BOHB over Growing ConvNets

Joint Architecture Search and Hyperparameter Optimization of a Convolutional Neural Network

Lukas Berg[1]

[1]Leibniz University Hannover

Abstract Hyperparameter-efficient configuration spaces are the dream of every algorithm that does Joint Architecture and Hyperparameter Search (JAHS). In this paper, I analyse and adapt a neurogenesis strategy that dynamically grows layers during training. Then, I apply an of-the-shelf multi-fidelity approach based on BOHB to search for the optimal configuration, with neurogenesis enabling a hyperparameter search space of $O(2)$ instead of $O(d)$ for network-depth d.

1 Introduction

The goal of the project motivating this paper is to do Joint Architecture and Hyperparameter Search (JAHS) on FashionMNIST (Xiao et al., 2017), an image-classification dataset, beating the accuracy of a baseline network.

In principle, efficient multi-fidelity hyperparameter optimization strategies like Bayesian Optimization and Hyperband (BOHB) (Falkner et al., 2018) can be used for Neural Architecture Search (NAS) to find any architecture that has been (hyper-)parameterized. However, the configuration space becomes very large very quickly (scaling exponentially). A naive approach to limit the search space is to only consider simple feed forward networks with convolutional layers in the front and fully-connected layers in the back. Even then, the network requires a hyperparameter specifying channel- or neuron-count for each layer, as successful networks of this type, e.g. VGG (Simonyan and Zisserman, 2014), vary the number of channels or neurons per layer. Neurogenesis presents an opportunity to overcome this linear increase in hyperparameters with depth.

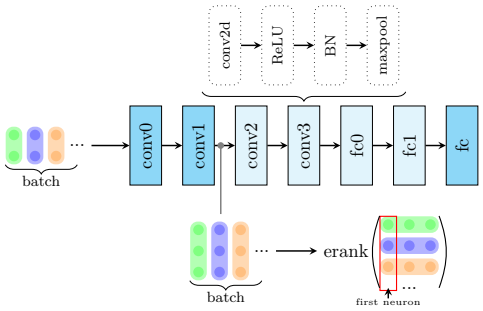
In their recent paper *When, where, and how to add new neurons to ANNs* (2022), Maile et al. introduced the NORTH* neurogenesis strategies for growing the number of neurons/channels in fully-connected and convolutional layers dynamically during training. In terms of image classification, they applied the strategies to VGG-11 (Simonyan and Zisserman, 2014) and WideResNet-28 (Zagoruyko and Komodakis, 2016) backbones on the CIFAR10 dataset (Krizhevsky, 2009). I adopt and adapt their NORTH-Select strategy.

2 Growing ConvNets

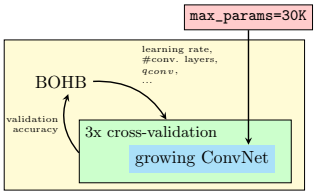
The NORTH* strategies work on a predefined backbone network. In fully-connected layers, they grow the number of neurons, in convolutional layers the number of channels. The number of layers or other computational elements remain untouched. This paper focuses on the NORTH-Select strategy, which was well motivated and showed a high performance throughout all experiments in the original paper. From here on, I will use the term *neurons* for channels as well for readability reasons.

2.1 NORTH-Select

The NORTH-Select strategy is based around the activations of a layer (i.e., the layer's output). Intuitively, NORTH-Select wants to grow layers if they already use most of their representative



(a) Network architecture framework of convolutional and fully-connected layers. Less saturated layers are optional. Illustration includes computation of erank for conv1.



(b) BOHB optimizes training and architectural hyperparameters jointly.

Figure 1: Architecture framework and setup with BOHB.

power across different inputs. To measure how much of the representational space is used, it captures the activations[1] $\boldsymbol{H}$ across a mini-batch of size b and computes the ϵ-numerical rank:

$$\text{rank}_\epsilon(\boldsymbol{H}) = \text{COUNT}(\sigma > \epsilon), \quad \sigma \in \text{SVD}(\frac{1}{\sqrt{b}}\boldsymbol{H}), \quad \epsilon = 0.1 \tag{1}$$

where COUNT returns the number of singular values of $\frac{1}{\sqrt{b}}\boldsymbol{H}$ larger than ϵ. In other words, rank_ϵ is a metric for the orthogonality of neuron activations.

After the i-th weight update during training of the network, NORTH-Select adds

$$n_{add} = \max(0, \lfloor \text{rank}_{\epsilon,i} - \gamma \frac{n_i}{n_0} \text{rank}_{\epsilon,0} \rfloor) \tag{2}$$

neurons to each layer individually, with scaling factor γ (usually $\gamma = 0.9$) and layer size n_i in step i.[2] Therefore, the number of neurons grows whenever the relative rank of the current activations surpasses the one from the very first mini-batch.

The new neurons are incorporated into the existing network. Their outgoing edges are initialized to 0, therefore they do not affect the network's output immediately. For the incoming edges, many candidate weights are sampled randomly. Those which maximize the activation's orthogonality are chosen as weights for the new neurons. See appendix A for contributions regarding the efficiency of this selection process.

2.2 Adjustments to NORTH-Select

Maile et al. experimented with NORTH-Select on the CIFAR-10 dataset, which is very similar to the FashionMNIST dataset that this paper targets. Even though the networks grown using NORTH-Select often outperformed the manually-crafted baseline, I found the following issues with their approach, some of which were also pointed out by reviewers on OpenReview[3].

1. Due to computational constraints, layers cannot grow indefinitely. The authors set a maximum size k-times that of the backbone for each layer which was reached often (see Figure 5 in the original paper). Therefore, the generality without a strong backbone remained unclear.

2. A layer's initial rank_ϵ functions as a relative target for the rest of training. The initial activations, however, vary depending on the random weight initialization and very first mini-batch.

3. For very small layers (e.g., $n = 8$), the discrete nature of rank_ϵ presents a challenge. If a layer exhibits full rank initially, it can never grow: $n_{add} = \max(0, \lfloor \text{rank}_{\epsilon,i} - \gamma \text{rank}_{\epsilon,0} \rfloor) = \lfloor 8 - 0.9 * 8 \rfloor = 0$

[1] For convolutional layers, the activations are flattened to (height×width×batch) × neurons

[2] The formula in the original paper uses min instead of max, which is probably a typo.

[3] `https://openreview.net/forum?id=SWOg-arIg9`

To combat these issues, this paper presents the following alterations to NORTH-Select: I propose to use the effective rank (erank) metric introduced by Roy and Vetterli (2007) instead of the ϵ-numerical rank.

$$\text{erank}(\boldsymbol{H}) = \exp(-\sum_{k=1}^{n} p_k \log p_k), \quad p_k = \frac{\sigma_k}{||\sigma||_1}, \quad \sigma \in \text{SVD}(\frac{1}{\sqrt{b}}\boldsymbol{H}) \tag{3}$$

Essentially, the erank-metric is defined as the Shannon entropy of normalized singular values to the power of e. It is continuous (fixing issue 3) and removes the need for the additional hyperparameter ϵ. Figure 4 shows the behaviour of both metrics over the course of training the baseline network (which does not grow).

Instead of using the initial erank value as a rank target for growth, I propose to use a constant threshold q for the relative erank r, introducing hyperparameter q while eliminating γ.

$$n_{add} = \max(0, \lceil (r-q)n \rceil), \quad r = \frac{\text{erank}(\boldsymbol{H})}{n}, \quad q \text{ const.} \tag{4}$$

where n is the current number of neurons. This solves issue 2.

As the added neurons try to maximize the activations' orthogonality, I reduce the frequency of growth from every step to every 10 steps and smooth the r-metric using an exponential moving average. Previously, the new neurons were not updated before the next activations were computed which sometimes resulted in an overshoot of added neurons.

To avert issue 1, the initial architecture is a simple 'rectangular' network with the same neuron count (of $n = 8$) in all layers. Growth is limited either by choosing a sufficiently high threshold q, so the maximum (implementation-bound) neuron count in a layer is never reached, or by introducing a total parameter limit which halts growth in all layers simultaneously.

type	channels	parameters/10^3	accuracy
baseline	16-32-64	29.1	91.2 ± 0.5%
N.-Select[4]	58-43-42	43.7 ± 19.1	91.3 ± 0.3%
q=1.0	8-8-8	2.0	88.7 ± 0.2%
q=0.8	10-16-21	6.4 ± 0.4	90.0 ± 0.3%
q=0.6	19-51-54	38.3 ± 3.9	91.0 ± 0.4%
q=0.4	38-126-131	205.4 ± 49.9	91.4 ± 0.3%

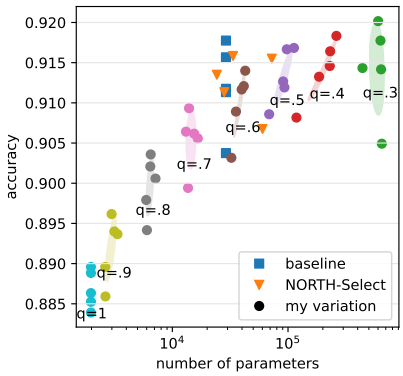


Figure 2: Impact of q on the growing process. All other hyperparameters stay the same.

3 Method

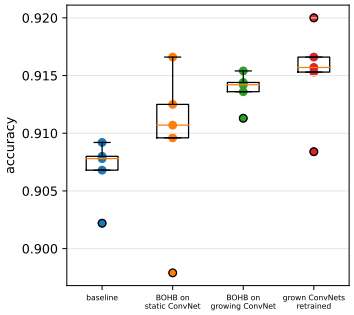
Applying this neurogenesis strategy to a simple architecture framework (see fig. 1a) results in the configuration space shown in fig. 3a. BOHB searches this space (see fig. 1b) for 6h using (only) two budgets: a minimum budget of 4 and a final budget of 10 epochs. I constrain growth by limiting the total number of weights to $\leq 30K$. This is about the size of the baseline network which uses 3 convolutional layers with channel counts 16-32-64.

For comparison, I also run BOHB on a static ConvNet which uses one size-specifying hyperparameter per layer, resulting in 9 hyperparameters instead of 5. To enforce the size limit, configurations which result in models that are too large are not trained and penalized by $\frac{num\_params}{max\_params}$.

[4]The original NORTH-Select strategy does not work on this experiment: see issue 3. When starting from a larger 16-16-16 channel network, the layers grow to their maximum size of 256 channels immediately. To provide a non-degenerated point for comparison, I additionally replace the ϵ-numeric rank with the erank.

hyperparameter	range	incumbent
learning rate	$[10^{-4}, 5 \cdot 10^{-2}]$	0.0125
#conv. layers	$\{2,\ 3,\ 4\}$	4
#f.-c. layers	$\{0,\ 1,\ 2\}$	0
q for conv. layers	$[0,\ 1]$	0.104
q for f.-c. layers	$[0,\ 1]$	-

(a) Configuration space.



(b) Final performance.

Figure 3: Search space and results after search.

4 Experiments

The code for the experiments, written in Python using PyTorch, is available on GitHub[5]. I use SMAC3's (Lindauer et al., 2021) SMAC4MF facade (which uses random forests as predictive models) to run BOHB, and reimplement NORTH-Select. All experiments were run on a single machine with Intel Core i7-9750H CPU and NVIDIA GeForce GTX 1660 Ti (Mobile) GPU.

4.1 Threshold hyperparameter q

To gauge the impact of threshold q, I run a sweep using the same setup and architecture as the baseline (except channel count). The experiment uses 5-fold cross-validation and exclusively the training section of FashionMNIST's default split. Figure 2 shows the results. Lower numbers of q result in larger, better performing networks. Appendix B contains figures visualizing the erank and channel count over time.

4.2 Searching the configuration space

In 6h, BOHB evaluated 60 different configurations of growing ConvNets. The resulting incumbent performs significantly better than the baseline (see fig. 3b). Assuming normally distributed results, I am able to reject the null-hypothesis of baseline and incumbent having the same mean accuracy using the Z-test ($p < 5\%$).

The growing networks also have higher average performance than the incumbent resulting from the naive parameterization, where BOHB evaluated 171 configurations (many of which were aborted immediately because of the parameter limit).

Training static ConvNets with the final size of the grown networks results in slightly higher performance on average. This was not the case in the original paper by Maile et al..

Surprisingly, the incumbent does not make use of the slow growth that the q-threshold enables. Instead, it uses a low q-threshold to reach the size-limit in very few steps. Therefore, growth does not help the network over the whole training process whenever neurons are needed, and is applied like a scaling concept independent from the actual learning process. Further research is needed to get a decisive picture on the reasons for this behaviour.

5 Conclusion

In this paper, I adapt NORTH-Select to grow convolutional networks of different sizes. In the end, networks grown using the incumbent configuration found by BOHB beat the baseline network significantly but evade continuous growth.

[5] https://github.com/automl-classroom/final-project-sprinting-panda

References

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1436–1445.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., and Hutter, F. (2021). Smac3: A versatile bayesian optimization package for hyperparameter optimization. In *ArXiv: 2109.09831*.

Maile, K., Rachelson, E., Luga, H., and Wilson, D. G. (2022). When, where, and how to add new neurons to ANNs. In *First Conference on Automated Machine Learning (Main Track)*.

Roy, O. and Vetterli, M. (2007). The effective rank: A measure of effective dimensionality. In *2007 15th European Signal Processing Conference*, pages 606–610.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *BMVC*.

A Selecting weights by most orthogonal activations

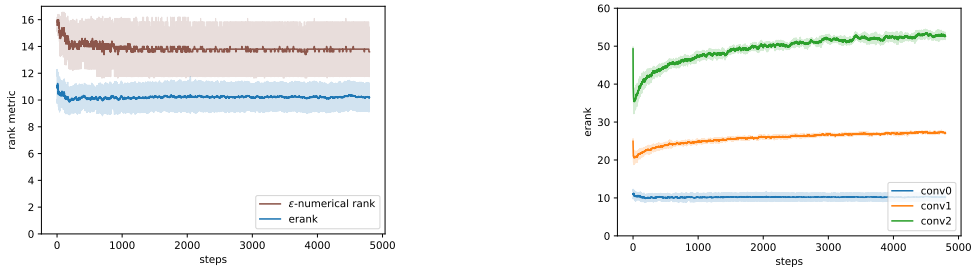
In their implementation[6], the authors measure the resulting rank$_\epsilon$-metric for each candidate's activation individually, which results in one SVD of $H_{\text{new},i}$ per candidate i (where $H_{\text{new},i}$ are all old activations and the one of the current candidate).

Leveraging the pseudo-inverse of the original SVD $\boldsymbol{H} = USV^T$ avoids additional SVDs altogether. Using the pseudo-inverse $\boldsymbol{H}^{-1} = VS^{-1}U^T$, I project all new activations $\boldsymbol{H}_{new}$ into the space spanned by the original activations, thus representing each new activation vector (one vector per neuron) using the existing neurons: $\boldsymbol{H}^{-1}\boldsymbol{H}_{new}$. Then, I reconstruct $\boldsymbol{H}_{new}$ by transforming by $\boldsymbol{H}$.

$$H_{\text{new, restored}} = \boldsymbol{H}\boldsymbol{H}^{-1}H_{\text{new}} \tag{5}$$

Those neurons whose activations have the highest reconstruction error (L2-norm) are most orthogonal to the existing neurons and added to the layer.

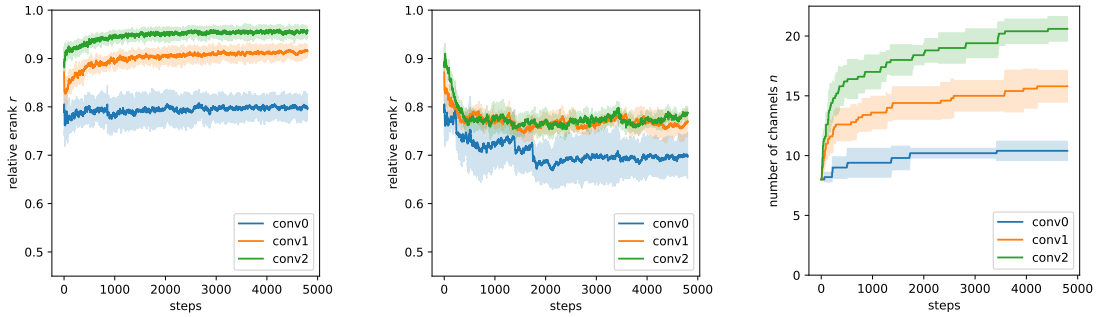
B erank metric



(a) Rank metrics in first layer.

(b) erank in all 3 convolutional layers.

Figure 4: Rank metrics observed in the baseline network during 10 epochs of training.



(a) r without growth ($q = 1.0$)

(b) r with $q = 0.8$

(c) growing channels ($q = 0.8$)

Figure 5: Full 10-epoch training run, starting from the 8-8-8 channels.

C Results after running BOHB

type	channels	parameters/10^3	accuracy
baseline	16-32-64	29.1	90.7 ± 0.2%
BOHB on static ConvNet	14-34-45	26.2	90.9 ± 0.6%
BOHB on growing ConvNet	30-34-36	29.9 ± 0.1	91.4 ± 0.1%
retraining grown ConvNets	30-34-36	29.9 ± 0.1	91.5 ± 0.4%

Figure 6: For the static ConvNet, BOHB chooses the layer sizes directly (from [8, 128]).

[6]The reference implementation, written in Julia, is available on GitHub: `https://github.com/neurogenesisauthors/Neurogenesis`