of post-activation outputs or weights to trigger growth. Verbockhaven et al. [3] introduce the TINY algorithm, which measures expressivity gaps between functional and realizable gradients and resolves them by optimally adding neurons through a quadratic optimization step.

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

In contrast to previous growing schemes, our approach introduces a lightweight and gradient-based mechanism for adaptive network expansion. We define the *Normalized Activation Gradient Norm* (NAGN), a simple layer-wise statistic derived directly from standard backpropagation signals to identify under-expressive layers, without extra expensive operations. Building on this measure, we design an output-preserving growth operator applicable to both dense and convolutional layers. Experiments demonstrate that this method consistently discovers compact architectures that match or exceed the performance of wider static backbones at less training cost.

2 Growing layers

Consider a feed-forward classification neural network that contains L layers. The width (number of features) of each layer ℓ is n_ℓ. The transformation in layer ℓ is characterized by a weight matrix $\mathbf{W}_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and a bias vector $\mathbf{b}_\ell \in \mathbb{R}^{n_\ell}$. For a given network input $\mathbf{x} = \mathbf{h}_0 \in \mathbb{R}^{n_0}$, pre-activation output $\mathbf{z}_\ell$ and post-activation output $\mathbf{h}_\ell$ are computed as $\mathbf{z}_\ell = \mathbf{W}_\ell \mathbf{h}_{\ell-1} + \mathbf{b}_\ell$ and $\mathbf{h}_\ell = \phi_\ell(\mathbf{z}_\ell)$. Typically, nonlinear function ϕ_ℓ is the ReLU function if $\ell < L$ and softmax if $\ell = L$. We denote the cross-entropy loss between the network output $\mathbf{h}_L$ and one-hot label $\mathbf{y}$ as $\mathcal{L}(\mathbf{h}_L, \mathbf{y})$.

Assume that the decision to grow (or leave unchanged) each layer ℓ with $\Delta n_\ell \geq 0$ units has been taken[1]. How to choose these layers, and the number of neurons to add, is left to the next section. The general growth scheme can be described by assuming that two successive dense layers ℓ and $\ell+1$ gain Δn_ℓ and $\Delta n_{\ell+1}$ units, respectively. In such case, weight matrix $\mathbf{W}_{\ell+1}$ gains rows and columns, leading to the following grown weight matrix

$$\hat{\mathbf{W}}_{\ell+1} = \begin{array}{c} \\ n_{\ell+1} \\ \Delta n_{\ell+1} \end{array} \overset{\begin{array}{cc} n_\ell & \Delta n_\ell \end{array}}{\begin{bmatrix} \mathbf{W}_{\ell+1} & \mathbf{W}_{\ell+1}^{\text{new1}} \\ \mathbf{W}_{\ell+1}^{\text{new2}} & \mathbf{W}_{\ell+1}^{\text{new3}} \end{bmatrix}}. \tag{1}$$

We propose that new weights be initialized so that the network output $\mathbf{h}_L$ is unchanged, but that the loss gradient w.r.t to these new weights be nonzero. One possible way to ensure this is that submatrices $\mathbf{W}_{\ell+1}^{\text{new1}}$ and $\mathbf{W}_{\ell+1}^{\text{new3}}$ be initialized with zeros, and $\mathbf{W}_{\ell+1}^{\text{new2}}$ with normal zero-centered random values, with variance $2/n_\ell$ (following Kaiming initialization). The grown bias vector $\hat{\mathbf{b}}_\ell$ is built by adding Δn_ℓ zeros to $\mathbf{b}_\ell$. The simpler case where only one layer among ℓ and $\ell+1$ is grown can be easily deduced from the previous equations, setting Δn_ℓ or $\Delta n_{\ell+1}$ to 0, accordingly.

The extension to convolutional layers is straightforward. For a convolutional layer ℓ, the pre-activation output $\mathbf{z}_\ell$ and post-activation output $\mathbf{h}_\ell$ are 3D tensors in $\mathbb{R}^{n_\ell \times H_\ell \times W_\ell}$, where n_ℓ, H_ℓ and W_ℓ are the number of channels, image height

[1]Except for the output layer, for which we always have $\Delta n_L = 0$.

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

and image width, respectively. Weight $\mathbf{W}_\ell$ is a 4D tensor in $\mathbb{R}^{n_\ell \times n_{\ell-1} \times k_\ell \times k_\ell}$, where k_ℓ is the kernel size. When growing layer ℓ, we add Δn_ℓ channels and the kernel size remains fixed. Assuming that two successive convolutional layers ℓ and $\ell+1$ are grown, the three additional subtensors $\mathbf{W}_{\ell+1}^{\text{new1}} \in \mathbb{R}^{n_{\ell+1} \times \Delta n_\ell \times k_\ell \times k_\ell}$, $\mathbf{W}_{\ell+1}^{\text{new2}} \in \mathbb{R}^{\Delta n_{\ell+1} \times n_\ell \times k_\ell \times k_\ell}$ and $\mathbf{W}_{\ell+1}^{\text{new3}} \in \mathbb{R}^{\Delta n_{\ell+1} \times \Delta n_\ell \times k_\ell \times k_\ell}$ are set analogously to the dense layer case, except that the variance for initializing $\mathbf{W}_{\ell+1}^{\text{new2}}$ is $2/(n_\ell k_\ell^2)$. It is common that a convolutional layer ℓ is followed by a batchnorm layer, which is parameterized by scale and shift vectors, both of size n_ℓ. If this is the case, the scale vector is expanded with Δn_ℓ ones, and the shift vector with Δn_ℓ zeros.

3 Detecting under-expressive layers

In what follows, we formulate the network as a sequence of L layers partitioned at index K: the backbone consists of convolutional layers for $1 \leq l < K$, followed by dense layers for $K \leq l \leq L$. Consequently, the pre-activation z_l is a 3D tensor when $l < K$ and a vector when $l \geq K$, with an implicit flattening operation at the interface. Note that auxiliary layers (e.g., batch normalization, pooling) are treated as implicit operations between indices.

From the functional point of view of Verbockhaven [3], the most effective direction to update the pre-activation output of layer ℓ is $-\partial\mathcal{L}/\partial\mathbf{z}_\ell$. However, in practice, changes in $(\mathbf{W}_\ell, \mathbf{b}_\ell)$ cannot achieve this optimal update for all samples (of the whole training or, at least, from the current minibatch). This indicates a limitation in the layer's expressive capacity. We propose a novel criterion to detect under-expressive layers, based on the observation that these layers tend to exhibit much larger magnitudes of activation gradient $\partial\mathcal{L}/\partial\mathbf{z}_\ell$ than well-sized layers. Intuitively, when the representation produced by layer ℓ is not expressive enough, changes in its parameters may fail to generate the activation variations needed to decrease the training loss, yielding activation gradients that are consistently larger than those of neighboring layers, as depicted in Fig. 1a.

A special case of an under-expressive layer is an *explicit bottleneck*[2], which is a layer ℓ such that $n_{\ell-1} > n_\ell < n_{\ell+1}$. Such layers are common in autoencoders but are usually avoided in image classification backbones[3]. Our criterion should be able to detect at least these explicit bottlenecks as under-expressive layers. This necessary sanity check is described in Section 4. We define the *Normalized Activation Gradient Norm* (NAGN) as

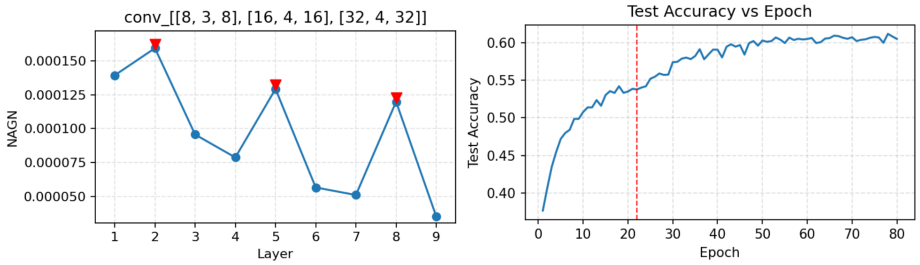
$$\text{NAGN}_\ell = \sqrt{\mathbb{E}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}}\left[\frac{1}{T_\ell}\left\|\frac{\partial\mathcal{L}}{\partial\mathbf{z}_\ell}\right\|_2^2\right]},$$

where $\mathcal{D}$ is (a subset of) the training set, and normalization factor T_ℓ is the total number of activation elements in $\mathbf{z}_\ell$ for one sample, that is $T_\ell = n_\ell$ for a dense

[2]Note that the term "bottleneck" has several meanings in machine learning, for example in ResNet bottleneck blocks or in the information bottleneck principle. This is a different concept here.

[3]The simpler condition $n_\ell < n_{\ell-1}$ is not even supposed to appear in image classification backbones.

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.



(a) (b)

Fig. 1: Illustration of a one-step growth example on a VGG-like network with initial sizes $\big[[8,3,8],[16,4,16],[32,4,32]\big]$ (Three blocks of three convolutional layers each). (a) NAGN profile captured at the growth trigger. (b) Test accuracy evolution, the vertical red dashed line marks the epoch where growth is triggered.

layer, and $T_\ell = n_\ell H_\ell W_\ell$ for a conv layer. This normalization ensures that the NAGN can be compared over different layers. Our claim is that a large NAGN_ℓ is a good indicator that layer ℓ is under-expressive. Advantageously, it involves only quantities already computed during backpropagation, and thus introduces negligible additional computational cost. We actually propose to detect strong local maxima of NAGN_ℓ as under-expressive layers. To do so, for interior layers $(2 \le \ell \le L-1)$, we define a local neighborhood average and a relative contrast:

$$\widetilde{\mathrm{NAGN}}_\ell = \frac{\mathrm{NAGN}_{\ell-1} + \mathrm{NAGN}_{\ell+1}}{2}, \quad r_\ell = \frac{\mathrm{NAGN}_\ell}{\widetilde{\mathrm{NAGN}}_\ell} - 1. \tag{2}$$

At the two end layers ($\ell = 1$ and $\ell = L$), the neighborhood is incomplete. To estimate their local neighborhood average, we first get the median slope, $m = \mathrm{median}\big(\Delta_1, \ldots, \Delta_{L-1}\big)$, with $\Delta_\ell = \mathrm{NAGN}_{\ell+1} - \mathrm{NAGN}_\ell$, and then extrapolate

$$\widetilde{\mathrm{NAGN}}_1 = \mathrm{NAGN}_2 - m, \qquad \widetilde{\mathrm{NAGN}}_L = \mathrm{NAGN}_{L-1} + m.$$

The corresponding relative contrasts can then be computed as in Eq. (2). We choose the width increment Δn_ℓ, involved in Eq. (1), to be a piecewise linear function of the relative constrast, such that layer width can be, at most, doubled. We threshold and normalize the relative contrast, to obtain the under-expressiveness $s_\ell \in [0,1]$, and compute Δn_ℓ,

$$p_\ell = \max\{r_\ell - \tau, 0\}, \quad s_\ell = \frac{p_\ell}{\max_j p_j} \in [0,1], \quad \Delta n_\ell = \min\{\Delta_{\max}, \lfloor s_\ell n_\ell \rfloor\},$$

where τ is the contrast threshold (if unmet, layers with $g_l > g_{l-1}$ grow conservatively) and $\Delta_{\max}$ is the maximum number of neurons that can be added to a layer in a single growth.

4 Experiments

We evaluate the proposed growth mechanism on CIFAR-10 with standard data augmentation, randomly holding out 20% of the training set for validation. Net-

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

works are compact VGG-style ConvNets composed of three backbone blocks and a dense classifier. Each block consists of Conv+BN+ReLU sequences ending with 2×2 maxpooling. Convolutional layers use 3×3 kernels with unit padding and stride to preserve spatial dimensions. Widths are denoted as a list of sublists, as in Fig. 1.

Training employs a sliding window of size w_{acc} to monitor validation accuracy. A growth phase is activated when the relative gain between two consecutive windows falls below a specific threshold δ_{acc}. Formally, letting $\bar{\mathcal{A}}^{\text{val}}_{\text{prev}}$ and $\bar{\mathcal{A}}^{\text{val}}_{\text{curr}}$ denote the mean validation accuracies of the previous and current windows, a trigger occurs when $(\bar{\mathcal{A}}^{\text{val}}_{\text{curr}} - \bar{\mathcal{A}}^{\text{val}}_{\text{prev}})/\bar{\mathcal{A}}^{\text{val}}_{\text{prev}} < \delta_{\text{acc}}$. To prevent inefficient expansion, we also introduce a termination threshold δ_{stop}. If the improvement in mean validation accuracy between two consecutive growth triggers is less than δ_{stop}, we permanently disable further growth for the remainder of the run.

Once growth is triggered, we compute the width increments Δn_ℓ according to Sec. 3. We enforce a non-decreasing width constraint so that no layer becomes narrower than its predecessors, adjusting widths forward if necessary while respecting the per-step growth limit. The expanded weights are initialized as described in Sec. 2, and the optimizer state (including momentum buffers and step counts) is preserved to ensure a smooth transition.

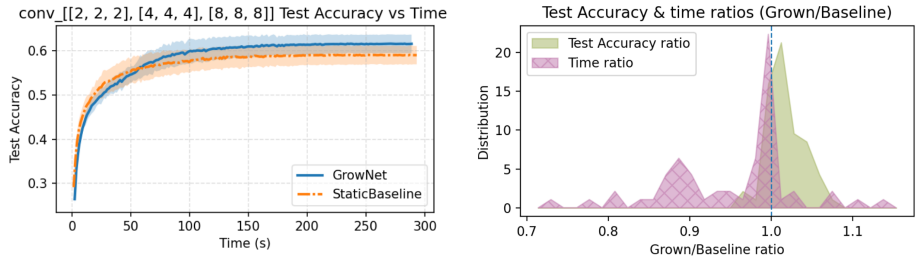
For all experiments, we use the AdamW optimizer with a cosine-annealing learning-rate schedule. We set the batch size to 500, the initial learning rate to 1×10^{-3}, and weight decay to 1×10^{-2}. Training runs for a total of 200 epochs. Regarding the growth hyperparameters, we set $\delta_{\text{acc}} = 3 \times 10^{-3}$ and $w_{\text{acc}} = 3$ epochs, with a contrast threshold $\tau = 5 \times 10^{-2}$. For the termination criterion, we set $\delta_{\text{stop}} = 5 \times 10^{-2}$; that is, if the accuracy gain between growth steps fails to exceed this margin, we cease network expansion.

After training a growing model, we build a corresponding fixed architecture with these final widths for comparison. This static model is then trained from scratch under identical conditions. We refer to this model as the *static baseline*.

4.1 Results and analysis

All reported results represent the average of five independent runs with distinct random seeds. As shown in Fig. 2, growing networks achieve accuracy comparable to or exceeding that of larger static baselines within the same epoch budget. Crucially, the total training time is reduced, yielding a superior accuracy–efficiency trade-off. The growth patterns remain consistent across seeds, demonstrating the robustness of the detection mechanism. The aggregate statistics in Fig. 2(b) cover twelve convolutional configurations of varying depth and width:[[2, 2, 2], [4, 4, 4], [8, 8, 8]], [[4, 4, 4], [4, 4, 4], [4, 4, 4]], [[4, 4], [4, 4], [4, 4]], [[4, 4], [8, 8], [8, 8]], [[8, 8], [8, 8], [8, 8]], [[8], [16], [16]], [[4], [4], [16]], [[4], [4], [4]], [[4], [4], [8]], [[4], [8], [16]], [[8], [8], [16]], and [[8], [8], [8]]. On average, grown networks match the baseline accuracy, with a smaller training time. The grown/baseline accuracy ratio (mean $\pm$ std. dev.) is 1.016 ± 0.020, while the training time ratio is 0.953 ± 0.075.

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.



(a) (b)

Fig. 2: Comparison of growing networks to static baselines. (a) Illustrative growth for a single convolutional architecture showing test accuracy (mean ± std across five runs). The growing model smoothly approaches and surpasses the static baseline with slightly reduced training time, its final widths typically converging to [[4, 4, 4], [4, 4, 4], [8, 8, 8]]. (b) Accuracy and training time ratios (grown vs. baseline) across all twelve architectures.

5 Conclusion

In this work, we proposed a lightweight strategy for determining *where* to grow neural networks. By monitoring the NAGN, our method identifies under-expressive layers dynamically without requiring expensive computations. Empirical results on various convolutional architectures demonstrate that this targeted expansion allows models to match or exceed the accuracy of larger static baselines while reducing training costs. Future work could investigate applying this criterion to other architectures, such as Residual Networks or attention mechanisms, or evaluating its effectiveness on more complex datasets.

References

[1] Q. Liu, L. Wu, and D. Wang. Splitting steepest descent for growing neural architectures. In *NeurIPS*, 2019.

[2] K. Maile, E. Rachelson, H. Luga, and D.G. Wilson. When, where, and how to add new neurons to ANNs. In *AutoML*, 2022.

[3] M. Verbockhaven, T. Rudkiewicz, S. Chevallier, and G. Charpiat. Growing tiny networks: spotting expressivity bottlenecks and fixing them optimally. *TMLR*, 2024.

[4] T. Ash. Dynamic node creation in backpropagation networks. *Connection Science*, 1(4):365–375, 1989.

[5] M. Wynne-Jones. Node splitting: A constructive algorithm for feed-forward neural networks. In *NIPS*, 1991.

[6] U. Evci, B. van Merrienboer, T. Unterthiner, F. Pedregosa, and M. Vladymyrov. GradMax: Growing neural networks using gradient information. In *ICLR*, 2022.

[7] L. Wu, B. Liu, P. Stone, and Q. Liu. Firefly neural architecture descent: a general approach for growing neural networks. In *NeurIPS*, 2020.

[8] W. Wen, F. Yan, Y. Chen, and H. Li. AutoGrow: Automatic layer growing in deep convolutional networks. In *KDD*, 2020.