Towards Efficient Search for Customized Activation Functions With Gradient Descent

Lukas Strack[1] **Mahmoud Safari**[1] **Frank Hutter**[2,1]

[1]Department of Computer Science, University of Freiburg
[2]ELLIS Institute Tübingen

Abstract We leverage recent advancements in gradient-based search techniques for neural architectures to efficiently identify high-performing activation functions for a given application. We propose a fine-grained search cell that combines basic mathematical operations to model activation functions, allowing for the exploration of novel activations. Our approach enables the identification of specialized activations, leading to improved performance in every model we tried, from image classification to language models. Moreover, the identified activations exhibit strong transferability to larger models of the same type, as well as new datasets. Importantly, our automated process is orders of magnitude more efficient than previous approaches. It can easily be applied on top of arbitrary deep learning pipelines and thus offers a promising practical avenue for enhancing deep learning architectures.

1 Introduction and related work

Nonlinearities are an indispensable component of any deep neural network, and their design choice crucially affects the training dynamics and performance of neural networks.

The rectified linear unit (ReLU) is the most commonly used activation due to its simplicity and consistent performance across different tasks. However, it took several years of empirical research [14, 18, 25] before it was widely adopted by practitioners as an activation function in deep neural networks.

Despite desirable properties of ReLU, other alternatives have been introduced [21, 15, 8, 17, 12, 22], each with their own theoretical or empirical justification, to address potential issues associated with ReLU, such as the dying ReLU problem [30, 1]. These alternative activations lead to performance improvements in particular settings, although none is as widely adopted yet. As evidenced by previous research, manually designing an activation that suits a certain task is non-trivial and established choices (such as ReLU, SiLU and GELU) are made possibly at the cost of losing optimal performance.

Automated search methods have been previously employed to learn activation functions and have primarily followed two distinct approaches. One approach involves learning highly parameterized adaptable activations, concurrently with network training. [2] utilize a general piecewise linear unit, while [13] employ a weighted sum of polynomial basis elements. In contrast, [23] utilize the Padé approximant, which exhibits improved stability. [28] adopt a piecewise linear approximation but introduce inductive bias to simplify optimization. Another approach treats activation functions as hyperparameters optimized during a search phase, akin to neural architecture search (NAS). [27] define a search space comprising basic unary and binary operations and use reinforcement learning to guide an RNN controller in predicting activation components. Subsequent works, such as [3, 4, 19, 5] employ evolutionary strategies to explore activation function spaces. [6] introduce AQuaSurF, which efficiently searches the activation space using a regression algorithm, reducing computational cost compared to previous approaches. The black-box nature of these optimization methods makes them computationally demanding and impractical to apply to large spaces.

Our approach instead draws on recent developments in the rapidly growing field of Neural Architecture Search (NAS) with over a thousand papers in the last few years (see [32] for a recent survey). NAS has mostly been limited to architectural choices, such as network depth or width in macro search spaces, or a pre-defined set of operations in cell-based search spaces, in all of which the activations are fixed. Recently, gradient-based one-shot methods [20, 7, 9] have shown promise in efficiently optimizing architecture search spaces, reducing time costs by orders of magnitude compared to blackbox methods. Here, we adapt these NAS methods to mimic this success for searching activation functions by combining primitive mathematical operations. We summarize our contributions as follows[1]:

- We implement several key adjustments to modern gradient-based architecture search methods, tailoring them to search within the space of activations.
- Within image classification tasks with ResNet and ViT architectures, as well as language modelling with GPT, we demonstrate that using gradient-based one-shot search strategies we can discover from scratch specialized activations that improve a network's performance. Notably, our approach proves orders of magnitude more efficient compared to previous methods.
- Moreover, we investigate the transferability of the discovered activations to different models and datasets, and show that activation functions selected on a network/dataset, are among the top-performing activations on similar but larger models, as well as on new datasets.

2 Methodology

Following [27, 4, 5] the space of activations is defined as a combination of unary and binary operations, which form a scalar function f, as shown in Figure 1 (Right). The unary and binary functions are chosen from a set of primitive mathematical operations, listed in Figure 1 (Left). We also include several existing activation functions as unary operations to enrich the search space further as in [5].

In order to enable gradient-based optimization on this discrete space we continuously relax the space by assigning a weighted sum of all unary(binary) operations to the edge(vertex) of the graph as in DARTS [20]. These *activation parameters* are then optimized in a bi-level fashion. However vanilla DARTS is known to suffer from performance degradation at discretizaion [34]. In order to overcome this problem and also encourage more exploration in the search space we closely align with the distribution learning concept introduced in DrNAS [7]. However, given the slightly different nature of activation function spaces compared to those of neural architectures, this optimizer, at least in its original form, is not the best fit for discovering top performing activations.

We hypothesize that this is why this approach does not exist in the literature yet for activation function search. In order to make gradient-based optimization work for such spaces, we now introduce a series of techniques to robustify the approach.

Constraining unbounded operations. Naïvely applying gradient-based optimizers to activation search fails due to unbounded activations that lead to exploding gradients. To address this issue, we regularize the search space by constraining unbounded operations. That is, operation outputs y with magnitude beyond a threshold $|y| > \ell$ will be set to $y = \ell \operatorname{sign}(y)$. Here, we take $\ell = 10$.

Warmstarting the search. To robustify the search we introduce a short warm-starting phase during which the model weights are updated in the inner loop using the original activation, while the search cell is optimized in the outer loop. This ensures initializing the search with reasonable settings for both the network weights and the activation function parameters. After warm-starting the bi-level search continues, updating both model weights in the inner loop and activation parameters in the outer loop.

[1]To facilitate reproducibility, we make our code available here.

Unary		Binary
x	$\sinh(x)$	$x_1 + x_2$
$-x$	$\tanh(x)$	$x_1 - x_2$
x^2	$\text{arcsinh}(x)$	$x_1 x_2$
x^3	$\arctan(x)$	$\max(x_1, x_2)$
$\sqrt{x}$	$\text{erf}(x)$	$\min(x_1, x_2)$
e^x	$\min(0, x)$	$\sigma(x_1)\, x_2$
$\lvert x \rvert$	$\text{ReLU}(x)$	$\sigma(\gamma)x_1 + (1 - \sigma(\gamma))x_2$
γ	$\text{GELU}(x)$	$L(x_1, x_2)$
γx	$\text{SiLU}(x)$	$R(x_1, x_2)$
$x + \gamma$	$\text{ELU}(x)$	
$\sigma(x)$	$\text{LeakyReLU}(x)$	
$\log(1 + e^x)$		

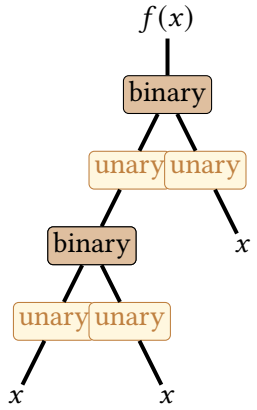

Figure 1: (Left) set of unary and binary operations. γ is a learnable parameter that is trained along with the activation parameters and becomes frozen after the search is completed. $\sigma(x)$ is the sigmoid function, and L, R are the left and right projection operations. (Right) activation cell: combination of unary and binary operations

Progressive shrinking. There are some key differences between architecture spaces and those of activation functions. In particular, unlike architecture spaces, operations in the space of activations are nearly parameter free, as these are basic mathematical functions. Furthermore, different unary and binary functions operate on different scales, making it challenging to rank their significance based on their coefficients.

Because of such inherent differences, it turns out that these methods do not perform well enough initially. Moreover the problem of performance drop at discretization, present in most NAS approaches, is more pronounced in the activation function space. To address these challenges we track activation parameters and drop unary and binary operations with lowest parameters at each epoch, following a logarithmic schedule[2]. This *progressive shrinking* of the search cell not only improves efficacy of the approach but further expedites the search process.

DrNAS with variance reduction sampling. To optimize the activation cell we closely follow DrNAS, where a Dirichlet distribution $Dir(\rho)$ is assigned to each edge/vertex of the search cell and the concentration parameters ρ are trained to minimize the expected validation loss. At each iteration, DrNAS draws activation parameters from its Dirichlet distribution. While DrNAS uses a single sample throughout the network, in our variant, to reduce the variance introduced by this sampling process, we draw independent samples for each activation cell within the network. Algorithm 1 outlines the pseudocode for our **GR**adient-based **A**ctivation **F**unction **S**earch (GRAFS) approach.

3 Experiments

We explore high-performing activation functions across ResNet, ViT and GPT architectures. All examined models employ a single type of activation throughout the network, which is globally replaced with the search cell in Figure 1 (Right) and optimized as per Section 2.

For each model, we repeat the search procedure with five different seeds, resulting in up to five distinct activation functions. The identified activations are evaluated on the networks/datasets they are searched on and subsequently also transferred to larger models of the same type and/or applied to new datasets. For the evaluation of each discovered activation, we train the models with it for five seeds on the train set, and report test set performance (mean ± the standard error of the mean).

In all Image classification experiments we utilized the implementation provided in the GitHub repository [33], but employed the TrivialAugment (TA) setup [24] as the augmentation method.

[2]See Algorithm DropOps and Appendix C for details.

	ResNet20			ResNet32		
act.func	CIFAR10	CIFAR100	SVHN	CIFAR10	CIFAR100	SVHN
F_{RN}^1	91.188 ± 0.123	**66.852** ± 0.167	95.904 ± 0.045	**92.494** ± 0.05	68.746 ± 0.11	96.384 ± 0.028
F_{RN}^2	91.216 ± 0.074	66.282 ± 0.191	**96.042** ± 0.06	92.35 ± 0.038	68.686 ± 0.14	96.353 ± 0.041
F_{RN}^3	91.44 ± 0.037	66.102 ± 0.139	95.953 ± 0.03	92.132 ± 0.044	68.232 ± 0.14	96.393 ± 0.031
F_{RN}^4	**91.446** ± 0.105	66.142 ± 0.127	95.973 ± 0.022	92.39 ± 0.12	68.552 ± 0.138	96.398 ± 0.044
F_{RN}^5	91.368 ± 0.064	66.272 ± 0.188	95.91 ± 0.053	92.448 ± 0.059	68.756 ± 0.144	96.392 ± 0.051
SiLU	91.44 ± 0.148	66.504 ± 0.126	95.982 ± 0.041	92.368 ± 0.017	68.566 ± 0.059	96.398 ± 0.028
GELU	91.136 ± 0.094	66.458 ± 0.109	95.895 ± 0.022	92.47 ± 0.065	68.476 ± 0.237	**96.418** ± 0.028
ELU	91.054 ± 0.064	66.464 ± 0.108	95.923 ± 0.055	92.204 ± 0.095	**68.784** ± 0.108	96.276 ± 0.044
LeakyReLU	91.086 ± 0.077	66.228 ± 0.11	95.887 ± 0.024	92.194 ± 0.061	68.478 ± 0.183	96.355 ± 0.033
ReLU	90.932 ± 0.11	66.314 ± 0.138	95.904 ± 0.042	92.12 ± 0.061	68.716 ± 0.127	96.327 ± 0.036

Table 1: Test performance of activations found on ResNet20 / CIFAR10. Evaluations are on ResNet20 and ResNet32 / CIFAR10, CIFAR100, SVHN.

ResNet. Residual networks (ResNets) were introduced to address the challenges of training deep networks [16]. This work focuses on ResNet20 and ResNet32 architectures. Table 1 compares our five activations discovered on ResNet20 / CIFAR10 (See Appendix F), with baselines. It also assesses their generalization on CIFAR100 and SVHN. Additionally, the table compares the generalization performance on ResNet32 across CIFAR10, CIFAR100, SVHN. Table 1 illustrates the effectiveness of our search method in identifying task-specific activations. On CIFAR10, one activation surpasses all baselines, and all five improve upon the default ReLU. Furthermore, the discovered activations demonstrate transferability to larger models and new datasets, outperforming baselines in most cases. The search overheads on different models and datasets range from 2.64 to 4.85 function evaluations (Table 11). These low ratios are partly due to the lower number of search epochs (See Appendix D.2), and the aggressive pruning of the search cell at the early stages (See Appendix C).

Vision Transformers. Vision Transformers [10] based on the self-attention mechanism [31] have become increasingly popular in the vision domain. In the original ViT model GELU has been the default activation function. Here we let the automated search discover the activation that is well-suited to the ViT architecture. To avoid computational burden, we conduct the search on the ViT-Ti [29] model which is a light version of ViT. The specific version of this model, as well as a larger variant used for evaluation in this study, is adapted from the implementation [33], which we denote as ViT-tiny and ViT-small, respectively (See D.1 for details of the architectural choices). Table 2 compares the five novel activations found in the search process on ViT-tiny/CIFAR10 (Appendix F) to baselines, on ViT-tiny/CIFAR10, CIFAR100, SVHN. Four out of five activations outperform existing baselines on all three datasets. This pattern further extends to the larger variant ViT-small. Table 12 shows small search overheads of 0.32 to 0.92 function evaluations in this case.

Generative pre-trained transformers. To diversify our experiments, we also evaluate our approach on language modeling tasks, specifically using the Generative Pre-trained Transformer (GPT). We focus on Andrej Karpathy's streamlined implementation[3] of GPT-2 [26] for simplicity. We optimize the activation within a down-scaled version of this architecture with 11M parameters which we denote as miniGPT. We employ the TinyStories [11] dataset for training. We repeat the search five times, warm-starting it with the default GELU activation. This results in five new activations (See Appendix F) all of which demonstrate lower test losses compared to GELU (Table 3). For three activations, highlighted in gray, these improvements also transfer to two larger variants which we refer to as tinyGPT and smallGPT, with 30M and 65M parameters respectively. The ratios of search time to evaluation time for all models are reported in Table 13 and range from 0.25 to 0.8.

[3]https://github.com/karpathy/nanoGPT

	ViT-tiny			ViT-small		
act.func	CIFAR10	CIFAR100	SVHN	CIFAR10	CIFAR100	SVHN
F^1_{ViT}	90.808 ± 0.107	68.452 ± 0.227	96.382 ± 0.024	93.074 ± 0.15	72.078 ± 0.223	96.968 ± 0.036
F^2_{ViT}	91.838 ± 0.136	69.562 ± 0.276	96.661 ± 0.028	93.658 ± 0.025	72.82 ± 0.087	97.146 ± 0.02
F^3_{ViT}	**92.132** ± 0.172	**70.408** ± 0.138	96.624 ± 0.068	**93.972** ± 0.069	**73.372** ± 0.158	**97.202** ± 0.032
F^4_{ViT}	91.71 ± 0.065	69.544 ± 0.172	96.573 ± 0.038	93.792 ± 0.089	72.798 ± 0.201	97.15 ± 0.007
F^5_{ViT}	91.914 ± 0.183	70.034 ± 0.146	**96.673** ± 0.028	93.802 ± 0.072	73.26 ± 0.055	97.155 ± 0.041
ELU	90.736 ± 0.048	67.826 ± 0.365	96.358 ± 0.017	92.174 ± 0.185	68.292 ± 0.149	96.788 ± 0.054
GELU	91.396 ± 0.210	68.326 ± 0.220	96.436 ± 0.052	93.238 ± 0.112	71.278 ± 0.255	97.059 ± 0.045
LeakyReLU	91.210 ± 0.095	68.000 ± 0.190	96.449 ± 0.026	92.930 ± 0.076	70.786 ± 0.271	97.021 ± 0.036
ReLU	91.180 ± 0.144	68.064 ± 0.102	96.479 ± 0.065	92.890 ± 0.037	70.734 ± 0.120	96.940 ± 0.037
SiLU	91.554 ± 0.097	68.706 ± 0.132	96.509 ± 0.044	93.254 ± 0.084	71.216 ± 0.203	97.004 ± 0.036

Table 2: Comparison of activations identified over ViT-tiny/CIFAR10 with baselines on ViT-tiny and ViT-small/CIFAR10, CIFAR100, SVHN. Highlighted activations surpass baselines on all tasks.

activ.func.	miniGPT	tinyGPT	smallGPT
F^1_{GPT}	**1.919 ± 0.002**	1.492 ± 0.002	1.324 ± 0.003
F^2_{GPT}	**1.919 ± 0.002**	**1.489 ± 0.002**	1.324 ± 0.003
F^3_{GPT}	1.933 ± 0.003	1.495 ± 0.002	**1.322 ± 0.003**
F^4_{GPT}	1.934 ± 0.002	1.501 ± 0.002	1.331 ± 0.003
F^5_{GPT}	1.932 ± 0.002	1.509 ± 0.002	1.351 ± 0.003
GELU	1.941 ± 0.002	1.499 ± 0.002	1.325 ± 0.003

Table 3: Comparison of activations identified over miniGPT / TinyStories with GELU on miniGPT, tinyGPT and smallGPT / TinyStories. Highlighted activations outperform GELU on all models.

4 Conclusions

We have adapted modern gradient-based architecture search techniques to explore the space of activation functions. Our proposed search strategy can identify activations tailored to specific deep learning models that surpass commonly-used alternatives and exhibit transferability to larger models of the same type, as well as new datasets. Most notably, our method requires only a few function evaluations, in contrast to thousands required by existing methods, making it highly efficient and convenient for practitioners.

This work aims to demonstrate the potential of gradient techniques in identifying top activations, and as the first such work is not intended to represent the optimal pipeline. While our approach may potentially already improve available strong models, we mostly see this work as opening the door for a host of possible follow-ups, such as improved search spaces and methods, searching for activations with robust performance across workloads, or strong scaling to larger networks. We hope that our work encourages further research and exploration in this direction.

5 Broader Impact Statement

This paper introduces a new line of work on gradient-based search for activation functions in deep learning. While the societal implications of deep learning are vast, we focus on the efficiency of our search method, which represents advancement in sustainability and democratization of this research area. In particular our search times are between 0.25 and 4.85 times the evaluation time (multiplied by 5 repetitions), in contrast to thousands of evaluations required by existing methods. This addresses the urgent need for sustainable computing practices and green machine learning, especially in the context of Automated Machine Learning.

Acknowledgements. This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828. The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG. Frank Hutter is a Hector Endowed Fellow at the ELLIS Institute Tübingen.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *ArXiv*, abs/1803.08375, 2018.

[2] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[3] Mina Basirat and Peter M Roth. The quest for the golden activation function. *arXiv preprint arXiv:1808.00783*, 2018.

[4] Garrett Bingham, William Macke, and Risto Miikkulainen. Evolutionary optimization of deep learning activation functions. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 289–296, 2020.

[5] Garrett Bingham and Risto Miikkulainen. Discovering parametric activation functions. *Neural Networks*, 148:48–65, 2022.

[6] Garrett Bingham and Risto Miikkulainen. Efficient activation function optimization through surrogate modeling. *arXiv preprint arXiv:2301.05785*, 2023.

[7] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2020.

[8] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[9] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1761–1770, 2019.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[11] Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

[12] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

[13] Mohit Goyal, Rajan Goyal, and Brejesh Lall. Learning activation functions: A new paradigm for understanding neural networks. *arXiv preprint arXiv:1906.09529*, 2019.

[14] Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[16] 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.

[17] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[18] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, 2009.

[19] Hanxiao Liu, Andy Brock, Karen Simonyan, and Quoc Le. Evolving normalization-activation layers. *Advances in Neural Information Processing Systems*, 33:13539–13550, 2020.

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018.

[21] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Icml*, 2013.

[22] Diganta Misra. Mish: A self regularized non-monotonic activation function. In *British Machine Vision Conference*, 2020.

[23] Alejandro Molina, Patrick Schramowski, and Kristian Kersting. Padé activation units: End-to-end learning of flexible activation functions in deep networks. In *International Conference on Learning Representations*, 2019.

[24] Samuel G Müller and Frank Hutter. Trivialaugment: Tuning-free yet state-of-the-art data augmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 774–782, 2021.

[25] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[26] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. In *OpenAI blog 1, no. 8*, 2019.

[27] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[28] Mohammadamin Tavakoli, Forest Agostinelli, and Pierre Baldi. Splash: Learnable activation functions for improving accuracy and adversarial robustness. *Neural Networks*, 140:1–12, 2021.

[29] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[30] Ludovic Trottier, Philippe Giguère, and Brahim Chaib-draa. Parametric exponential linear unit for deep convolutional neural networks. *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 207–214, 2016.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[32] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023.

[33] Kentaro Yoshioka. `https://github.com/kentaroy47/vision-transformers-cifar10`, 2024.

[34] A Zela, T Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and F Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

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

   (b) Did you describe the limitations of your work? N/A

   (c) Did you discuss any potential negative societal impacts of your work? N/A

   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? `https://2022.automl.cc/ethics-accessibility/` YES

2. If you ran experiments...

   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources)? YES

   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning)? YES

   (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? YES

   (d) Did you report the uncertainty of your results (e.g., the variance across random seeds or splits)? YES

   (e) Did you report the statistical significance of your results? NO

   (f) Did you use tabular or surrogate benchmarks for in-depth evaluations? N/A

   (g) Did you compare performance over time and describe how you selected the maximum duration? N/A

   (h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? YES

   (i) Did you run ablation studies to assess the impact of different components of your approach? YES

3. With respect to the code used to obtain your results...

   (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 versions), random seeds, an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? YES

   (b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? YES

   (c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? YES

   (d) Did you include the raw results of running your experiments with the given code, data, and instructions? NO

   (e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? NO

4. If you used existing assets (e.g., code, data, models)...

(a) Did you cite the creators of used assets? YES

(b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? N/A

(c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? N/A

5. If you created/released new assets (e.g., code, data, models)…

(a) Did you mention the license of the new assets (e.g., as part of your code submission)? NO

(b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? YES

6. 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

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? N/A

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? N/A

7. If you included theoretical results…

(a) Did you state the full set of assumptions of all theoretical results? N/A

(b) Did you include complete proofs of all theoretical results? N/A

A Gradient-based Activation Function Search

Algorithm 1 provides an overview of our proposed GRAFS method.

Algorithm 1: GRAFS

Input : Shrinking schedule of the search cell D_e; Original activation function $\bar{a}$; Set of activation cells $\mathcal{A}$ that replace the original activation and their respective activation parameters $\alpha = \{\alpha_a | \text{ for } a \text{ in } \mathcal{A}\}$; Total number of epochs E; Warm-starting epochs E_0

Warm-starting:;

for $e \leftarrow 1$ **to** E_0 **do**

  For all a in $\mathcal{A}$ sample $\alpha_a \sim Dir(\rho)$;

  Update distribution parameters ρ by descending $\nabla_\rho \mathcal{L}_{valid}(w, \mathcal{A}(\alpha))$;

  Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \bar{a})$;

end

Search:;

for $e \leftarrow E_0$ **to** E **do**

  DropOps(D_e) ▹ see Procedure DropOps;

  For all a in $\mathcal{A}$ sample $\alpha_a \sim \text{Dir}(\rho)$;

  Update distribution parameters ρ by descending $\nabla_\rho \mathcal{L}_{valid}(w, \mathcal{A}(\alpha))$;

  For all a in $\mathcal{A}$ sample $\alpha_a \sim \text{Dir}(\rho)$;

  Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \mathcal{A}(\alpha))$;

end

Procedure DropOps(D)

for $i \leftarrow 1$ **to** D **do**

  $O \leftarrow$ edge or vertex with most operations left;

  Drop operation in O with lowest activation param;

end

B Dirichlet Neural Architecture Search

For completeness and comparison with our proposed method, in this section we present the pseudocode for DrNAS for neural architecture search. A regularizer term $\lambda d(\rho, \hat{\rho})$ appears with coefficient λ which enforces the distribution parameters ρ to stay close to an anchor $\hat{\rho} = 1$.

Algorithm 2: DrNAS - Dirichlet Neural Architecture Search

Input : One-shot model with Initialized weights w; Dirichlet distribution parameters ρ; Anchor $\hat{\rho} = 1$, anchor regularizer parameter λ, and metric d

while *not converged* **do**

  1. Sample architecture parameters $\alpha \sim Dir(\rho)$;

  2. Update distribution parameters β by descending $\nabla_\rho \big(E_{\theta \sim Dir(\rho)}[\mathcal{L}_{valid}(w, \alpha)] + \lambda d(\rho, \hat{\rho})\big)$;

  3. Sample architecture parameters $\alpha \sim Dir(\rho)$;

  4. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

end

Return: Derive the final discretized architecture based on argmax of learned ρ

C Shrinking schedule

In Algorithm 1 the shrinking schedule D_e denotes the number of operations to be dropped at epoch e during the search phase. In this work we adopt a log schedule for D_e. Specifically, given the initial

(total) number of operations in the activation cell $D = 4 \times 23 + 2 \times 9 = 110$, $D - 6$ operations have to be dropped in order to reach a fully discretized architecture. $D - 6$ points are then distributed with a log spacing among the epochs, starting from epoch $e = S$, at which shrinking begins, and the final epoch $e = E$. These points are then binned into unit intervals, determining the number of operations to drop at each epoch (see Fig.2 for a visualization). In this work we always start shrinking at twice the warm-starting epoch $S = 2E_0$.

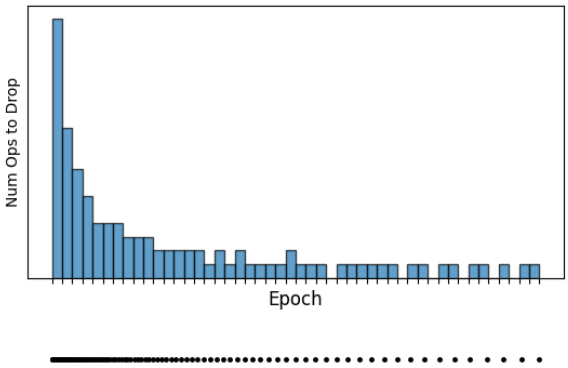

Figure 2: (Bottom) Log-scaled distribution of epochs at which operations are dropped. (Top) Histogram determines number of operations to drop per epoch.

D Experimental settings

In this section we collect the settings of the experiments in the paper. All the search and evaluation experiments have been done on a single GeForce RTX 2080 Ti GPU.

D.1 Architectural parameters.

Given the many versions of the ViT and GPT architectures, to avoid ambiguity, we present here the architectural parameters of the models we have used in this work.

	ViT-tiny	ViT-small
embed_dim	512	512
depth	4	6
num_heads	6	8
mlp_dim	256	512
patch_size	4	4
img_size	32	32

Table 4: Architectural parameters for ViT-tiny and ViT-small.

	miniGPT	tinyGPT	smallGPT
n_layers	3	6	9
n_heads	3	6	9
n_embd	192	384	576

Table 5: Architectural parameters for miniGPT, tinyGPT and smallGPT.

D.2 Search settings for image classification tasks

Search - ResNet20	
dataset	CIFAR10
augmentation	TrivialAugment
search_epochs	50
batch_size	32
gradient_accumulation_steps	16
optimizer	Adam(lr=0.001, betas=(0.9, 0.999))
scheduler	CosineAnnealing(eta_min=0.0)
train_val_split	0.75
optimizer_arch	Adam(lr=0.1, betas=(0.9, 0.999))
warmstart_epoch	1
start_shrinking_epoch	2

Table 6: Hyperparameter settings for the bi-level search process on ResNet20.

Search - ViT-tiny	
dataset	CIFAR10
augmentation	TrivialAugment
search_epochs	50
batch_size	128
gradient_accumulation_steps	4
optimizer	Adam(lr=0.001, betas=(0.9, 0.999))
scheduler	CosineAnnealing(eta_min=0.0)
train_val_split	0.75
optimizer_arch	Adam(lr=0.001, betas=(0.9, 0.999))
warmstart_epoch	1
start_shrinking_epoch	2

Table 7: Hyperparameter settings for the bi-level search process on ViT-tiny.

D.3 Evaluation settings for image classification tasks

Evaluation - Image classification	
dataset	CIFAR10, CIFAR100, SVHN
augmentation	TrivialAugment
n_epochs	500
batch_size	512
optimizer	Adam(lr=0.001, betas=(0.9, 0.999))
scheduler	CosineAnnealing(eta_min=0.0)

Table 8: Hyperparameter settings for the evaluation process on ResNet20, ResNet32, ViT-tiny, ViT-small.

D.4 Search settings for language modelling tasks

Search - Language modelling	
dataset	TinyStories
eval_interval	100
max_iters	1000
batch_size	4
gradient_accumulation_steps	40
train_val_split	0.75
optimizer_arch	Adam(lr=1e-3, betas=(0.9, 0.999))
scheduler_arch	CosineAnnealingWarmRestarts(T_0=125, T_mult=1, eta_min=1e-4)
warmstart_iterations	100
start_shrinking_iteration	200

Table 9: Hyperparameter settings for the bi-level search process on miniGPT.

D.5 Evaluation settings for language modelling tasks

Evaluation - Language modelling	
dataset	TinyStories
compile	False
max_iters	10000
batch_size	16
gradient_accumulation_steps	40
optimizer	AdamW(lr=6e-4, weight_decay=1e-1, betas=(0.9, 0.99))
scheduler	CosineAnnealing(lr=6e-4, min_lr=1e-4)

Table 10: Hyperparameter settings for the evaluation process on miniGPT, tinyGPT and smallGPT.

E Search overheads

We collect here the search time to evaluation time ratios for ResNet, ViT and GPT experiments. The search time is the average search time for the five searches performed with different seeds, and

the evaluation time is the average training time over five seeds with the default activation function, i.e. ReLU for ResNet and GELU for ViT anf GPT.

	CIFAR10	CIFAR100	SVHN
ResNet20	4.55	4.85	2.82
ResNet32	4.65	4.57	2.64

Table 11: Search time to evaluation time ratios. Search is always on ResNet20 / CIFAR10.

	CIFAR10	CIFAR100	SVHN
ViT-tiny	0.92	0.79	0.54
ViT-small	0.54	0.57	0.32

Table 12: Search time to evaluation time ratios. Search is always on ViT-tiny / CIFAR10.

	miniGPT	tinyGPT	smallGPT
TinyStories	0.80	0.54	0.25

Table 13: Search time to evaluation time ratios. Search is always on miniGPT / TinyStories.

F Discovered activation functions

In this section we provide the explicit formulas and plots for all the 15 activation functions discovered on ResNet20 / CIFAR10, ViT-tiny / CIFAR10 and miniGPT / TinyStories.

$$
\begin{aligned}
F^1_{\text{RN}}(x) &= 0.5775\,\text{ReLU}(x) + 0.4225\,\text{SiLU}(x) \\
F^2_{\text{RN}}(x) &= 0.5644\,\text{ELU}(0.1673\sqrt{\text{ReLU}(x)} + 0.8327\,\text{SiLU}(x)) + 0.4356\,\text{LeakyReLU}(x) \\
F^3_{\text{RN}}(x) &= 0.2520\,\text{arcsinh}(\text{ReLU}(x) + 0.7480\,\text{SiLU}(x) \\
F^4_{\text{RN}}(x) &= 0.5318\,\text{ELU}(0.2796\sqrt{\text{ReLU}(x)} + 0.7204\,\text{ReLU}(x)) + 0.4682\,\text{SiLU}(x) \\
F^5_{\text{RN}}(x) &= \max(\text{ELU}(0.6127\,\text{ReLU}(x) + 0.3873\,\text{SiLU}(x)), \text{SiLU}(x))
\end{aligned}
\tag{1}
$$

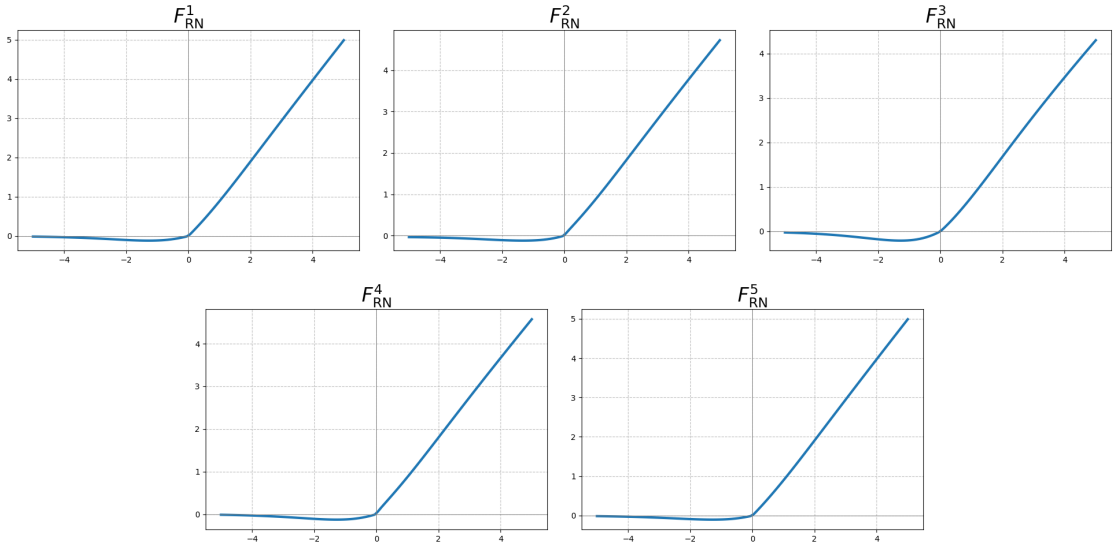

Figure 3: Plots of activation functions in Eq.1, found on ResNet20 / CIFAR10.

$$
\begin{aligned}
F^1_{\mathrm{ViT}}(x) &= 0.6991\,\mathrm{GELU}(\mathrm{GELU}(x)^2) + 0.3009\,\mathrm{GELU}(x) \\
F^2_{\mathrm{ViT}}(x) &= 0.7283\,\mathrm{GELU}(\mathrm{SiLU}(x)\mathrm{GELU}(x)) + 0.2717\,x^2 \\
F^3_{\mathrm{ViT}}(x) &= 0.3826\,\mathrm{GELU}(x^2) + 0.6174\,\mathrm{SiLU}(x) \\
F^4_{\mathrm{ViT}}(x) &= 0.7388\,\mathrm{GELU}(\mathrm{SiLU}(x)\mathrm{GELU}(x)) + 0.2612\,x^2 \\
F^5_{\mathrm{ViT}}(x) &= 0.6955\,\mathrm{SiLU}(0.3398\,x^2 + 0.6602\,\mathrm{SiLU}(x)) + 0.3045\,\mathrm{GELU}(x)
\end{aligned}
\tag{2}
$$

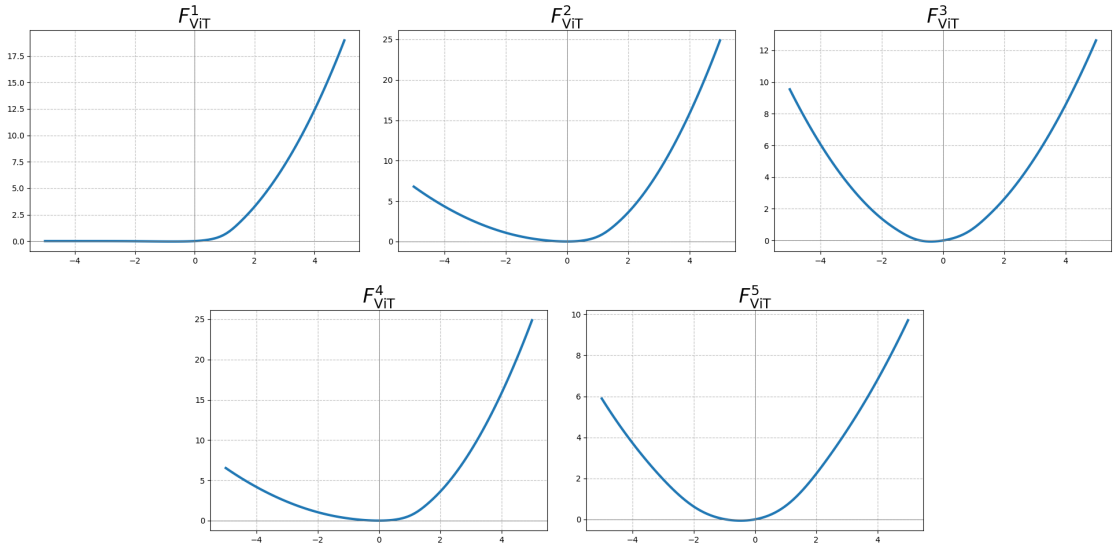

Figure 4: Plots of activation functions in Eq.2, found on ViT-Tiny / CIFAR10.

$$
\begin{aligned}
F^1_{\text{GPT}} &= \min(x^2, \text{ReLU}(x))^2 \, \text{LeakyReLU}(x) \\
F^2_{\text{GPT}} &= \text{ReLU}(x)^3 \\
F^3_{\text{GPT}} &= 0.5004 \, \text{ReLU}(x)^2 + 0.4996 \, \text{ReLU}(x) \\
F^4_{\text{GPT}} &= 0.4342 \, \text{GELU}(x^2 \sinh(x)) + 0.5658 \, \text{LeakyReLU}(x) \\
F^5_{\text{GPT}} &= \min(x^2, \sinh(x))^2 \, \text{LeakyReLU}(x)
\end{aligned}
\tag{3}
$$

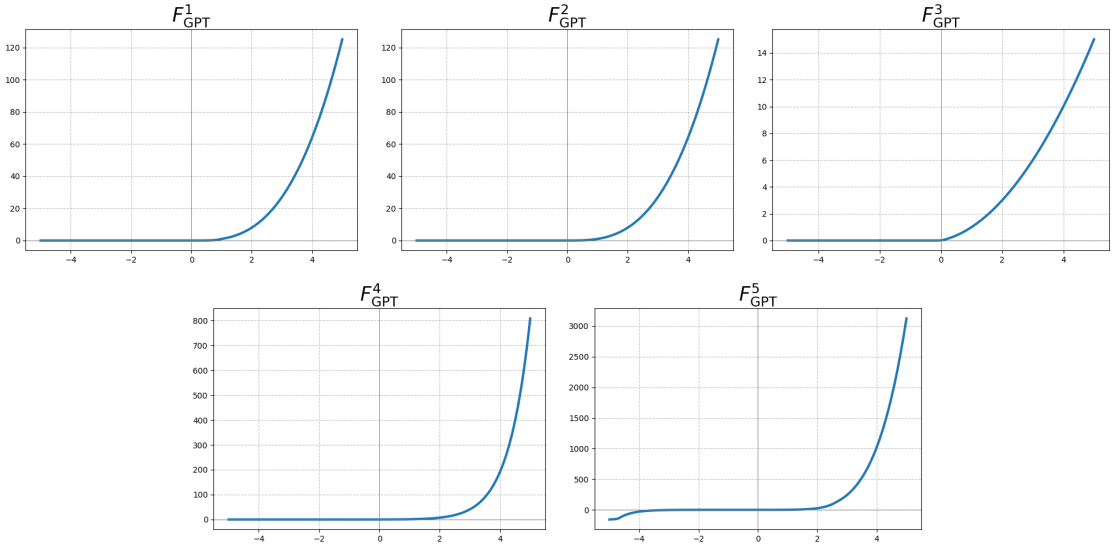

Figure 5: Plots of activation functions in Eq.3 found on miniGPT / TinyStories.