OPT2024: 16th Annual Workshop on Optimization for Machine Learning

Learning Morphisms with Gauss-Newton Approximation for Growing Networks

Neal Lawton NLAWTON@USC.EDU
Aram Galstyan GALSTYAN@ISI.EDU
Greg Ver Steeg GREG.VERSTEEG@UCR.EDU
Information Sciences Institute

Abstract

An appealing method for Neural Architecture Search (NAS) is based on growing networks via small local changes to the network's architecture called *network morphisms*. These methods start with a small seed network and progressively grow the network by adding new neurons in an automated way. However, efficiently determining the best way to grow the network remains a challenge. Here we propose a NAS method for growing a network which uses a Gauss-Newton approximation of the loss function to efficiently learn and evaluate candidate network morphisms. We then optimize this approximate loss function to efficiently learn morphism parameters. We compare our method with similar NAS methods for CIFAR-10 and CIFAR-100 classification tasks, and conclude our method learns similar quality or better architectures at a smaller computational cost.

1. Introduction

Neural Architecture Search (NAS), which seeks to automate the architectural design of neural networks, has become a central problem in machine learning research [7]. Researchers often advance state-of-the-art by carefully designing novel network architectures for specific problems, e.g., ResNets [12] for image classification and transformers [37] for natural language processing.

There are many different optimization methods for performing NAS, such as evolutionary methods [5, 29, 32], reinforcement learning methods [9, 41, 42], and pruning methods[8, 11, 25]. Another unique category of methods for NAS is growing methods [3, 10, 18, 22, 24, 36, 39]. Growing methods begin with a small seed architecture, then progressively grow a larger, more complex architecture by repeatedly applying small parameterizable local changes to the network's architecture called *network morphisms*. To grow a network, we must choose which morphisms to apply as well as the parameters for those morphisms. However, this optimization problem is challenging to solve at scale, when there are many possible morphisms to consider.

In this paper, we propose a method for learning and evaluating morphism parameters quickly and efficiently. Our method utilizes a Gauss-Newton approximation of the loss function to estimate the decrease in loss resulting from applying each morphism. We then optimize this loss to learn and evaluate morphism parameters. We use this method to design a NAS algorithm that iteratively applies network morphisms to progressively grow a network architecture.

We compare our method with other NAS methods on CIFAR-10 and CIFAR-100 classification tasks [20]. We present promising experiments that demonstrate that our method grows networks with similar or better parameter-accuracy tradeoff compared to similar methods.

(*a*) Channel splitting morphism (*b*) Channel pruning morphism

Figure 1: Network morphisms. Square nodes represent convolutional layers, circular nodes represent convolutional channels.

2. Related Work

Pruning methods [8, 11, 25, 26] have become popular for shrinking large, high-performing networks down to much smaller networks without sacrificing test accuracy. One-shot methods [30] are similar: they simplify NAS by constraining the search space to subgraphs of a large trained network. These methods are much less computationally expensive than reinforcement learning and evolutionary methods, but still require training a large network.

The computational inexpensiveness of growing progressively larger networks has been exploited for NAS [10, 22, 33] and for training fixed networks [17]. Growing networks via network morphisms has previously been used in combination with reinforcement learning [2] and evolutionary NAS methods [6]. In contrast, we use network morphisms to view NAS as a continuous optimization problem, similar to other differentiable architecture search methods [23, 27, 34]. Unlike Net2Net [4], which applies network morphisms with random parameters, we build upon a recent line of work [24, 38–40] that has made progress in efficiently learning and evaluating morphisms.

We use a Gauss-Newton approximation to estimate the decrease in the loss achieved by applying a network morphism. Bayesian optimization NAS methods [16, 19, 22, 28] also try to estimate the performance of new networks without training them. However, these methods use the performance of previously seen networks to predict the performance of future unseen networks, while our predictions are made independently of previously seen networks. In fact, our technique is more similar to [21] in which the authors use a diagonal approximation of the Hessian to estimate the change in the loss when pruning neurons.

3. Method

3.1. Morphisms

A *network morphism* is a small change in a neural net's architecture parameterized by θ so that when $\theta = 0$, the morphism is function-preserving, i.e., the input-output mapping of the neural network is unchanged. In this paper, we consider several morphisms.

The first morphism we consider is a channel-splitting morphism that grows a network wider, depicted in Figure 1(*a*)subfigure. For a convolutional channel y with input from a layer x with incoming kernel parameters w_{in} and output to a layer z with outgoing kernel parameters w_{out}, applying the channel-splitting morphism replaces the channel y with two channels y_1 and y_2, with incoming kernel parameters $w_{in} + \theta$ and $w_{in} - \theta$ respectively, and each with outgoing kernel pa-

rameters $w_{out}/2$. If $\theta = 0$, then this morphism duplicates the channel y without changing the input-output mapping of the neural network; if $\theta \neq 0$, then this morphism replaces the feature detected by y with two new feature detectors with parameters $w_{in} + \theta$ and $w_{in} - \theta$. For example, if y is an edge detector, then y_1 and y_2 may detect two similar edges with slightly different angles.

The second morphism we consider is a channel-pruning morphism, depicted in Figure 1(*b*)subfigure. For a channel y with incoming kernel parameters w_{in}, applying the channel-pruning morphism subtracts θ from w_{in}. If $\theta = 0$, then the network is unchanged, but if $\theta = w_{in}$, then the incoming kernel parameters of y are zero, and the channel y can be pruned. The parameters of this morphism are not learned, but instead are always chosen to be $\theta = w_{in}$.

To apply a particular morphism, we must first choose values for the morphism's parameters θ. The best choice for the morphism's parameters would maximally decrease the loss of the network when the morphism is applied. However, these parameters are computationally prohibitive to calculate exactly at scale.

3.2. Gauss-Newton Approximation

Instead, we can approximate the decrease in the loss function for each morphism. Each morphism we consider is *local*, so that there exists a collection of network activations z such that the mapping between the network input and any activation higher than z in the computational DAG is unchanged for any choice of θ. Consider the expanded networks depicted in Figure 1. Denote $\Delta\mathcal{L}(\theta)$ the change in the loss function after applying the morphism with parameters θ; $\Delta z(\theta)$ the change in z after applying the morphism with parameters θ; g the gradient of the loss function with respect to z at $\theta = 0$; and H the Hessian of the loss function with respect to z at $\theta = 0$. Consider the second-order approximation of the change in loss function with respect to z centered at $\theta = 0$:

$$\Delta\mathcal{L}(\theta) \approx \Delta z(\theta) \cdot g + \frac{1}{2}\Delta z(\theta)^\top H \Delta z(\theta)$$

Computing the Hessian matrix of second derivatives in this approximation is computationally expensive. Instead, we can make a Gauss-Newton approximation of the Hessian matrix, where $\hat{\mathcal{L}}$ is the current training loss:

$$H \approx \frac{1}{2\hat{\mathcal{L}}} g g^\top$$

Plugging in this approximation yields:

$$\Delta\mathcal{L}(\theta) \approx \Delta z(\theta) \cdot g + \frac{1}{4\hat{\mathcal{L}}}(\Delta z(\theta) \cdot g)^2$$

Recent work [38, 40] also uses a second-order approximation of the loss function to learn morphism parameters. In that work, the authors make a second-order approximation of the loss function with respect to θ. In contrast, we make a second-order approximation of the loss function with respect to $\Delta z(\theta)$. Critically, because $\Delta z(\theta)$ is a nonlinear function of θ, our Gauss-Newton approximation is still a high-order approximation of $\Delta\mathcal{L}(\theta)$ with respect to θ.

3.3. Algorithm

Since $\Delta\mathcal{L}(\theta)$ is computed independently for each training mini-batch, we record an exponential moving average of $\Delta\mathcal{L}(\theta)$ across mini-batches using a momentum hyperparameter to get a lower

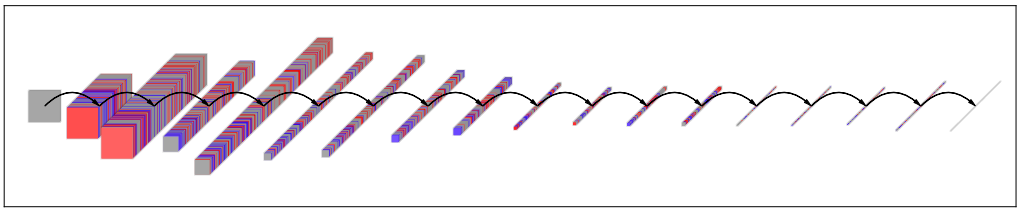


Figure 2: Network grown from a VGG-19 seed network by our algorithm for classifying CIFAR-100. Here the network is at the end of its 15-th growth phase. Channels colored red will be split with their learned channel-splitting morphism parameters in the next epoch; splitting the reddest channels is estimated to give the highest loss-resource tradeoff. Channels colored blue will be pruned in the next epoch; pruning the bluest channels is estimated to give the highest loss-resource tradeoff.

variance estimate of the decrease in the loss function. We can then weigh the tradeoff for each morphism between the estimated change in loss $\Delta\mathcal{L}(\theta)$ and the change in the number of parameters introduced when applying the morphism. To quantify this tradeoff, we introduce a regularization hyperparameter indicating the desired tradeoff between training loss and model size. Then we say a morphism has a positive loss-resource tradeoff if

$$-\Delta\mathcal{L}(\theta) > \lambda_p \Delta R_p$$

where λ_p is a hyperparameter regularization constant on the model size and ΔR_p is the change in the number of parameters resulting from applying the morphism. Given the exponential moving average estimate of $\Delta\mathcal{L}(\theta)$, checking whether a morphism has positive loss-resource tradeoff takes constant time.

We use our Gauss-Newton approximation to design a NAS algorithm for growing networks, summarized in Algorithm 1 in the appendix. Our algorithm alternates between a training phase and a growing phase. In all our experiments, each phase lasts 20 epochs. In the training phase, the model architecture is frozen while the model parameters are optimized to minimize the training loss. In the growing phase, the model parameters are frozen while morphism parameters are optimized to minimize our Gauss-Newton approximation of the loss. After morphism parameters are learned, we compute each morphism's loss-resource tradeoff. Then for each layer, we apply the top 30% of morphisms local to that layer with positive loss-resource tradeoff.

4. Experiments

In all our experiments, we train with a batch size of 64 and use a simple data augmentation scheme for CIFAR-10 and CIFAR-100: random horizontal flips and random crops with padding 4. In the appendix, we present additional experiments evaluating the accuracy of our Gauss-Newton approximation and the quality of our learned morphism parameters.

Here we compare our NAS algorithm end-to-end with other methods for learning architectures for classifying CIFAR-10 and CIFAR-100. We experiment with different choices of the loss-resource tradeoff hyperparameter to grow networks of many different sizes. We grow networks

from one of two seed networks. The first is a VGG-19 network with 16 channels in each convolutional layer. The second is a MobileNetV1 network with 32 channels in each convolutional layer. In each experiment, we run our algorithm for a total of 30 training and growing phases. We optimized model parameters using SGD with Nesterov momentum 0.9, weight decay 10^{-4}, and a learning rate that begins at 0.1 and decreases by a factor of 10 at epochs 300 and 450. We optimized morphism parameters with Adam and a learning rate of 10^{-2}. After our algorithm terminates, we reinitialize the network's model parameters and retrain the model from scratch to more accurately determine the best test accuracy achievable for the learned architecture.

A visualization of a network grown by our algorithm from a VGG-19 seed network for classifying CIFAR-100 is in Figure 2. It is worthwhile to point out that growing from a uniform-width seed network, our algorithm naturally discovers that a unique, bottleneck-shaped architecture provides the best loss-parameter tradeoff.

Next, we report the results for CIFAR-10 and CIFAR-100 classification tasks in Tables 1 and 2, respectively. We compare with other NAS methods as well as human-designed baselines. We observe that our method produces networks with similar or better parameter-accuracy tradeoff at a smaller computational cost. For example, a network we grew from a VGG-19 seed network using $\lambda_p = 3 \times 10^{-7}$ achieved 5.6% test error on CIFAR-10 using only 1.2 million parameters, which achieves lower test error with fewer parameters compared to [25], which pruned a VGG-19 model down to 2.3 million parameters and achieved 6.2% test error. Similarly, a network we grew from a MobileNetV1 seed network using $\lambda_p = 3 \times 10^{-7}$ achieved 25.9% test error on CIFAR-100 using only 1.4 million parameters, which achieves lower test error with fewer parameters compared to [12], a ResNet that achieves 27.2% test error with 1.7 million parameters.

Note that our method of growing from simple VGG-19 and MobileNetV1 networks with simple channel splitting and pruning morphisms is not enough to outperform complex architectures like those produced by NASNET. Architecture elements necessary for high performance, like residual connections and squeeze-excite modules, make growing complicated because they force several layers to have the same number of channels, disallowing us from splitting channels in different layers independently. It may be possible to grow from these types of seed networks using more complex morphisms that split channels in multiple layers jointly, but this is left for future work.

5. Conclusion

In this paper, we presented a neural architecture search method for growing a network with network morphisms while training. We used a Gauss-Newton approximation of the loss to learn morphism parameters and to estimate the change in the loss resulting from applying those morphisms. We used the estimated change in loss to compute a loss-resource tradeoff for each morphism using hyperparameters that regularized the number of parameters of the grown network. We compared our method with state of the art NAS methods for classifying CIFAR-10 and CIFAR-100 and concluded that our algorithm finds similar or better architectures at a smaller computational cost.

Method Type	Reference	Error (%)	Params (Millions)	Reachable	GPU time (days)
SOTA	AmoebaNet-A [32]	3.3	3.2		3150
	NASNET-A [42]	3.4	3.3		2000
	Large-scale Evolution [31]	5.4	5.4		2600
Morphisms	NASH [5]	5.2	19.7	✓	1.0
	Slimming [25]	6.2	2.3	✓	-
	Firefly [39]	6.2	1.9	✓	-
	Net2Net [4]	6.5	3.9	✓	2.1
Human-Designed	DenseNet [15]	3.5	25.6		N/A
	VGG-19 Baseline [35]	6.3	20.0	✓	N/A
	ResNet [14]	6.4	1.7		N/A
	MobileNetV1 Baseline [13]	6.6	3.2	✓	N/A
Ours	Seed VGG-19, $\lambda_p = 3 \times 10^{-7}$	5.6	1.2	✓	0.7
	Seed VGG-19, $\lambda_p = 1 \times 10^{-6}$	6.5	0.6	✓	0.5
	Seed MobileNetV1, $\lambda_p = 3 \times 10^{-8}$	5.8	0.8	✓	1.0
	Seed MobileNetV1, $\lambda_p = 3 \times 10^{-7}$	6.0	0.5	✓	1.0
	Seed MobileNetV1, $\lambda_p = 1 \times 10^{-6}$	6.2	0.4	✓	0.7

Table 1: Classification performance of various architectures on CIFAR-10

Method Type	Reference	Error (%)	Params (Millions)	Reachable	GPU time (days)
SOTA	Large-scale Evolution [31]	23.0	40.4		-
	SMASH [1]	22.1	4.6		-
Morphisms	NASH [5]	23.4	22.3	✓	1.0
	Slimming [25]	26.5	5.0	✓	-
Human Designed	DenseNet [15]	17.2	25.6		N/A
	Resnet [12]	27.2	1.7		N/A
	VGG-19 Baseline [35]	27.6	20.1	✓	N/A
	MobileNetV1 Baseline [13]	28.7	3.3	✓	N/A
Ours	Seed VGG-19, $\lambda_p = 3 \times 10^{-7}$	27.2	2.2	✓	0.7
	Seed VGG-19, $\lambda_p = 6 \times 10^{-7}$	28.0	1.6	✓	0.6
	Seed MobileNetV1, $\lambda_p = 1 \times 10^{-6}$	27.2	0.8	✓	0.8
	Seed MobileNetV1, $\lambda_p = 6 \times 10^{-7}$	26.9	1.3	✓	1.0
	Seed MobileNetV1, $\lambda_p = 3 \times 10^{-7}$	25.9	1.4	✓	1.0

Table 2: Classification performance of various architectures on CIFAR-100

References

[1] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[2] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[3] Arnav Chavan, Deepak Gupta, et al. Beyond uniform scaling: Exploring depth heterogeneity in neural architectures. *arXiv preprint arXiv:2402.12418*, 2024.

[4] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

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

[6] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018.

[7] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019. URL http://jmlr.org/papers/v20/18-598.html.

[8] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[9] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3224–3234, 2019.

[10] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1586–1595, 2018.

[11] 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.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] 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.

[14] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[15] 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.

[16] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, 5, 2018.

[17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[18] Ryan King and Bobak Mortazavi. Growing representation learning. *arXiv preprint arXiv:2110.08857*, 2021.

[19] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. 2016.

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[21] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[22] 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.

[23] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[24] Qiang Liu, Lemeng Wu, and Dilin Wang. Splitting steepest descent for growing neural architectures. *arXiv preprint arXiv:1910.02366*, 2019.

[25] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

[26] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[27] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. *arXiv preprint arXiv:1808.07233*, 2018.

[28] Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.

[29] Vladimir Nekrasov, Hao Chen, Chunhua Shen, and Ian Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9126–9135, 2019.

[30] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018.

[31] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

[32] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[33] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[34] Richard Shin, Charles Packer, and Dawn Song. Differentiable neural network architecture search. 2018.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[36] Ruilin Tong. Growing neural network with shared parameter. *arXiv preprint arXiv:2201.06500*, 2022.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[38] Dilin Wang, Meng Li, Lemeng Wu, Vikas Chandra, and Qiang Liu. Energy-aware neural architecture optimization with fast splitting steepest descent. *arXiv preprint arXiv:1910.03103*, 2019.

[39] Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. Firefly neural architecture descent: a general approach for growing neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[40] Lemeng Wu, Mao Ye, Qi Lei, Jason D Lee, and Qiang Liu. Steepest descent neural architecture optimization: Escaping local optimum with signed neural splitting. *arXiv preprint arXiv:2003.10392*, 2020.

[41] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018.

[42] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

Appendix A. Algorithm Pseudocode

Algorithm 1: Growing networks with Gauss-Newton

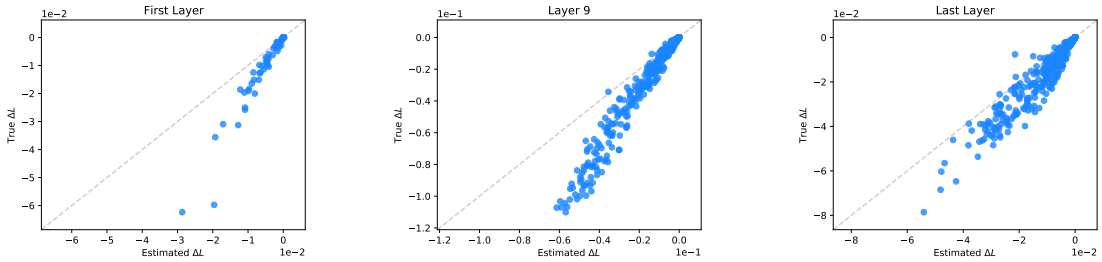
Data: Dataset D, model M, phase length n_{phase}
for $t = 1, \ldots, n_{phase}$ **do**
  foreach *mini-batches* $\{d_s\} \in D$ **do**
    Compute mini-batch loss $\mathcal{L} = M(\{d_s\})$;
    Compute all $\nabla_w \mathcal{L}$ with backprop;
    SGD step model parameters w;
  end
end
for $t = 1, \ldots, n_{phase}$ **do**
  foreach *mini-batches* $\{d_s\} \in D$ **do**
    Compute all $\Delta\mathcal{L}(\theta)$ and $\nabla_\theta \mathcal{L}$ with backprop;
    Update exponential moving average of $\Delta\mathcal{L}(\theta)$;
    SGD step morphism parameters θ;
  end
end
foreach *top* 30% *morphisms with positive tradeoff* **do**
  Apply morphism;
end

Appendix B. Gauss-Newton Approximation Accuracy

Here we evaluate the accuracy of our Gauss-Newton approximation of the loss. We begin by constructing a VGG-19 model for CIFAR-10 and equip it with channel-splitting morphisms, one for each channel in each convolutional layer in the network. We trained the VGG-19 model with SGD with learning rate 0.1 for 20 epochs while holding morphism parameters constant. Then we updated morphism parameters with Adam with learning rate 10^{-2} for 20 epochs while updating the exponential moving average estimate of $\Delta\mathcal{L}$ using momentum hyperparameter $\frac{64}{50000} \times \frac{1}{2}$ so that our estimate of $\Delta\mathcal{L}$ is approximately an average over the last 2 epochs. We then computed the true change in loss achieved by each morphism with its current parameters by applying each morphism to construct an independent expanded network and evaluating that expanded network on the test dataset. We then compared our exponential moving average estimate of $\Delta\mathcal{L}$ with the true value.

The results are depicted in Figure 3. Each circle represents a single channel-splitting morphism in the specified layer. There are 64 channels in the first layer of the VGG-19 model, and 512 channels each in the 9-th and last layers. The figure plots our exponential moving average estimate of $\Delta\mathcal{L}$ against the true $\Delta\mathcal{L}$ computed via brute force. If our method were 100% accurate, all circles would lie on the grey dashed lines.

The figure shows that the Gauss-Newton approximation used by our algorithm is quite accurate. This result by itself is significant. Other methods expend enormous computational resources trying to estimate how the loss of a network changes when channels are added or removed from the net-



(*a*) (*b*) (*c*)

Figure 3: Estimated versus actual decrease in loss for morphisms learned while holding model parameters constant.

work. This result shows that the change in loss can be approximated to a high degree of accuracy using only statistics of the network, namely $\Delta z(\theta) \cdot g$.

We also observe that the Gauss-Newton approximation seems to be most accurate for the layer closest to the network output, and least accurate for the layer closest to the network input, though the reason for this behavior is unclear.

We conclude that the Gauss-Newton approximation used by our algorithm estimates $\Delta\mathcal{L}$ for each morphism to a high degree of accuracy.

Appendix C. Learned Morphism Quality

Here we compare the quality of our learned morphisms to those learned via other methods. Another method for learning morphism parameters is to apply the morphism to construct an expanded network, then optimize the loss of the expanded network with respect to the morphism parameters. This allows us to learn morphism parameters that minimize the loss rather than an approximation of the loss, but is computationally expensive to scale when there are many morphisms under consideration.

Another method for choosing morphism parameters is to use the steepest descent direction as in [38–40]. However, the steepest descent direction does not indicate the optimal scale for θ. To approximately compute the optimal scale, we perform a line search along the steepest descent direction, though this is computationally expensive.

We compare the true decrease in loss achieved by the morphism parameters learned by our algorithm with the true decrease in loss achieved by the morphism parameters produced by the two baselines described above. We do this for each of the possible 64 channel-splitting morphisms in the first layer of the VGG-19 network trained in the previous experiment. The result is in Figure 4. Each 3-bar cluster plots the true decrease in loss achieved by the morphism parameters learned by each method for the corresponding channel-splitting morphism. For ease of viewing, we have sorted the channels with respect to the true decrease in loss achieved by the first baseline method. Note that for some channels, none of the methods are able to find good morphism parameters. After inspecting these features, we observe that at this point in training (epoch 20), those channels have already "died" due to L_2 weight regularization, so it is likely not possible to split such bad feature detectors into two good feature detectors.

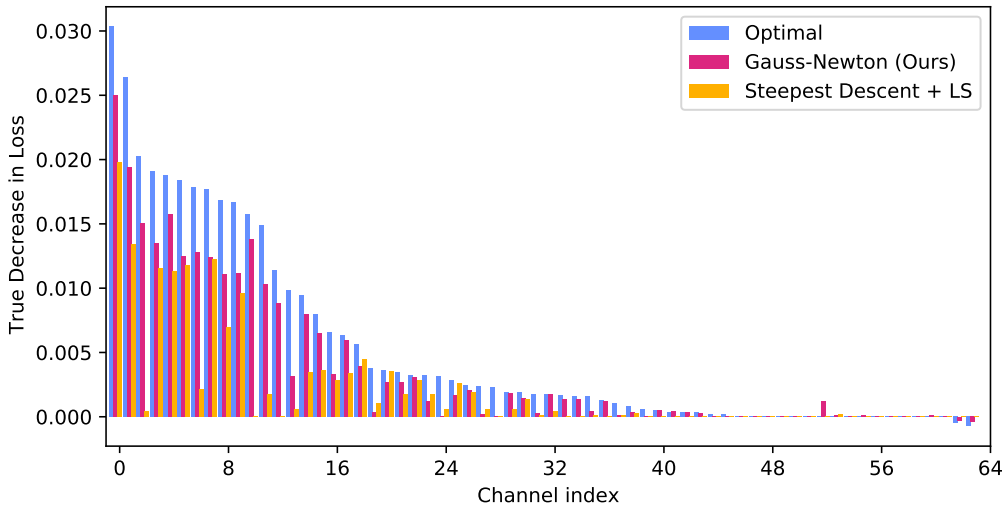


Figure 4: Comparison of different morphism learning strategies. Each 3-bar cluster plots the true decrease in loss achieved by the channel-splitting morphism learned by each method for one of the 64 channels in the first layer of VGG-19.

From the figure, we observe that the morphisms learned by our method most often achieve a greater decrease in loss than those learned by the steepest descent with line search baseline method. We also observe that the true decrease in loss achieved by our learned morphisms most often comes within a constant factor of the decrease achieved by the expensive network expansion baseline. We observe this most often among the morphisms with the highest potential decrease in loss; this is important, since these are the morphisms that will be selected by our algorithm to be applied to grow the network. We conclude that our algorithm learns high quality morphisms, on par with the expensive network expansion baseline method.

Appendix D. Gauss-Newton Approximation

In this section we review the justification for Gauss-Newton approximation. We begin by assuming that the loss function is well-approximated by a least-squares problem in z, i.e., for some matrix A and vector b,

$$\begin{aligned}\mathcal{L}(z) &\approx \frac{1}{2}\|Az - b\|_2^2 \\ &= \frac{1}{2}b^\top b - z^\top A^\top b + \frac{1}{2}zA^\top Az.\end{aligned}$$

Denote the residual:

$$r = Az - b$$

Note that $\mathcal{L} = \frac{1}{2} r^\top r$. Denote the gradient and Hessian of the loss:

$$g = A^\top r \qquad\qquad H = A^\top A$$

Consider the change in the loss function when adding a quantity Δz to z. Denote the change in loss:

$$\begin{aligned}\Delta\mathcal{L}(\Delta z) &\equiv \mathcal{L}(z + \Delta z) - \mathcal{L}(z) \\ &= \Delta z \cdot g + \frac{1}{2}\Delta z^\top A^\top A \Delta z \\ &= \Delta z \cdot g + \frac{1}{2}\Delta z^\top H \Delta z\end{aligned}$$

In this paper, we write the Gauss-Newton approximation as

$$H \approx \frac{1}{2\mathcal{L}} g g^\top,$$

Theorem 1 (General Gauss-Newton Approximation) *If $\Delta z = \lambda \Delta z^*$ for some $\lambda \in \mathbb{R}$ and some Δz^* satisfying $A(z + \Delta z^*) = b$, then*

$$\frac{1}{2}\Delta z^\top H \Delta z = \frac{1}{2}\Delta z \frac{g g^\top}{2\mathcal{L}} \Delta z$$

Proof If $\Delta z = \lambda \Delta z^*$ and $A(z + \Delta z^*) = b$, then $A\Delta z = -\lambda r$. So

$$\begin{aligned}&\frac{1}{2}\Delta z^\top \left(\frac{g g^\top}{2\mathcal{L}}\right) \Delta z \\ =\ &\frac{1}{2}\Delta z^\top \left(\frac{A^\top r r^\top A}{r^\top r}\right) \Delta z \\ =\ &\frac{1}{2}\lambda^2 r^\top r\end{aligned}$$

Similarly,

$$\begin{aligned}\frac{1}{2}\Delta z^\top H \Delta z &= \frac{1}{2}\Delta z A^\top A \Delta z \\ &= \frac{1}{2}\lambda^2 r^\top r\end{aligned}$$

■

Therefore, we say the Gauss-Newton approximation is exact in the space spanned by the solutions Δz^* to the linear system $A(z + \Delta z^*) = b$.

Theorem 2 (Rank-1 Gauss-Newton Approximation) *If H is rank-1 and there exists a solution z^* to the linear system $Az^* = b$, then for all Δz,*

$$\frac{1}{2}\Delta z^\top H \Delta z = \frac{1}{2}\Delta z \frac{g g^\top}{2\mathcal{L}} \Delta z$$

Proof If H is rank-1, then A consists of a single row $u^\top$ and $b \in \mathbb{R}$ is a scalar.

Let Δz be arbitrary. Then there exists a solution Δz^* to the linear system $A(z + \Delta z^*) = b$ and $\lambda \in \mathbb{R}$ such that $\Delta z = \lambda \Delta z^*$, namely

$$\Delta z^* = \frac{b - u^\top z}{u^\top z} \Delta z$$
$$\lambda = \frac{u^\top \Delta z}{b - u^\top z}$$

since then

$$\begin{aligned} A(z + \Delta z^*) &= u^\top z + u^\top \Delta z^* \\ &= u^\top z + u^\top \left(\frac{b - u^\top z}{u^\top z} \Delta z \right) \\ &= b \end{aligned}$$

Applying the previous theorem yields the result. ■

From this it is clear that we can expect the Gauss-Newton approximation to be quite accurate if the true Hessian matrix H is low-rank.