Neural Growth and Pruning in Dynamic Learning Environments

Anonymous[1]

[1]Anonymous Institution

Abstract Training artificial neural networks (ANNs) under the shifting distributions of dynamic learning environments can be augmented by dynamic architectural adjustments in addition to the standard parameter tuning. Towards effective yet efficient models, we study neural grow-and-prune in the basic shifting distribution case of transfer learning: adapting a generically pre-trained model to a target dataset. We propose the DYNO (**Dy**namic **N**eural **O**ptimization) grow-and-prune algorithm, which dynamically performs neural pruning to remove neurons that are dormant or redundant and neural growth to add orthogonal features during fine-tuning. Our experiments across a variety of transfer tasks show that DYNO yields an efficient yet performant network with dynamically shaped layers.

1 Introduction

Artificial neural networks (ANNs) are typically hand-architected with parameters trained from scratch. However, this paradigm is not scalable with the growing desire to apply ANNs in more complex and even dynamic tasks, where the training data distribution may change over time. The simplest form of a dynamic learning environment is transfer learning, where an ANN may be generally pre-trained on a source dataset before fine-tuning a specific copy on the target dataset, improving computational reuse and even performance on the target task. Fine-tuning usually consists of the same generic training approach of gradient descent with a static architecture.

As intermediate activations are effective for out-of-distribution (OoD) detection [Hein et al., 2019], the inverse suggests they could also inform neural growth [Maile et al., 2022] and pruning. During fine-tuning on the target task, some existing features may be more useful or adaptable to the target task than others: neurons that are redundant with others or show OoD activation patterns such as dormancy [Sokar et al., 2023] may be pruned and new neurons with incoming distribution-aware initialization may be added. This strategically adapts the model to the new domain by augmenting the most useful components of the inductive bias transferred from pre-training.

We propose and investigate the utility of growing and pruning neurons during transfer learning, resulting in a fine-tuned model with a customized architecture for the target task. We study the use of informed scheduling for the growth and pruning operations, removing the hand-designed schedules and algorithmic details seen in other comparable approaches. We define the DYNO (**Dy**namic **N**eural **O**ptimization) grow-and-prune algorithm: the proposed strategies that comprise DYNO only use activation information from just the forward pass, without any masking or gradient calculations. We study DYNO in multiple transfer learning scenarios, analyzing how different layers respond in across various transfer contexts. DYNO yields dynamically shaped efficient models with tuned layer widths and tuned parameters, without predetermined architectural scheduling.

2 Problem Definition

An artificial neural network (ANN) f may be optimized through empirical risk minimization of a loss function L for a dataset D consisting of inputs x and outputs y:

$$\arg\min_{f} \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim D}\, L(f(\boldsymbol{x}), \boldsymbol{y}). \tag{1}$$

Algorithm 1 A general framework for neural grow-and-prune optimization.

```
procedure GROW-AND-PRUNE(TRIG_GROW, INIT_GROW, TRIG_PRUNE, SELECT_PRUNE, initial ANN $f$)
    while $f$ not converged do
        for each hidden layer $l$ do
            if TRIG_PRUNE$(f, l) > 0$ then
                Remove TRIG_PRUNE$(f, l)$ neurons by SELECT_PRUNE$(f, l)$
            if TRIG_GROW$(f, l) > 0$ then
                Add TRIG_GROW$(f, l)$ neurons using INIT_GROW$(f, l)$
        for $n$ steps do
            Gradient descent step on current existing weights and dataset
    return trained $f$
```

For a dense multi-layer perceptron (MLP) with d hidden layers, f may be expressed as $f(\boldsymbol{x}) = \sigma_{d+1}(\boldsymbol{W}_{d+1}\sigma_d(...\boldsymbol{W}_2\sigma_1(\boldsymbol{W}_1\boldsymbol{x})...))$, where σ_l is a nonlinear activation function that also adds a row for the bias, and $\boldsymbol{W}_l \in \mathbb{R}^{M_l \times (M_{l-1}+1)}$ is the weight matrix for the l^{th} layer, including the bias parameters. The M_l rows of $\boldsymbol{W}_l$ each represents the fan-in weights of a neuron in layer l receiving input from each of the M_{l-1} preceding neurons, while each of first M_{l-1} columns represents the fan-out weights of a layer $l-1$ neuron and the last column represents the biases.

We define $\boldsymbol{h}_l = \sigma_l(\boldsymbol{W}_l\sigma_{l-1}(...\boldsymbol{W}_2\sigma_1(\boldsymbol{W}_1\boldsymbol{x})...))$ as the post-activations of layer l. For n samples, $\boldsymbol{H}_l \in \mathbb{R}^{M_l \times n}$ is the post-activation matrix.

For convolutional layers, we consider that a channel is analogous to a neuron in a dense layer. However, both the activation for a channel and a single sample as well as the parameterization of the connection between two channels are matrices instead of single values. Thus, $\boldsymbol{H}_l \in \mathbb{R}^{H_l \times W_l \times M_l \times n}$ and $\boldsymbol{W}_l \in \mathbb{R}^{k_l \times k_l \times M_l \times (M_{l-1}+1)}$.

Standard ANN training procedures pre-define the size and structure of the weight matrices W, updating their parameters via stochastic gradient descent over mini-batches sampled from the current dataset towards optimizing Equation (1).

To perform neurogenesis or neural pruning, the addition or removal of k neurons to the l^{th} layer is accomplished by appending or removing k rows of fan-in weights in $\boldsymbol{W}_l$ and k columns of fan-out weights in $\boldsymbol{W}_{l+1}$. Utilizing these operations in the empirical risk minimization of Equation (1) defines the neural grow-and-prune optimization. We restrict the search space of where to add new neurons to within existing layers.

3 Grow and Prune for Transfer Learning

Our general proposed framework for neural grow-and-prune is presented in Algorithm 1. This framework cycles through each optimization operation of pruning, growth, and parameter optimization via gradient descent, allowing the grow-and-prune details such as the trigger strategies, growth initialization strategy, and pruning selection strategies to be defined alongside other training details such as the gradient descent optimizer. We propose the **Dy**namic **N**eural **O**ptimization (DYNO) grow-and-prune algorithm with the strategies defined as follows.

Following Maile et al. [2022], we define triggers and initialization/selection strategies for each operation of growing neurons and pruning neurons. The trigger is evaluated regularly to determine, for each layer, how many neurons to grow or prune. The initialization strategy for growing or selection strategy is then applied to carry out the structural change.

The trigger for both operations is based on the orthogonality metric of *effective dimensionality*, the ϵ-numerical rank of the post-activation matrix [Kumar et al., 2021, Lyle et al., 2022]. For layer l and n samples generating the post-activation $\boldsymbol{H}_l$, the effective dimension metric ϕ_{ED} may be estimated by

$$\phi_{ED}\left(f,l\right) = \frac{1}{M_l}\left|\left\{\sigma \in \mathrm{SVD}\left(\frac{1}{\sqrt{n}}\boldsymbol{H}_l\right)\middle|\sigma > \epsilon\right\}\right|, \tag{2}$$

where $\mathrm{SVD}\left(\frac{1}{\sqrt{n}}\boldsymbol{H}_l\right)$ is the set of singular values of $\frac{1}{\sqrt{n}}\boldsymbol{H}_l$ and $\epsilon > 0$ is a small threshold. For convolutional layers, $\boldsymbol{H}_l$ is flattened to $\mathbb{R}^{M_l\times(H_l W_l n)}$, so that the metric is relative to the number of neurons or channels in layer l.

A significant increase in the orthogonality metric is used to trigger neurogenesis, while a significant decrease is used to trigger pruning:

$$\textsc{Trig}_{\textsc{Prune}}\left(f,l\right) = \min\left(0, \left\lfloor M_l\left((1-\delta)\phi^*_{ED} - \phi_{ED}\left(f,l\right)\right)\right\rfloor\right), \tag{3}$$
$$\textsc{Trig}_{\textsc{Grow}}\left(f,l\right) = \min\left(0, \left\lfloor M_l\left(\phi_{ED}\left(f,l\right) - (1+\delta)\phi^*_{ED}\right)\right\rfloor\right), \tag{4}$$

where $\delta > 0$ is a small threshold and ϕ^*_{ED} is the value of $\phi_{ED}\left(f,l\right)$ at the last structural change of f. This combination of trigger strategies slims the layer when its activations are expressible in fewer dimensions, and expands it when the feature space in near expressible capacity. The updating baseline allows structural operations to occur dynamically in response to changes in the training distribution, with minimal dependence on the frequency of trigger evaluation.

Neurons to prune are selected based on a greedy cosine similarity algorithm, $\textsc{Select}_{\textsc{Prune}}(f,l)$. In the case of ReLU-activated layers, all completely dormant neurons are first pruned. Then, until the total number of neurons to prune is reached, the neuron with the highest norm of cosine similarities of normalized post-activations with other remaining neurons is greedily pruned.

New neurons are grown with the initialization strategy, $\textsc{Init}_{\textsc{Grow}}(f,l)$, as in NORTH-Select [Maile et al., 2022]: from candidates with scaled random fan-in vectors and zeroed fan-out vectors, select neurons that independently maximize $\phi_{ED}\left(f,l\right)$ with the existing neurons in that layer.

4 Experiments

We apply neural grow-and-prune transfer learning on three classification scenarios, each beginning with VGG11 pretrained on Imagenet [Russakovsky et al., 2015]. The target datasets are:

Imagenette: 10 easily distinguishable classes of Imagenet [Howard], thus making high-level features useful but not detailed features that may distinguish more similar classes.

Imagewoof: 10 dog breed classes of Imagenet [Howard], thus requiring minute details within a subspace of the original distribution.

Galaxy10: satellite images of galaxies in 10 shape-based categories [Leung and Bovy, 2019], so the underlying distribution is very different from natural images as in Imagenet.

Training details may be found in Appendix A. We additionally implemented Surgical Fine-Tuning (SFT) [Lee et al., 2023], which dynamically adjusts layer-wise learning rates based on the layer's relative gradient norm, as an orthogonal approach to transfer learning. The analysis of the interplay of DYNO and SFT can be found in Appendix B.

The results of applying DYNO compared to standard static fine-tuning are shown in Figure 1. When comparing test accuracy to inference FLOPs, the fine-tuned models have slightly lower average accuracy but are consistently more efficient in compute cost. Network sizes are consistent within tasks: models for Galaxy10 are about half as expensive as the original pre-trained model. DYNO causes immediately large architectural changes, then progressively smaller later in training, approaching architectural convergence but allowing small amounts of turnover well after the distribution change. Considering layer-wise architectural changes of DYNO, models across all three tasks had most change occur in the later layers of the network, particularly the final convolution (layer 7) and final dense layer before the classification layer (layer 9). Models for Galaxy10 had significantly more architectural change, both for pruning and growing, and in more layers.

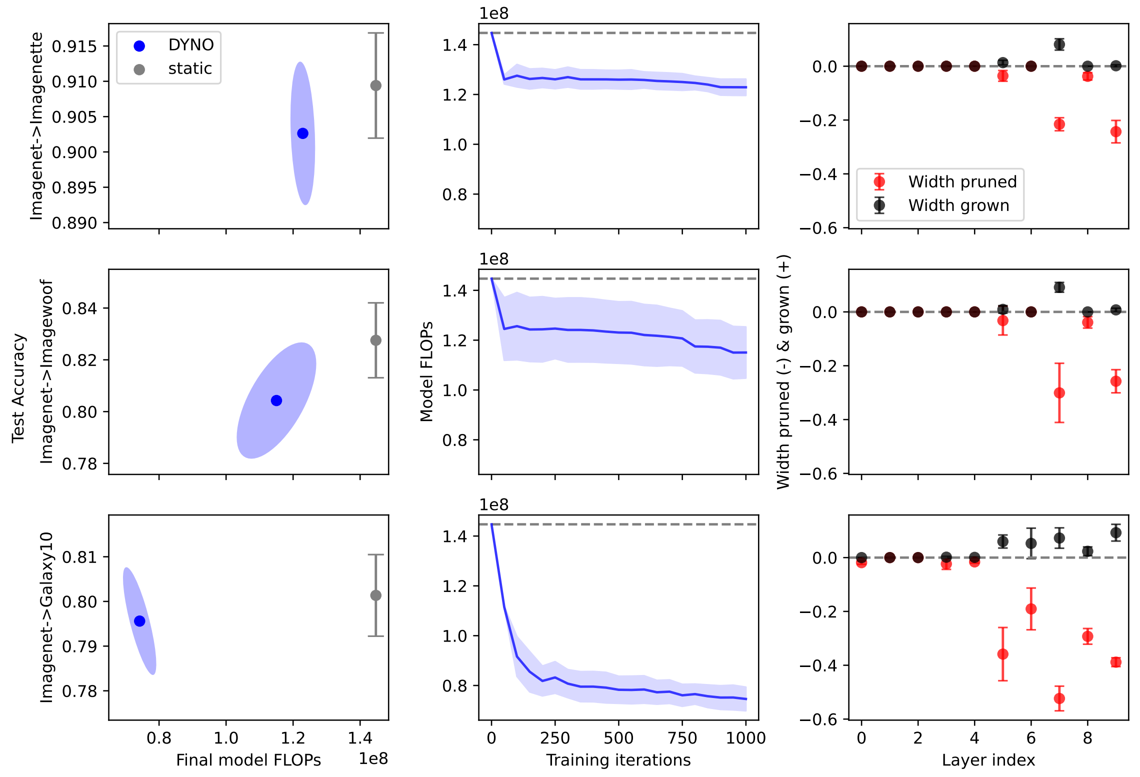

Figure 1: Experimental results for transfer to Imagenette (top), Imagewoof (middle), and Galaxy10 (bottom). Left: final test accuracy against final model inference FLOPs. Center: progression of inference cost over the course of training. Right: total neurons grown and pruned per layer over the course of DYNO, relative to the original layer width. All covariance ellipses, error clouds, and error bars show standard deviation across the 5 trials.

To fine-tune on a similar task, such as transferring to Imagenette, DYNO made very few changes, similar to what would happen with standard pruning of a final layers. However, when distribution shifts, such as transferring to Galaxy10, the architecture changes throughout: major modifications are made in the final layer, and there are even structural modifications in the early layers. This shows that DYNO responds to distribution shift based on the learned task to appropriately modify the architecture.

DYNO was not able to surpass the test accuracy of larger static models. This may be due to the simple linear schedule of architectural evaluation without a long final tuning phase or any learning rate decay, nor extensive hyperparameter tuning. Such "tricks" are often employed for improving performance but may confound results so thus were avoided in this preliminary study.

The strategies of DYNO are intended to minimize the number and impact of hyperparameters. For example, the moving baseline used in the trigger functions reduces dependence on the trigger evaluation frequency. Further work on hyperparameter optimization and specialization is warranted, such as learnable thresholds [Azarian et al., 2020] and layer-wise thresholds, which could improve test accuracy compared to static fine-tuning. We study layer-wise dynamic learning rates in Appendix B.

5 Discussion

The DYNO grow-and-prune algorithm shows dynamic layer-specific responsiveness for architectural changes during fine-tuning without the need for any hand-engineered bias, such has predetermined schedules of architectural change. This is a benefit of dynamically informed scheduling and minimal hyper-parameterization. This direction is important for the motivation of AutoML towards more automation and less human engineering.

Grow-and-prune algorithms have a very complex design space due to the interplay of growing, pruning, and training a model. This preliminary study only considered one intuition-guided instance within this space by defining each of $\textsc{Trig}_{\textsc{Grow}}$, $\textsc{Init}_{\textsc{Grow}}$, $\textsc{Trig}_{\textsc{Prune}}$, and $\textsc{Select}_{\textsc{Prune}}$, but the vastness prompts algorithmic optimization to find useful combinations of strategies. Previous automated algorithmic discovery methods such as Lange et al. [2023], Real et al. [2020] could be extended to include neural growth and pruning as potential operators during training.

Transfer learning is the first step from static supervised learning: further in the same direction include continual learning and multi-task learning. These dynamic paradigms benefit from domain generalization and avoiding catastrophic forgetting, which can be supported architecturally. Architectural optimization could help find efficient parameter sharing between distributions or tasks. Grow-and-prune may naturally be effective for alleviating the decay in plasticity normally observed in ANNs [Dohare et al., 2021]. To avoid catastrophic forgetting, the pruning mechanism may be modified: activation information could be used to mark which existing neurons could either be used with locked parameters or be copied and fine-tuned. These architectural techniques may additionally detect new contexts, as demonstrated by Hein et al. [2019], to trigger appropriate architectural adjustment, potentially incorporated with structurally aware representation disentanglement. These in tandem could ameliorate problems such as representation collapse [Lyle et al., 2022] and lack of generalization [Nikishin et al., 2022], in addition to finding dynamically constructed architectures for improved efficiency.

6 Additional Related Works

Previous neural grow-and-prune works often focus on the "where" and "how" of pruning, using generic schedules and neurogenesis initializations. The most comparable works that separately consider informed neural creation and pruning are Firefly [Wu et al., 2020] and NeST [Dai et al., 2019], which use gradient-based information. Further works perform less informed neural creation [Qiao et al., 2019, Du et al., 2019] or ephemeral pruning via masking [Wan et al., 2020]. Dai et al. [2020] studies synaptic grow-and-prune for incremental learning and Tung et al. [2017] performs unstructured pruning during fine-tuning, or fine-pruning. Structured fine-pruning has mostly been studied in the case of large language models (LLMs) driven by their prohibitively large training and inference costs when unpruned [Santacroce et al., 2023]. Zhao et al. [2023] performs structured pruning driven by redundancy metrics. No other known neural grow-and-prune works consider the transfer learning case, nor use only activation-based information.

7 Conclusion

Grow-and-prune during transfer learning and other dynamic learning environments is an intuitively useful yet challenging addition that warrants further investigation. With DYNO, we propose a framework for triggering pruning and growing strategies and demonstrate that this can result in structural changes during the learning process which respond to a downstream task distribution.

8 Broader Impact Statement

After careful reflection, we have determined that this work presents no notable negative impacts to society or the environment. We believe that this work supports the general practice of reusing pre-trained models instead of training from scratch, reducing compute costs and thus emissions.

9 Submission Checklist

1. For all authors…

(a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] All claims align with provided contributions, evidence, and scope.

(b) Did you describe the limitations of your work? [Yes] Only substantiated claims are made, noting limitations.

(c) Did you discuss any potential negative societal impacts of your work? [Yes] No negative impacts identified beyond those of basic ML research. See Section 8

(d) Have you read the ethics author's and review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/` [Yes] We confirm our paper confirms to all provided guidelines.

2. If you are including theoretical results…

   (a) Did you state the full set of assumptions of all theoretical results? [N/A] No theoretical results included. All formal math is defined.

   (b) Did you include complete proofs of all theoretical results? [N/A] No theoretical results included. All formal math is defined.

3. If you ran experiments…

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [N/A] Code may be released upon acceptance, but is not required for workshop submissions.

   (b) Did you include the raw results of running the given instructions on the given code and data? [N/A] Code may be released upon acceptance, but is not required for workshop submissions.

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [N/A] Code may be released upon acceptance, but is not required for workshop submissions.

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [N/A] Code may be released upon acceptance, but is not required for workshop submissions.

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] Training details are included in Appendix A.

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] All results included were run by us.

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] We compare DYNO to statically structured models. We leave ablation of the pruning and growing components as future work due to lack of time, potentially to be included in the camera-ready versin if accepted.

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes] All results included were run by us with identical protocols.

(i) Did you compare performance over time? [Yes] We compare accuracy vs. efficiency.

(j) Did you perform multiple runs of your experiments and report random seeds? [Yes] As detailed in the paper, all protocols have 5 trials with random seeds 1-5.

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All results are reported with error bars, error clouds, or covariance ellipses.

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [No] Surrogate benchmarks may be included in future work.

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUS, internal cluster, or cloud provider)? [No] Compute cost was not recorded: each trial was run on a GPU on the order of 1 hour.

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes] Hyperparameters were only informally tuned due to time constraints.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets…

   (a) If your work uses existing assets, did you cite the creators? [Yes] All datasets cited.

   (b) Did you mention the license of the assets? [Yes] See Appendix A.

   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] No new assets.

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] No new assets.

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No new assets.

5. If you used crowdsourcing or conducted research with human subjects…

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] No crowdsourcing or human subjects.

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] No crowdsourcing or human subjects.

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] No crowdsourcing or human subjects.

References

Kambiz Azarian, Yash Bhalgat, Jinwon Lee, and Tijmen Blankevoort. Learned threshold pruning. *arXiv preprint arXiv:2003.00075*, 2020.

Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. NeST: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019.

Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Incremental learning using a grow-and-prune paradigm with efficient neural networks. *IEEE Transactions on Emerging Topics in Computing*, 10(2):752–762, 2020.

Shibhansh Dohare, Richard S Sutton, and A Rupam Mahmood. Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*, 2021.

Xiaocong Du, Zheng Li, Yufei Ma, and Yu Cao. Efficient network construction through structural plasticity. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(3):453–464, 2019.

Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 41–50, 2019.

Jeremy Howard. Imagenette. `https://github.com/fastai/imagenette`.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *9th International Conference on Learning Representations*, 2021.

Robert Tjarko Lange, Tom Schaul, Yutian Chen, Tom Zahavy, Valenti Dallibard, Chris Lu, Satinder Singh, and Sebastian Flennerhag. Discovering evolution strategies via meta-black-box optimization. *11th International Conference on Learning Representations*, 2023.

Yoonho Lee, Annie S Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. Surgical fine-tuning improves adaptation to distribution shifts. *11th International Conference on Learning Representations*, 2023.

Henry W Leung and Jo Bovy. Deep learning of multi-element abundances from high-resolution spectroscopic data. *Monthly Notices of the Royal Astronomical Society*, 483(3):3255–3277, 2019.

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *10th International Conference on Learning Representations*, 2022.

Kaitlin Maile, Emmanuel Rachelson, Hervé Luga, and Dennis G. Wilson. When, where, and how to add new neurons to anns. In *International Conference on Automated Machine Learning*, pages 18–1. PMLR, 2022.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, pages 16828–16847. PMLR, 2022.

Siyuan Qiao, Zhe Lin, Jianming Zhang, and Alan L Yuille. Neural rejuvenation: Improving deep network training by enhancing computational resource utilization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 61–71, 2019.

Esteban Real, Chen Liang, David So, and Quoc Le. AutoML-Zero: Evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*. PMLR, 2020.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115 (3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Michael Santacroce, Zixin Wen, Yelong Shen, and Yuanzhi Li. What matters in the structured pruning of generative language models? *arXiv preprint arXiv:2302.03773*, 2023.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2023.

F. Tung, S. Muralidharan, and G. Mori. Fine-pruning: Joint fine-tuning and compression of a convolutional network with bayesian optimization. In *British Machine Vision Conference (BMVC)*, 2017.

Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12965–12974, 2020.

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*, 2020.

Chenglong Zhao, Yunxiang Zhang, and Bingbing Ni. Exploiting channel similarity for network pruning. *IEEE Transactions on Circuits and Systems for Video Technology*, 2023.

A Training Details

All experiments begin with the same Imagenet-pretrained VGG11 backbone, with standard layer widths. The results show the mean, standard deviation, and (when applicable) covariance across 5 trials each with different random seeds. For Galaxy10, the final classification layer is replaced with a reinitialized 10-way classification layer. For Imagenette and Imagewoof, the final classification layer is replaced with a 10-way classification layer consisting of the 10 respective classification neurons corresponding to each class. All trials are run for 1000 iterations with 64 samples per iteration, performing pruning and growth trigger evaluations every 50 iterations. The effective dimensionality threshold ϵ is 1e-2 and the trigger threshold δ is 3e-2. The optimizer is Adam with a learning rate of 3e-4 and weight decay of 1e-3. All metrics are evaluated on post-activations, which are tracked for each layer in a FIFO buffer that is dynamically sized to contain more samples than neurons, as necessary for the effective dimensionality calculation. Only forward pass information is required. The number of candidates generated for neural growth is $100 \times \textsc{Trig}_{\textsc{Grow}}(f, l)$. ImageNet, Imagenette, and Imagewoof are available freely for research purposes. Galaxy10 is available under the MIT license.

B DYNO with Surgical Fine-Tuning

Towards structurally heterogeneous learning rates, we additionally reimplemented Surgical Fine-Tuning (SFT) [Lee et al., 2023] as an orthogonal transfer learning tool. We specifically use their Auto-RGN method, which dynamically scales the learning rate of each layer based on the relative gradient norm. The experimental results, including those from Figure 1 and also the same experimental protocols but with SFT, are shown in Figure 2. All hyperparameters were kept the same, with SFT hyperparameters used from Lee et al. [2023].

With SFT, DYNO performs as well or better than static architectures across all tasks. However, DYNO+SFT has significantly less architectural change: virtually none during transfer to Imagewoof, only late changes for Imagenette, and immediate pruning but then subtle growth for Galaxy10. A potential explanation is that SFT reduces some signaled need for grow-and-prune by heterogeneously and dynamically tweaking learning rates. Further hyperparameter tuning towards synergy could yield improved results: implementing SFT on a neuronal level could be even more beneficial.

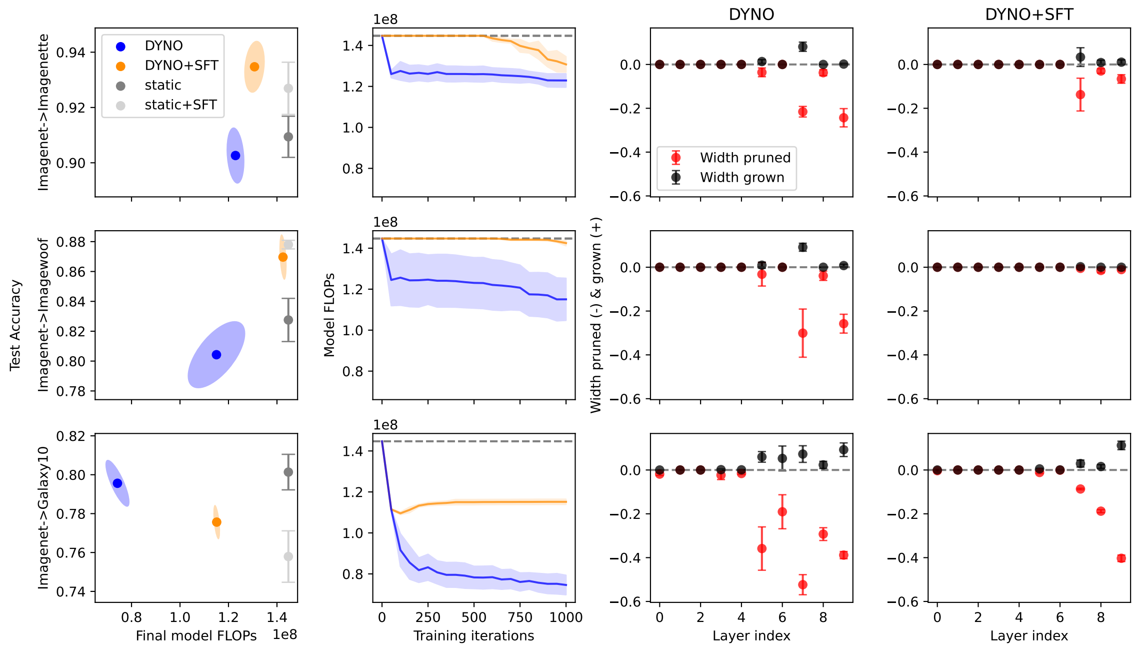

Figure 2: Experimental results for transfer to Imagenette (top), Imagewoof (middle), and Galaxy10 (bottom), including combinations of DYNO and SFT [Lee et al., 2023]. First column: final test accuracy against final model inference FLOPs. Second column: progression of inference cost over the course of training. Third and fourth columns: total neurons grown and pruned per layer over the course of DYNO (third column) and DYNO+SFT (fourth column), relative to the original layer width. All covariance ellipses, error clouds, and error bars show standard deviation across the 5 trials.

A notable potential confounder is that SFT only reduces the learning rate from the baseline value used, so all trials using SFT had smaller learning rates overall. This explains why performance was improved for Imagenette and Imagewoof, which are very close in distribution to Imagenet, but was worse for Galaxy10.