WEIGHT-ENTANGLEMENT MEETS GRADIENT-BASED NEURAL ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Weight sharing is a fundamental concept in neural architecture search (NAS), enabling gradient-based methods to explore cell-based architecture spaces significantly faster than traditional blackbox approaches. In parallel, weight *entanglement* has emerged as a technique for intricate parameter sharing among architectures within macro-level search spaces. Since weight-entanglement poses compatibility challenges for gradient-based NAS methods, these two paradigms have largely developed independently in parallel sub-communities. This paper aims to bridge the gap between these sub-communities by proposing a novel scheme to adapt gradient-based methods for weight-entangled spaces. This enables us to conduct an in-depth comparative assessment and analysis of the performance of gradient-based NAS in weight-entangled search spaces. Our findings reveal that this integration of weight-entanglement and gradient-based NAS brings forth the various benefits of gradient-based methods (enhanced performance, improved supernet training properties and superior any-time performance), while preserving the memory efficiency of weight-entangled spaces. The code for our work is openly accessible here.

1 INTRODUCTION

The concept of weight-sharing in Neural Architecture Search (NAS) arose from the need to improve the efficiency of conventional blackbox NAS algorithms, which demand significant computational resources to evaluate individual architectures. Here, weight-sharing (WS) refers to the paradigm by which we represent the search space with a single large *supernet*, also known as the *one-shot* model, that subsumes all the candidate architectures in that space. Every edge of this supernet holds all the possible operations that can be assigned to that edge. Importantly, architectures that share a particular operation also share its corresponding operation weights, allowing simultaneous training of an exponential number of subnetworks, unlike the sequential approach of blackbox NAS.

Gradient-based NAS algorithms (or *optimizers*), such as DARTS (Liu et al., 2019), GDAS (Dong and Yang, 2019) and DrNAS (Chen et al., 2021b), assign an *architectural parameter* to every choice of operation on a given edge of the supernet. The output feature maps of these edges are thus an aggregation of the outputs of the individual operations on that edge, weighted by their architectural parameters. These architectural parameters are learned using gradient updates by differentiating through the validation loss. Supernet weights and architecture parameters are therefore trained simultaneously in a bi-level fashion. Once this training phase is complete, the final architecture can be identified quickly, e.g., by selecting operations with the highest architectural weights on each edge as depicted in Figure 1(b). However, more sophisticated methods have also been explored (Wang et al., 2021) for this selection.

While gradient-based NAS methods have primarily been studied for cell-based NAS search spaces, a different class of search spaces focuses on macro-level structures (parameterizing kernel size, number of channels, etc.) for which all architectures in the space are *subnetworks* of the architecture with the largest architectural choices, which is identical to the supernet in this case. These search spaces share weights via the more intricate form of *weight-entanglement* (WE) between similar operations on the same edge; e.g., the nine weights of a 3×3 convolution are a subset of the 25 weights of a 5×5 convolution. This paradigm reduces the memory requirements of the supernet to the size of the largest architecture in the search space.

In order to efficiently search over such weight-entanglement spaces, *two-stage* methods have been introduced that first pre-train the supernet, and then perform blackbox search on it to obtain the final architecture. OFA (Cai et al., 2020), SPOS (Guo et al., 2020), AutoFormer (Chen et al., 2021a) and HAT (Wang et al., 2020) are prominent examples of methods that fall into this category. Note that these methods do not employ additional architectural parameters for supernet training or search. They typically train the supernet by randomly sampling subnetworks and training them as depicted in Figure 1(a). The post-hoc blackbox search relies on using the performance of subnetworks sampled from the trained supernet as a proxy for true performance on the unseen test set. To contrast with this two-stage approach, we refer to traditional gradient-based NAS approaches as *single stage* methods.

While to date, weight-entangled spaces have only been explored with two-stage methods, and cell-based spaces have only been optimized with single-stage approaches, in this paper we bridge the gap between these parallel sub-communities. We do so by addressing the challenges associated with integrating off-the-shelf single-stage NAS methods with weight-entangled search spaces.

After a discussion of related work (Section 2), we make the following main contributions:

- We propose a generalized scheme to apply single-stage methods to weight-entangled spaces, while maintaining search efficiency and efficacy at larger scales (Section 3, with visualizations in Figure 1(c) and Figure 2). We refer to this method as *TangleNAS*.
- We propose a fair playground for the comparative evaluation of single and two-stage methods (Section 4.1) and study the effect of weight-entanglement in conventional cell-based search spaces (i.e., NASBench201 and the DARTS search space) (Section 4.2).
- We evaluate our proposed generalized scheme for single-stage methods across a diverse set of weight-entangled macro search spaces and tasks, from image classification (Section 4.3.1, Section 4.3.2) to language modeling (Section 4.3.3).
- We conduct a comprehensive evaluation of the properties of single and two-stage approaches including any-time-performance, memory consumption, robustness to training fraction and effect of fine-tuning (Section 5), demonstrating that our generalized gradient-based NAS method achieves the best of single and two-stage methods: the enhanced performance, improved supernet fine-tuning properties, superior any-time performance of single-stage methods, and the low memory consumption of two-stage methods .

To facilitate reproducibility, our code is openly accessible here.

2 Related Work

Weight-sharing was first introduced in ENAS (Pham et al., 2018), which reduced the computational cost of NAS by 1000$\times$ compared to previous methods. However, since this method used reinforcement learning its performance was quite brittle. Bender et al. (2018) simplified the technique, showing that searching for good architectures is possible by training the supernet directly with stochastic gradient descent. This was followed by DARTS (Liu et al., 2019), which set the cornerstone for efficient and effective gradient-based, *single-stage*, NAS approaches.

DARTS, however, had prominent *failure modes*, such as its *discretization gap* and *convergence towards parameter-free operations*, as outlined in Robust-DARTS (Zela et al., 2020). Numerous gradient-based one-shot optimization techniques were developed since then (Cai et al., 2019; Nayman et al., 2019; Xu et al., 2019; Dong and Yang, 2019; Hu et al., 2020; Li et al., 2021; Chen et al., 2021b; Zhang et al., 2021). Among these, we highlight DrNAS (Chen et al., 2021b), which we will use in our experiments as a representative of gradient-based NAS methods. DrNAS treats one-shot search as a distribution learning problem, where the parameters of a Dirichlet distribution over architectural parameters are learned to identify promising regions of the search space.

Despite the remarkable performance of single-stage methods, they are not directly applicable to some real-world architectural domains, such as transformers, because of the macro-level structure of these search spaces. DASH (Shen et al., 2022) employs a DARTS-like methodology to optimize CNN topologies (i.e. kernel size, dilation) for a diverse set of tasks, reducing computational complexity by appropriately padding and summing kernels with different sizes and dilations. FBNet-v2 (Wan et al., 2020) makes an attempt along these lines for CNN topologies, but its methodology is not easily

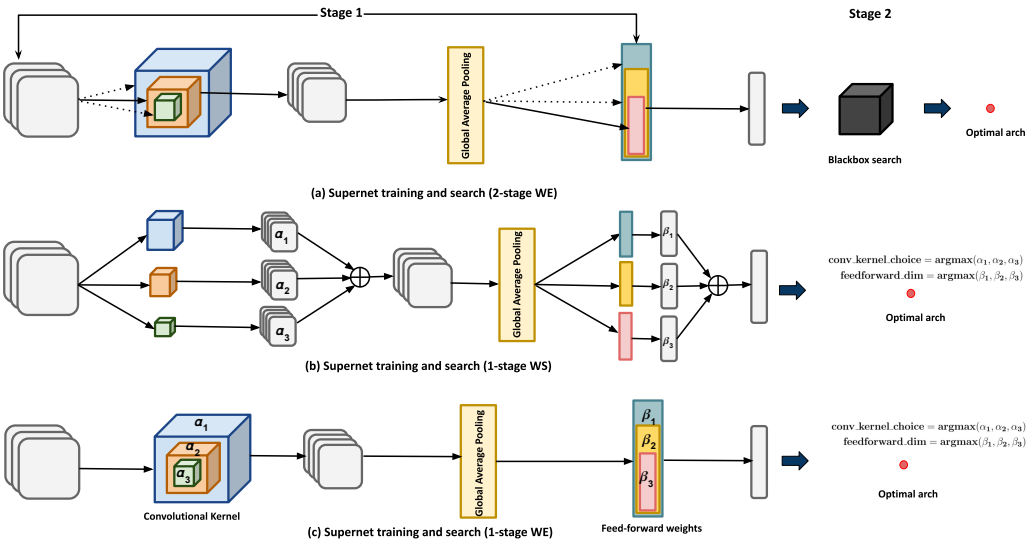


Figure 1: (a) **Two-Stage NAS with WE** (Algorithm 3): dotted paths show operation choices not sampled at the given step (b) **Single-Stage NAS with WS** (Algorithm 4): every operation choice is evaluated independently and contributes to the output feature map with corresponding architecture parameters (c) **Single-Stage NAS with WE** (Algorithm 1): operation choices superimposed with corresponding arhictecture parameters.

extendable to search spaces like transformers with multiple interacting modalities, such as embedding dimension, number of heads, expansion ratio, and depth.

Weight-entanglement, on the other hand, provides a more effective way of weight-sharing exclusive to macro-level architectural spaces. The concept of weight-entanglement was developed in slimmable networks (Yu et al., 2018; Yu and Huang, 2019), OFA (Cai et al., 2020) and BigNAS (Yu et al., 2020) in the context of convolutional networks (see also AtomNAS (Mei et al., 2019)) and later spelled out in AutoFormer (Chen et al., 2021a), where it was applied to the transformer architecture.

Single-path-one-shot (SPOS) methods (Guo et al., 2020) have shown a lot of promise in searching weight-entangled spaces. SPOS trains a supernet by uniformly sampling single paths (one at a time to limit memory consumption) and then training the weights along that path. The supernet training is followed by a black-box search that uses the performance of the models sampled from the trained supernet as a *proxy*.

OFA used a similar idea to optimize different dimensions of CNN architectures, such as its depth, width, kernel size and resolution. Additionally, it enforced training of larger to smaller sub-networks sequentially to prevent interference between sub-networks. Subsequently, AutoFormer adopted the SPOS method to optimize a weight-entangled space of transformer architectures.

In this work, we demonstrate how single-stage methods can be applied to macro-level search spaces with weight-entanglement, where one can benefit from the time efficiency and remarkable effectiveness of modern differentiable NAS optimizers, while concurrently maintaining memory efficiency of the weight-entangled space. While we chose to adopt DrNAS as the primary approach for our exploration of the weight-entangled space in this study, it is worth highlighting that our methodology is generally applicable to other gradient-based NAS methods.

3 Methodology: Single-Stage NAS with Weight-Superposition

Our primary goal in this work is to effectively apply single-stage NAS to search spaces with macro-level architectural choices. To reduce the memory consumption and preserve computational efficiency, we propose two major modifications to single-stage methods. In our framework, the operators on any given edge inherit the weights of the largest operator on that edge. This reduces the size of the supernet to the size of the largest individual architecture in the search space.

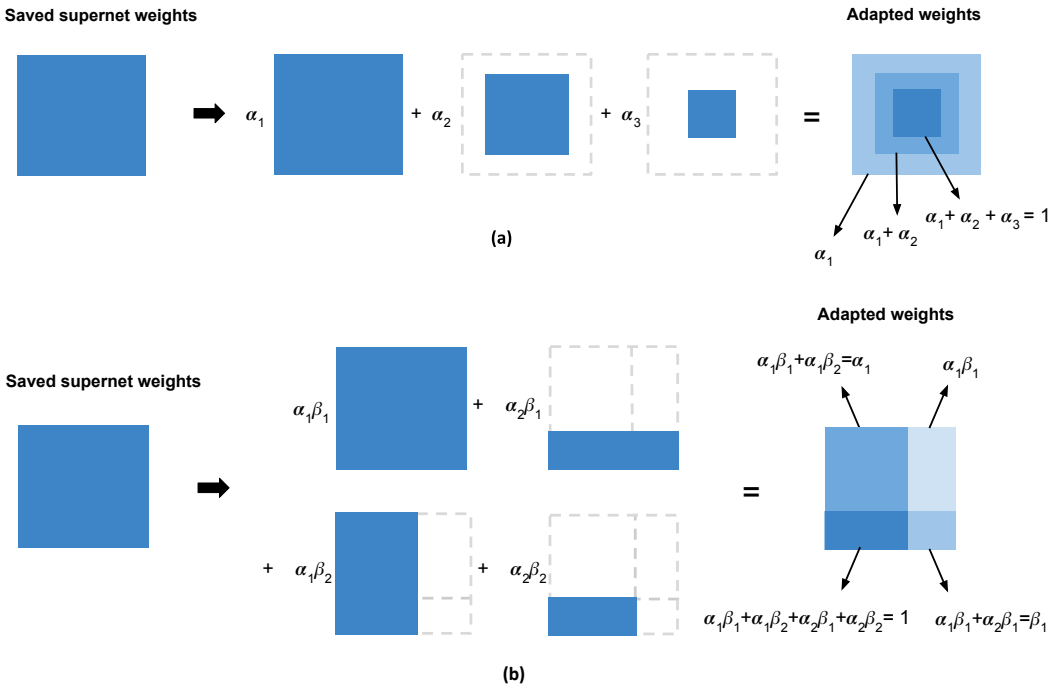


Figure 2: Supernet weight matrix (LHS) adapted to gradient-based methods (RHS). All operation choices are zero-padded to match the dimension of the largest operation and superimposed with given architecture parameter. (a) Weight superposition for a single dimension choice (e.g. kernel size) with architecture parameter α_i (b) Weight superposition for combination of multiple dimensions (e.g. embedding dimension and expansion ratio) with architecture parameters α_i, β_j. White areas inside the dashed gray boundaries indicate zero-padding and color values reflect scaling.

Then, instead of weighting the outputs of operations on a given edge by architectural parameters, as single-stage NAS methods do (see Figure 1(b)), we simply take each operation choice, zero-pad that to match the dimensions of the largest operation and sum these terms, each weighted by the corresponding architectural parameter. Figure 2(a) provides an overview of the idea for a single architectural choice such as the kernel size. This is equivalent to taking the largest operation and re-scaling the weights of each sub-operation by the sum of architectural parameters of operations it is embedded in (see right-most weight matrix in Figure 2(a).

This process results in a *superposition* of operation weights whose structural property is similar to the largest operation. Consequently, a *single* forward pass suffices to capture the effect of all operational choices, thus making our approach computationally efficient. Furthermore, to accommodate for operations depending on two or more architectural dimension choices we introduce the combi-superposition outlined in Figure 2(b) and Algorithm 2. This allows us to apply any arbitrary gradient-based NAS method, such as DARTS, GDAS or DrNAS, to macro-level search spaces that leverage weight-entanglement, without additional memory consumption and computational cost during forward propagation. We refer to our memory and compute optimal single-stage architecture search method as *TangleNAS* and provide an overview of the ap-

Algorithm 1 TangleNAS

Input: $M \leftarrow$ number of cells, $N \leftarrow$ number of operations
$\mathcal{O} \leftarrow [o_1, o_2, o_3, ...o_N]$
$\mathcal{W}_{\max} \leftarrow \cup_{i-1}^{N} w_i$
$\mathcal{A} \leftarrow [\alpha_1, \alpha_2, \alpha_3, ...\alpha_N]$
$\gamma =$ learning rate of $\mathcal{A}$, $\eta =$ learning rate of $\mathcal{W}_{\max}$
f is a function or distribution s.t. $\sum_{i=1}^{N} f(\alpha_i) = 1$
$Cell_j \leftarrow DAG(\mathcal{O}_j, \mathcal{W}_{max\,j})$ /* defined for j=1...M */
$Supernet \leftarrow \cup_{j}^{M} Cell_j \cup \mathcal{A}$
/* example of forward propagation on the cell */
for $j \leftarrow 1$ to M **do**
  /* PAD weight to output dimension of $\mathcal{W}_{\max}$ before summation */
  /* Generalized Weighing Scheme */
  $\overline{o_j(x, \mathcal{W}_{\max})} = o_{j,i}(x, \sum_{i=1}^{N} f(\alpha_i)\, \mathcal{W}_{\max}[:i])$
end for
/* weights and architecture update */
$\mathcal{A} = \mathcal{A} - \gamma \nabla_{\mathcal{A}} \mathcal{L}_{val}(\mathcal{W}_{\max}{}^{*}, \mathcal{A})$
$W_{\max} = \mathcal{W}_{\max} - \eta \nabla_{\mathcal{W}_{\max}} \mathcal{L}_{train}(\mathcal{W}_{\max}, \mathcal{A})$
/* Architecture Selection */
$selected\_arch \leftarrow \arg\max(\mathcal{A})$

Search Type	Optimizer	Supernet type	Test acc	Search Time(hrs)
Single-Stage	DrNAS	WS	91.190 ± 0.049	6.277
	TangleNAS	WE	$\mathbf{91.300 \pm 0.023}$	6.222
Two-Stage	SPOS+RS	WE	90.680 ± 0.253	15.611
	SPOS+ES	WE	90.317 ± 0.223	13.244
Optimum	-	-	91.630	-

Table 1: Evaluation on the toy cell-based search space on the Fashion-MNIST dataset

Search Type	Optimizer	Supernet type	Test acc	Search Time(hrs)
Single-Stage	DrNAS	WS	10 ± 0.00	12.361
	TangleNAS	WE	$\mathbf{82.495 \pm .0.461}$	8.55
Two-Stage	SPOS+RS	WE	81.253 ± 0.6717	21.676
	SPOS+ES	WE	81.890 ± 0.800	26.359
Optimum	-	-	84.410	-

Table 2: Evaluation on the toy conv-macro search space on the CIFAR10 dataset. Note that DrNAS with weight sharing converges to a degenerate architecture here.

proach in Algorithm 1. The operation f in Algorithm 1 determines the differentiable optimizer used in the method. A $softmax$ function corresponds to DARTS, while sampling from the Dirichlet distribution corresponds to DrNAS. In all experiments of the upcoming sections we use DrNAS as a basic gradient-based NAS method. We therefore simply refer to this combination as TangleNAS throughout the paper.

4 Experiments

We evaluate TangleNAS across a broad range of search spaces, ranging from cell-based spaces, which serve as the foundation for single-stage methods, to weight-entangled convolutional and transformer spaces, which are central to two-stage methods. We initiate our studies by exploring three simple toy search spaces, which include a collection of cell-based and weight-entangled spaces. Later, we scale our experiments to larger analogs of these spaces. In all our experiments, we use *WE* to refer to the supernet type with entangled weights between operation choices and *WS* to refer to standard weight-sharing proposed in cell-based spaces. For details on our experimental setup please refer to Appendix E. Furthermore, in all our experiments the focus is on *unconstrained search*, i.e., a scenario where the user is interested in obtaining the architecture with the best performance on their metric of choice. The two-stage baselines we mainly compare against are SPOS (Guo et al., 2020) with Random Search (*SPOS+RS*) and SPOS with Evolutionary Search (*SPOS+ES*). For MobileNetV3 (Section 4.3.2) and ViT (Section 4.3.1) we use the original training scheme from OFA (Cai et al., 2020) and Autoformer (Chen et al., 2021a), respectively. Both of these works use SPOS (Guo et al., 2020) as their foundation.

4.1 Toy search spaces

We begin the evaluation of TangleNAS on two relatively compact *toy* search spaces that we designed as a contribution to the community to allow faster iterations of algorithm development:

- **Toy cell space**: a small version of the DARTS space; architectures are evaluated on the Fashion-MNIST dataset.
- **Toy conv-macro space**: a small space inspired by MobileNet, including kernel sizes and the number of channels in each convolution layer; architectures are evaluated on CIFAR-10.

We describe these spaces in Appendix C, including links to code for these open source toy benchmarks. The results of these experiments are summarized in Tables 1 and 2. Across both of these search spaces, TangleNAS outperforms its *two-stage* counterparts. Also, DrNAS without weight entanglement performs very poorly on the macro level search space.

4.2 Cell-based search spaces

We now start our comparative analysis of single and two-stage approaches by applying them to cell-based spaces that serve as focal points in the single-stage NAS literature to evaluate TangleNAS against DrNAS and SPOS. Here, we use the popular NAS-Bench-201 (NB201) (Dong and Yang, 2020) and DARTS (Liu et al., 2019) search spaces. We refer the reader to Appendix C for details about these spaces and Appendix E.4 for the experimental setup.

Our contribution on these spaces is two-fold. First, we study the effects of weight-entanglement on cell-based spaces in conjunction with single-stage methods. To this end, we entangle the weights of similar operations with different kernel sizes on both search spaces. For NB201, the weights of the 1x1 and 3x3 convolutions are entangled, and in the DARTS search space, the weights of dilated and separable convolutions with kernel sizes 3x3 and 5x5 are entangled. Second, we study SPOS on the NB201 and DARTS search spaces. To the best of our knowledge, we are the first to study a two-stage method like SPOS in such cell search spaces.

Tables 3 and 4 show the results. For both search spaces, TangleNAS yields the best results, outperforming the single-stage baseline DrNAS with WS, as well as both SPOS variants. TangleNAS also significantly lowers the memory requirements and runtime compared to its weight-sharing counterpart. We note that overall, the SPOS methods are ineffective in these cell search spaces.

Search Type	Optimizer	Supernet	CIFAR10	CIFAR100	ImageNet16-120	Search Time (hrs)
Single-Stage	DrNAS	WS	$\mathbf{94.36 \pm 0.00}$	72.245 ± 0.732	$\mathbf{46.37 \pm 0.00}$	20.9166
	TangleNAS	WE	$\mathbf{94.36 \pm 0.00}$	$\mathbf{73.51 \pm 0.000}$	$\mathbf{46.37 \pm 0.00}$	20.3611
Two-Stage	SPOS+RS	WE	89.107 ± 0.884	56.865 ± 2.597	31.665 ± 1.1460	29.51388
	SPOS+ES	WE	87.133 ± 2.605	56.463 ± 2.342	29.785 ± 3.0149	26.74721

Table 3: Comparison of test-accuracies of single and two-stage methods on NB201 search space

Search Type	Optimizer	Supernet	CIFAR10	ImageNet	Search Time(hrs)
Single-Stage	DrNAS	WS	2.625 ± 0.075	26.29	29.4444
	TangleNAS	WE	$\mathbf{2.556 \pm 0.034}$	**25.691**	25.472
Two-Stage	SPOS+RS	WE	2.965 ± 0.072	27.114	18.7444
	SPOS+ES	WE	3.200 ± 0.065	27.320	14.794

Table 4: Comparison of test errors of single and two-stage methods on the DARTS search space

4.3 Macro Search Spaces

Given the promising results of TangleNAS on the toy and cell-based spaces, we now extend our evaluation to the home base of two-stage methods. We study TangleNAS on a vision transformer space (AutoFormer-T) and a convolutional space (MobileNetV3), which have been proposed and explored using two-stage methods by Chen et al. (2021a) and Cai et al. (2020), respectively, as well as a language model transformer search space built around GPT-2 Radford et al. (2019).

4.3.1 AutoFormer

Search Type	Optimizer	CIFAR10			CIFAR100		
		Inherit	Fine-tune	Retrain	Inherit	Fine-tune	Retrain
Single-Stage	TangleNAS	93.57	$\mathbf{97.702 \pm 0.017}$	$\mathbf{97.872 \pm 0.054}$	76.59	$\mathbf{82.615 \pm 0.064}$	$\mathbf{82.668 \pm 0.161}$
Two-Stage	SPOS+RS	**94.29**	97.605 ± 0.038	97.767 ± 0.024	**78.210**	82.407 ± 0.026	82.210 ± 0.14242
	SPOS+ES	94.10	97.632 ± 0.047	97.6425 ± 0.023	77.97	82.517 ± 0.140	82.5175 ± 0.114

Table 5: Evaluation on the AutoFormer-T space for CIFAR10 and CIFAR100.

We evaluate TangleNAS on the AutoFormer-T space introduced by Chen et al. (2021a), which is based on vision transformers. The search space consists of the choices of embedding dimensions and number of layers, and for each layer, its MLP expansion ratio and the number of heads it uses to compute attention. More details can be found in Table 17 in the appendix. The embedding dimension choice is held constant throughout the network while the number of heads and the MLP expansion ratio varies for every layer. This amounts to a total search space size of about 10^{13} architectures. We train our supernet using the same training hyperparameters as used in AutoFormer.

NAS Method	ImageNet	Datasets				
		CIFAR10	CIFAR100	Flowers	Pets	Cars
SPOS+ES	75.474	98.019	86.369	**98.066**	91.558	91.935
TangleNAS	**78.842**	**98.249**	**88.290**	**98.066**	**92.347**	**92.396**

Table 6: Evaluation on the AutoFormer-T space on downstream tasks. ImageNet accuracies are obtained through inheritance, whereas accuracies for the other datasets are achieved through fine-tuning the imagenet-pretrained model.

Table 5 and Table 6 evaluates TangleNAS against AutoFormer on the AutoFormer-T space. Interestingly, we observe that although AutoFormer sometimes outperforms TangleNAS upon inheritance from the supernet, the TangleNAS architectures are always better upon fine-tuning and much better upon retraining. For ImageNet-1k we obtain an accuracy of 78.842% by inheriting the weights of the optimal architecture in contrast to 75.474% by unconstrained evolutionary search on the on AutoFormer-T space (Chen et al., 2021a), i.e. a net improvement of 3.368% (see Table 6).

4.3.2 MobileNetV3

Next, we study a convolutional search space based on the MobileNetV3 architecture. The search space is defined in Table 19 in the appendix and contains about 2×10^{19} architectures. This follows from the search space designed by OFA (Cai et al., 2020), which searches for kernel-size, number of blocks and channel-expansion factor. Unlike OFA, we do not use the progressive shrinking scheme for the supernet training but activate all choices in our supernet at all times.

Search Type	Optimizer	Top-1 acc
Single-Stage	TangleNAS	**77.424**
Two-Stage	OFA+RS	77.046
	OFA+ES	77.050
Largest Arch	-	77.336

Table 7: Evaluation on MobileNetV3

Table 7 shows that on this OFA search space, TangleNAS still outperforms OFA (based on both unconstrained evolutionary and random search on the pre-trained OFA supernet). Notably, TangleNAS even yields an architecture with higher accuracy than the largest architecture in the space (while OFA yields worse architectures).

4.3.3 Language Modelling Space

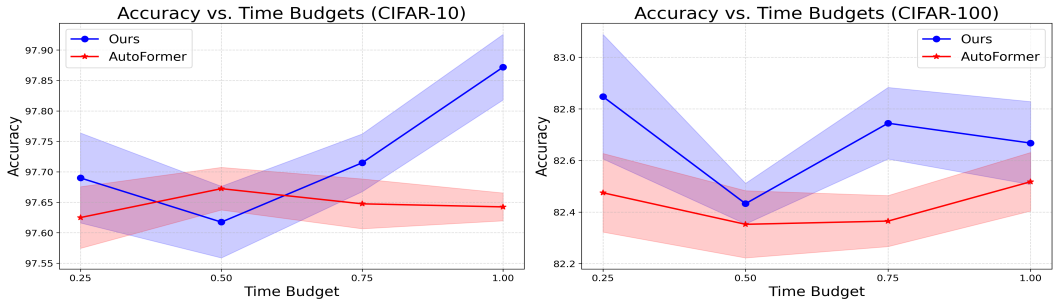
Finally, given the growing interest in efficiently training large language models and recent developments in scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022), we study our efficient and scalable TangleNAS method on a language-model space.

We create our language model space around a smaller version of Andrej Karpathy's nanoGPT model[1]. In this transformer search space, we search for the embedding dimension, number of heads, number of layers and MLP ratios as defined in Table 18 (in the appendix). For each of these transformer search dimensions we consider 3 choices. We evaluate our small language model on the TinyStories (Eldan and Li, 2023) dataset. We improve over the SPOS+ES and SPOS+RS baselines as presented in Table 8. This improvement is statistically very significant with a two-tailed p-value of 0.0064 for ours v/s SPOS+ES and p-value of 0.0127 for ours v/s SPOS+RS.

5 Results and Discussion

For a more in-depth evaluation, we now compare different properties of single-stage and two-stage methods, starting from their any-time performance, impact of the portion of train-val split on their performance and centered-kernel alignment of the supernet feature maps. Moreover, we also study the effect of pretraining, fine-tuning and retraining on the AutoFormer space for CIFAR10 and CIFAR100 and downstream performance of imagenet pre-trained best model on several classification datasets.

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



(a) CIFAR10 (b) CIFAR100

Figure 4: Any Time performance curves of AutoFormer vs. Ours

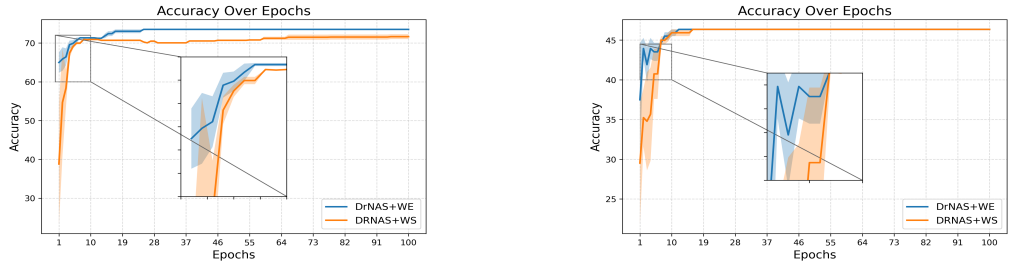
Finally, we discuss the insights derived from our single-stage approach in designing architectures on real world tasks.

Space and time complexity In practice, we observe that vanilla gradient-based NAS methods are memory and compute expensive in comparison to both two-stage methods and our TangleNAS approach with weight-superposition (described in Figure 2). In particular on the NB201 and DARTS search spaces we observe a 25.276% and 35.545% reduction in memory requirements for TangleNAS over DrNAS with WS. This issue only exacerbates for weight-entangled spaces like AutoFormer and MobileNetV3, making the application of vanilla gradient-based methods practically infeasible. This is because vanilla gradient-based methods incur an $\mathcal{O}(n)$ overhead over TangleNAS in terms of space and time complexity, where n is the number of operation choices.

Search Type	Optimizer	Test loss	Perplexity ↓
Single-Stage	TangleNAS	$\mathbf{1.412 \pm 0.011}$	**4.104**
Two-Stage	SPOS+RS	1.433 ± 0.005	4.191
	SPOS+ES	1.444 ± 0.013	4.238

Table 8: Comparison of single and two stage methods on language model search space. We report the test loss and perplexity on the tinystories dataset

Any-Time performance An essential characteristic of any neural architecture search (NAS) method is the attainment of strong *any-time performance*. End users of NAS techniques frequently prioritize the expedited identification of competitive architectures within limited time constraints. This becomes particularly crucial in light of the escalating computational expenses associated with training sophisticated and large neural networks, such as Transformers. Strong any-time performance is a pivotal criterion for the effective applicability of NAS in such resource-intensive domains. We thus compare the any-time performance of TangleNAS. Figure 3 demonstrates that TangleNAS (DrNAS+WE) is faster than its baseline method DrNAS+WS. Similarly, Figure 4 shows that TangleNAS has better anytime performance than AutoFormer.



(a) CIFAR100 (b) ImageNet16-120

Figure 3: Test accuracy evolution over epochs for NB201

Effect of fraction of training data One-shot NAS commonly employs a 50%-50% train-valid split for cell-based spaces and 80%-20% for weight-entangled spaces. To avoid potential confounding factors, we also study our method across different data splits for each search space. The results reported in Appendix A are largely similar across splits. In particular, we observe that single-stage methods are quite robust and performant across training fractions and search spaces, in comparison to two-stage methods.

Architecture design insights In transformer spaces reducing the *mlp-ratio* in the initial layers has a relatively low impact on performance (Figure 5) and can often work competitively or outperform handcrafted architectures. This observation is consistent across ViT and Language Model spaces. Conversely, *number of heads* and *embedding dimension* have a significant impact on the quantitative metric of choice. Pruning a few of the final layers also has a relatively low impact on performance. In the MobileNetV3 space, we find a strong preference for larger *number of channels* and larger *network depth*. In contrast, we discover that scaling laws for transformers may not necessary apply in convolutional spaces especially for *kernel sizes*. We observe that earlier layers favor 5x5 kernel sizes while later ones prefer 7x7 kernel sizes (3 being the smallest and 7 the largest).

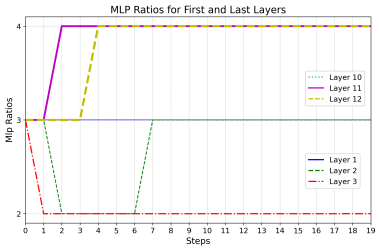


Figure 5: MLP ratio trajectory for LM. Number of layers range from 1-12 and mlp ratio choice can be 2,3 or 4.

Effect of pretraining, fine-tuning and retraining We study the impact of inheriting, fine-tuning, and retraining on the CIFAR10 and CIFAR100 datasets on the AutoFormer space. We observe that retraining almost always outperforms fine-tuning and inheriting. This brings to question the rank correlation between the inherited accuracy and retraining accuracy of two-stage methods and the training interference (Xu et al., 2022) in two-stage methods. Strong rank-correlation is desirable especially in two-stage methods as these methods rely on the performance proxy upon inheritance for the black-box search. Furthermore, we observe that while the SPOS+RS and SPOS+ES methods are performant upon inheritance, TangleNAS outperforms these models after fine-tuning and training from scratch. Improved supernet training properties for single-stage methods is further supported by Centered Kernel Alignment (CKA) analysis (Kornblith et al., 2019) (Table 25).

ImageNet-1k pre-trained architecture on downstream tasks Finally, we study the impact on fine-tuning the best model obtained from the search on downstream datasets. We follow the fine-tuning pipeline proposed in AutoFormer and fine-tune on different fine and coarse-grained datasets. We observe from Table 6 that the architecture discovered by TangleNAS on ImageNet is much more performant in fine-tuning to various datasets (CIFAR-10, CIFAR-100, Flowers, Pets and Cars) than the architecture discovered by SPOS.

6 Conclusion

In this paper, we investigate two of the most widely used NAS methods: *single-stage* and *two-stage*, each traditionally applied to different types of search spaces. We extend the applicability of single-stage NAS methods to weight-entangled spaces, a domain primarily explored by two-stage methods. By doing so, we aim to bridge the gap between these two distinct sub-fields. We empirically evaluate our proposed single-stage method, TangleNAS, on a diverse set of weight-entangled search spaces and tasks, showcasing its ability to outperform conventional two-stage NAS methods while enhancing search efficiency. Furthermore, we highlight the advantages of introducing weight-entanglement into traditional cell-based spaces, whenever feasible. Doing so not only improves the speed of single-stage methods but also reduces memory consumption. The modular nature of TangleNAS enables it to accommodate any differentiable optimizer and hence take advantage of the continually-improving set of methods for gradient-based NAS. Our results on various macro-level search spaces are already very encouraging and we therefore hope that this line of work will enable the NAS community to further improve modern architectures like Transformers.

REFERENCES

Proceedings of the International Conference on Learning Representations (ICLR'19), 2019. Published online: iclr.cc. 10, 11

Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'20), 2020. 10, 11

Proceedings of the International Conference on Learning Representations (ICLR'20), 2020. Published online: iclr.cc. 10, 11

Proceedings of the International Conference on Learning Representations (ICLR'21), 2021. Published online: iclr.cc. 10, 11

G. Bender, P-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le. Understanding and simplifying one-shot architecture search. In Dy and Krause (2018). 2

H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct Neural Architecture Search on target task and hardware. In *Proceedings of the International Conference on Learning Representations (ICLR'19)* icl (2019). Published online: iclr.cc. 2

H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. Once-for-All: Train one network and specialize it for efficient deployment. In *Proceedings of the International Conference on Learning Representations (ICLR'20)* icl (2020). Published online: iclr.cc. 2, 3, 5, 6, 7

M. Chen, H. Peng, J. Fu, and H. Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the 24nd IEEE/CVF International Conference on Computer Vision (ICCV'21)*, pages 12270–12280. cvfandieee, 2021a. 2, 3, 5, 6, 7

X. Chen, R. Wang, M. Cheng, X. Tang, and C-J. Hsieh. Dr{nas}: Dirichlet Neural Architecture Search. In *Proceedings of the International Conference on Learning Representations (ICLR'21)* icl (2021). URL https://openreview.net/forum?id=9FWas6YbmB3. Published online: iclr.cc. 1, 2

X. Dong and Y. Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'19)*, 2019. 1, 2

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020. 5, 13

J. Dy and A. Krause, editors. *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, 2018. Proceedings of Machine Learning Research. 10, 11

R. Eldan and Y. Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023. 7

Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun. Single path one-shot neural architecture search with uniform sampling. In A. Vedaldi, H. Bischof, T. Brox, and J. Frahm, editors, *16th European Conference on Computer Vision (ECCV'20)*, pages 544–560. Springer, 2020. 2, 3, 5

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. 7

S. Hu, S. Xie, H. Zheng, C. Liu, J. Shi, X. Liu, and D. Lin. Dsnas: Direct neural architecture search without parameter retraining. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'20)* cvp (2020), pages 12081–12089. 2

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 7

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMLR, 2019. 9, 18

L. Li, M. Khodak, M. Balcan, and A. Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'21)* icl (2021). Published online: iclr.cc. 2

M. Lindauer and F. Hutter. Best practices for scientific research on Neural Architecture Search. *Journal of Machine Learning Research*, 21(243):1–18, 2020. 19

H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'19)* icl (2019). Published online: iclr.cc. 1, 2, 5, 14

J. Mei, Y. Li, X. Lian, X. Jin, L. Yang, A. Yuille, and J. Yang. Atomnas: Fine-grained end-to-end neural architecture search. In *International Conference on Learning Representations* icl (2019). Published online: iclr.cc. 3

N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik. XNAS: Neural architecture search with expert advice. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche Buc, E. Fox, and R. Garnett, editors, *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*, pages 1975–1985, 2019. 2

H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient Neural Architecture Search via parameter sharing. In Dy and Krause (2018). 2

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 6

Junhong Shen, Misha Khodak, and Ameet Talwalkar. Efficient architecture search for diverse tasks. *Advances in Neural Information Processing Systems*, 35:16151–16164, 2022. 2

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. 16

A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, and K. Chen. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* cvp (2020), pages 12965–12974. 2

H. Wang, Z. Wu, Zhijian Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han. Hat: Hardware-aware transformers for efficient natural language processing. In D. Jurafsky, Joyce J. Chai, Natalie N. Schluter, and J. Tetreault, editors, *Annual Conference of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020. 2

R. Wang, M. Cheng, X. Chen, X. Tang, and C. Hsieh. Rethinking architecture selection in differentiable nas. In *International Conference on Learning Representation* icl (2021). Published online: iclr.cc. 1

Jin Xu, Xu Tan, Kaitao Song, Renqian Luo, Yichong Leng, Tao Qin, Tie-Yan Liu, and Jian Li. Analyzing and mitigating interference in neural architecture search. In *International Conference on Machine Learning*, pages 24646–24662. PMLR, 2022. 9, 18

Y. Xu, L. Xie, X. Zhang, X. Chen, G. Qi, Q. Tian, and H. Xiong. PC-DARTS: Partial channel connections for memory-efficient architecture search. *arXiv:1907.05737 [cs.CV]*, 2019. 2

Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1803–1811, 2019. 3

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2018. 3

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*, pages 702–717. Springer, 2020. 3

A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter. Understanding and robustifying differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'20)* icl (2020). URL https://openreview.net/forum?id=H1gDNyrKDS. Published online: iclr.cc. 2

M. Zhang, S. W. Su, S. Pan, X. Chang, E. M. Abbasnejad, and R. Haffari. idarts: Differentiable architecture search with stochastic implicit gradients. In M. Meila and T. Zhang, editors, *International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12557–12566. PMLR, PMLR, 2021. 2

A TRAINING ACROSS DATA FRACTIONS

Search Type	Optimizer	Train portion	CIFAR10	CIFAR100
Single-Stage	TangleNAS	50%	97.715 ± 0.0877	82.5375 ± 0.11764
		80%	$\mathbf{97.872 \pm 0.05369}$	$\mathbf{82.6675 \pm 0.16073}$
Two-Stage	SPOS+RS	50%	97.6799 ± 0.0262	82.5372 ± 0.27954
		80%	97.7674 ± 0.0239278	82.210 ± 0.14242
	SPOS+RE	50%	97.77 ± 0.03752	82.354 ± 0.11974
		80%	97.6425 ± 0.02286	82.5175 ± 0.1135

Table 9: Evaluation on the AutoFormer-T space for CIFAR10 and CIFAR100

Search Type	Optimizer	Train portion	Supernet	Accuracy (%)		
				CIFAR10	CIFAR100	ImageNet16-120
Single-Stage	DrNAS	50%	WS	$\mathbf{94.36 \pm 0.00}$	72.245 ± 0.7315	$\mathbf{46.37 \pm 0.00}$
		80%		$\mathbf{94.36 \pm 0.00}$	71.1525 ± 0.697	$\mathbf{46.37 \pm 0.00}$
	TangleNAS	50%	WE	$\mathbf{94.36 \pm 0.00}$	$\mathbf{73.51 \pm 0.000}$	$\mathbf{46.37 \pm 0.00}$
		80%		$\mathbf{94.36 \pm 0.00}$	$\mathbf{73.51 \pm 0.000}$	$\mathbf{46.37 \pm 0.00}$
Two-Stage	SPOS+RS	50%	WE	89.107 ± 0.88392	56.865 ± 2.5968	31.665 ± 1.1460
		80%		87.778 ± 2.4462	53.68 ± 4.1736	30.5449 ± 3.6425
	SPOS+ES	50%	WE	87.133 ± 2.6050	56.463 ± 2.3417	29.7849 ± 3.0149
		80%		89.095 ± 0.8250	56.363 ± 4.7244	30.935 ± 3.54648

Table 10: Comparison of test accuracies of single and two stage methods with WS and WE on NB201 search space

Search Type	Optimizer	Train portion	Supernet type	Test acc
Single-Stage	DrNAS	50%	WS	91.19 ± 0.0490
		80%		91.125 ± 0.0334
	TangleNAS	50%	WE	$\mathbf{91.3 \pm 0.023}$
		80%		91.065 ± 0.1633
Two-Stage	SPOS+RS	50%	WE	90.68 ± 0.25345
		80%		90.687 ± 0.10969
	SPOS+ES	50%	WE	90.3175 ± 0.2233
		80%		90.595 ± 0.2188
Optimum	-	-	-	91.63

Table 11: Evaluation on toy cell-based search space on fashion-mnist dataset

Search Type	Optimizer	Train portion	Supernet type	Test acc
Single-Stage	DrNAS	50%	WS	10%
		80%		10%
	TangleNAS	50%	WE	$\mathbf{83.020 \pm 0.000}$
		80%		$82.495 \pm .0.46074$
Two-Stage	SPOS+RS	50%	WE	81.253 ± 0.67174
		80%		81.345 ± 0.38336
	SPOS+ES	50%	WE	81.890 ± 0.8002
		80%		82.322 ± 0.60356
Optimum	-	-	-	84.41

Table 12: Evaluation on toy conv-macro search space on CIFAR10 dataset

Search Type	Optimizer	Train portion	Supernet	CIFAR10	ImageNet
Single-Stage	TangleNAS	50%	WE	$\mathbf{2.556 \pm 0.03406}$	**25.69**
		80%		2.67 ± 0.0756	25.742
	DrNAS	50%	WS	2.625 ± 0.0750	26.29
		80%		2.580 ± 0.02798	**25.67**
Two-Stage	SPOS+RS	50%	WE	2.965 ± 0.07193	27.114
		80%		2.965 ± 0.07193	27.114
	SPOS+RE	50%	WE	3.20000 ± 0.06557	27.424
		80%		3.00249 ± 0.037052	26.76

Table 13: Comparison of test errors of single and two stage methods with WS and WE on DARTS search space

B SEARCH AND TRAINING TIMES

B.1 NB201 AND DARTS

Search Type	Optimizer	Supernet	Search Time (hrs)
Single-Stage	DrNAS	WS	20.9166
	TangleNAS	WE	20.3611
Two-Stage	SPOS+RS	WE	29.51388
	SPOS+ES	WE	26.74721

Table 14: Comparison of search times NB201

Search Type	Optimizer	Supernet	GPU Mem%	Search Time(hrs)
Single-Stage	DrNAS	WS	81.08%	29.4444
	TangleNAS	WE	52.26%	25.472
Two-Stage	SPOS+RS	WE	25.66%	18.7444
	SPOS+ES	WE	25.66%	14.794

Table 15: Comparison of test errors of single and two-stage methods on the DARTS search space

C SEARCH SPACE DETAILS

Toy cell space This particular search space takes its inspiration from the prominently used Differentiable Architecture Search (DARTS) space, and is composed of diminutive triangular cells, with each edge offering four choices of operations: (a) Separable 3x3 Convolution, (b) Separable 5x5 Convolution, (c) Dilated 3x3 Convolution, and (d) Dilated 5x5 Convolution. The macro-architecture of the model comprises of three cells of types reduction, normal, and reduction again stacked together. Notably, we entangle the 3x3 and 5x5 kernel weights for each operation type, i.e., separable convolutions and dilated convolutions. We evaluate these search spaces and their architectures on the Fashion-MNIST dataset by creating a small benchmark which we release here.

Toy conv-macro space This toy space draws its inspiration from MobileNet-like spaces where we search for the number of channels and the kernel size of convolutional layers in a network (also called macro search space) for four convolutional layers. Every convolutional layer has a choice of three kernel sizes and number of channels. See Table 16 for more details. We evaluate this search space on the CIFAR10 dataset by creating a small benchmark which we release here.

Architectural Parameter	Choices
Kernel sizes (all layers)	3, 5, 7
Number of channels (layer 1)	8, 16, 32
Number of channels (layer 2)	16, 32, 64
Number of channels (layer 3)	32, 64, 128
Number of channels (layer 4)	64, 128, 256

Table 16: Toy Convolutional-Macro Search Space

NASBench201 Space The NASBench201 (Dong and Yang, 2020) has a single cell type which is stacked 5 times. Each cell has 4 nodes and each node are connected by operations from amongst skip connection, conv 3x3, conv 1x1, max-pool or average-pool. Opportunities for weight entanglement in this space is limited and we entangle the conv 3x3 and conv 1x1 kernel on every edge for every cell, thus obtaining parameter savings over traditional weight sharing.

DARTS Search Space The DARTS (Liu et al., 2019) search space has a cell with 13 possible edges stacked 8 times. The nodes of the cell are connected by operations amongst :none, max_pool_3x3, avg_pool_3x3, skip_connect, sep_conv_5x5, sep_conv_3x3, dil_conv_5x5, dil_conv_3x3.

AutoFormer Space and Language Model Space We present the details of the AutoFormer Space and the Language Model space is Table 17 and Table 18 respectively.

Architectural Parameter	Choices
Embedding dimension	192, 216, 240
Number of layers	12, 13, 14
MLP ratio (per layer)	3.5, 4
Number of heads (per layer)	3, 4

Table 17: Choices for AutoFormer-T Search Space

Architectural Parameter	Choices
Embedding Dimension	384, 576, 768
Number of Heads	6, 8, 12
MLP Expansion Ratio	2, 3, 4
Number of Layers	5,6,7

Table 18: Choices for Language Model Space

Architectural Parameter	Choices
Kernel sizes (all layers)	3, 5, 7
Channel expansion (all layers)	3,4,6
Number of blocks	2,3,4

Table 19: MobileNet Search Space

D Details on Method

Traditional cell-based search spaces primarily consider independent operations (e.g., convolution or skip). One-shot differentiable optimizers thus have their mixture operations tailored to these search spaces which are not general enough to be applied to macro-level architectural parameters. Consider, e.g., the task of searching for the embedding dimension and the expansion ratio for a transformer. Here, a single operation, i.e., a linear expansion layer, has two different architectural parameters - one corresponding to the choice of embedding dimension and the other to the expansion ratio. To adapt single-stage methods to these *combined* operation choices in the search space, we propose the *combi-superposition operation*.

The combi-superposition operation simply takes the cross product of architectural parameters for the embedding dimension and expansion ratio and assigns its elements to *every combination* of these dimensions, allowing us to optimize jointly in this space without the need for a separate forward pass for each combination. Every combination maps to a unique sub-matrix of the operator weight matrix, indexed using both the embedding dimension and the expansion ratio. To address shape mismatches of the different operation weights during forward passes, every sub-matrix is zero-padded to match the shape of the largest matrix. See Algorithm 2 for more details, and Figure 2(b) for an overview of the idea.

Algorithm 2 Combi-Superposition Operation

$embed\_dim \leftarrow [e_1, e_2, e_3, \ldots, e_n]$ {Choices for embedding dimension}
$expansion\_ratio \leftarrow [r_1, r_2, r_3, \ldots, r_m]$ {Choices for expansion ratio}
$\alpha \leftarrow [\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n]$ {Architecture parameters for embedding dimension}
$\beta \leftarrow [\beta_1, \beta_2, \beta_3, \ldots, \beta_m]$ {Architecture parameters for expansion ratio}
$X \leftarrow input\_feature$
$W, b \leftarrow fc\_layer\_weight, fc\_layer\_bias$
$W_{mix} \leftarrow \mathbf{0}$
$b_{mix} \leftarrow \mathbf{0}$
$\alpha, \beta \leftarrow \text{normalize}(\alpha), \text{normalize}(\beta)$ {Normalize architecture parameters (e.g., using softmax)}
$\alpha\beta \leftarrow \alpha \times \beta$ {Cross product of α and β for dimension $n \times m$}
for $i \leftarrow 1$ to $n \times m$ **do**
  $W\_i = W[: (embed\_dim[i] \times expansion\_ratio[i]), : embed\_dim[i]]$
  $b\_i = b[: embed\_dim[i] \times expansion\_ratio[i]]$
  $W_{mix} = W_{mix} + \alpha\beta[i] \times \text{PAD}(W_i)$
  $b_{mix} = b_{mix} + \alpha\beta[i] \times \text{PAD}(b_i)$
end for
$Y \leftarrow X \cdot W_{mix} + b_{mix}$ {Compute the output of the FC layer with a mixture of weights and bias}
return Y

For completeness and for comparison with TangleNAS (Algorithm 1), we present Algorithm 3 which describes a generic two-stage method on a macro search space with WE. Also, vanilla single-stage methods on cell-based WS spaces follow Algorithm 4.

Algorithm 3 Weight Entanglement (Two-Stage)

Input: $M \leftarrow$ number of cells, $N \leftarrow$ number of operations
$\mathcal{O} \leftarrow [o_1, o_2, o_3, ...o_N]$
$\mathcal{W}_{\max} \leftarrow \cup_{i-1}^{N} w_i$
$\eta =$ learning rate of $\mathcal{W}_{\max_{\mathcal{O}}}$
$Cell_j \leftarrow DAG(\mathcal{O}_{|}, \mathcal{W}_{\max_{|}})$ /* defined for j=1...M */
$Supernet \leftarrow \cup_{i}^{M} Cell_i$
/* example of forward propagation on the cell */
for $j \leftarrow 1$ to M **do**
  $i \sim \mathcal{U}(1, N)$
  /* Slice weight matrix corresponding to operation */
  $o_j(x, \mathcal{W}_{max}) = o_{(j,i)}(x, \mathcal{W}_{max}[:i])$
end for
/* weights update */
$\mathcal{W}_{\max}[:i] = \mathcal{W}_{\max}[:i] - \eta \nabla_{\mathcal{W}_{\max}}[:i] \mathcal{L}_{train}(\mathcal{W}_{\max})$
/* Search */
$Supernet^*$ $\leftarrow$ pre-trained supernet
$selected\_arch \leftarrow$ Evolutionary-Search($Supernet^*$)

Algorithm 4 Weight Sharing (Single-Stage)

Input: $M \leftarrow$ number of cells, $N \leftarrow$ number of operations
$\mathcal{O} \leftarrow [o_1, o_2, o_3, ...o_N]$
$\mathcal{W}_{\mathcal{O}} \leftarrow [w_1, w_2, w_3, ...w_N]$
$\mathcal{A} \leftarrow [\alpha_1, \alpha_2, \alpha_3, ...\alpha_N]$
$\gamma =$ learning rate of $\mathcal{A}$
$\eta =$ learning rate of $\mathcal{W}_{\mathcal{O}}$
f is a function or distribution s.t. $\sum_{i=1}^{N} f(\alpha_i) = 1$
$Cell_i \leftarrow DAG(\mathcal{O}_{|}, \mathcal{W}_{\mathcal{O}_{|}})$ /* defined for i=1...M */
$Supernet \leftarrow \cup_{i}^{M} Cell_i \cup \mathcal{A}$
/* example of forward propagation on the cell */
for $j \leftarrow 1$ to M **do**
  /* Compute mixture operation as weighted sum of output of operations*/
  $\overline{o_j(x, \mathcal{W}_{\mathcal{O}})} = \sum_{i=1}^{N} f(\alpha_i)\, o_{(j,i)}(x, w_{(j,i)})$
end for
/* weights and architecture update */
$\mathcal{A} = \mathcal{A} - \gamma \nabla_{\mathcal{A}} \mathcal{L}_{val}(\mathcal{W}_{\mathcal{O}}^*, \mathcal{A})$
$\mathcal{W}_{\mathcal{O}} = \mathcal{W}_{\mathcal{O}} - \eta \nabla_{\mathcal{W}_{\mathcal{O}}} \mathcal{L}_{train}(\mathcal{W}_{\mathcal{O}}, \mathcal{A})$
/* Search */
$selected\_arch \leftarrow \arg\max(\mathcal{A})$

E Experimental Setup

E.1 Toy Search Spaces

Search Space	Cell Space	Conv Macro
Epochs	100	100
Learning rate (LR)	0.1	3e-4
Min. LR	0.001	0.0001
Optimizer	SGD	Adam
Architecture LR	0.0003	0.0003
Batch Size	64	64
Momentum	0.9	0.9
Nesterov	True	False
Weight Decay	0.0005	0.0005
Arch Weight Decay	0.001	0.001
Regularization Type	L2	L2
Regularization Scale	0.001	0.001

Table 20: Configurations used in the DrNAS experiments on Toy Spaces.

Search Space	Cell Space	Conv Macro
Epochs	250	250
Learning rate (LR)	0.1	3e-4
Min. LR	0.001	0.0001
Optimizer	SGD	Adam
Batch Size	64	64
Momentum	0.9	0.9
Nesterov	True	False
Weight Decay	0.0005	0.0005

Table 21: Configurations used in the SPOS experiments on Toy Spaces.

E.2 LANGUAGE MODEL

We use the AdamW optimizer in all experiments pertaining to language modelling. Other hyperparameter choices are as in Table 22.

Search Space	Small-LM
Learning rate (LR)	5e-4
Min LR	5e-5
Beta2	0.99
Warmup Iters	100
Max Iters	6000
Lr decay iters	6000
Batch size	12
Weight decay	1e-1

Table 22: Configurations used in DrNAS on the Language Model Spaces

E.3 AUTOFORMER AND OFA

We use a 50%-50% train and validation split for the CIFAR10 and CIFAR100 dataset for the cell-based spaces and a 80%-20% for the weight-entangled spaces We use the official source code of AutoFormer available at code for all the AutoFormer experiments on CIFAR10 and CIFAR100. We closely follow the AutoFormer training pipeline and search space design. AutoFormer searched on three transformer sizes Autoformer-T (tiny), AutoFormer-S (small) and Autoformer-B (base). We currently restrict ourselves to Autoformer-T. For baselines like OFA and AutoFormer we follow their respective recipes to obtain the train-valid split for ImageNet-1k. Our models were trained on 2xA100s with the same effective batch-size as AutoFormer. For MobileNetV3 from once-for-all we use the same training hyperparameters as the baseline as here (in addition to architectural parameters same as Table 23) .

E.3.1 AUTOFORMER FINE-TUNING

C10 and C100 pretrained supernet We fine-tune the C10 and C100 selected networks (after inheriting them from the supernet) for 500 epochs. We set the learning rate to 1e-3, the warmup epochs to 5 , the warmup learning rate to 1e-6 and the min learning rate to 1e-5. All other hyperparameters are set same as Appendix E.3.

ImageNet pretrained supernet We follow the DeiT (Touvron et al., 2021) finetuning pipeline as used in AutoFormer. Mainly we set the epochs to 1000, the warmup-epochs to 5 , the scheduler to cosine, the mixup to 0.8, the smoothing to 0.1, the weight-decay to 1e-4, the batch-size 64, the optimizer to sgd, the learning rate to 0.01 and the warmup-lr to 0.0001 for all datasets.

E.4 NB201 AND DARTS

For single-stage optimizers, the supernet was trained with four different seeds. The supernet with the best validation performance from these four was discretized to obtain the final model, which is then trained from scratch four times to obtain the results shown in the table. For two-stage methods, yet again, we train the supernet four times, and perform Random Search (RS) and Evolutionary Search (ES) on each of them. The best model obtained across all four supernets for both methods are then trained from scratch with four seeds to compute the final results. For each DrNAS we follow the same recipe as suggested by the authors across all search spaces. To accommodate for multiple training recipes, we write a configurable training pipeline. The configurations for DrNAS, and SPOS are shown in Table 23 and Table 24, respectively.

Search Space	DARTS	NB201
Epochs	50	100
Learning rate (LR)	0.1	0.025
Min. LR	0.0	0.001
Architecture LR	0.0006	0.0003
Batch Size	64	64
Momentum	0.9	0.9
Nesterov	True	False
Weight Decay	0.0003	0.0003
Arch Weight Decay	0.001	0.001
Partial Connection Factor	6	-
Regularization Type	L2	L2
Regularization Scale	0.001	0.001

Table 23: Configurations used in the DrNAS experiments.

Search Space	DARTS	NB201
Epochs	250	250
Learning rate (LR)	0.025	0.025
Min. LR	0.001	0.001
Architecture LR	0.0003	0.0003
Batch Size	256	64
Momentum	0.9	0.9
Nesterov	True	True
Weight Decay	0.0005	0.0005
Arch Weight Decay	0.001	0.001

Table 24: Configurations used in the SPOS experiments.

F Optimal architectures derived

F.1 DARTS

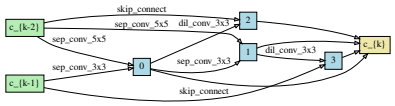


Figure 6: DRNAS with WE normal cell

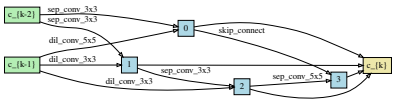


Figure 7: DRNAS with WE reduction cell

F.2 Small LM

num_layers: 7, embed_dim: 768, num_heads: [12, 12, 12, 12, 12, 12, 12], mlp_ratio: [3, 4, 4, 4, 4, 4, 4]

F.3 AUTOFORMER

F.3.1 CIFAR10

50%50% split mlp_ratio:$[4, 4, 4, 4, 4, 4, 4, 4, 4, 3.5, 4, 4, 3.5, 4]$, num_heads:$[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]$, num_layers: 14 embed_dim: 216

80%-20% split mlp_ratio:$[4, 4, 4, 4, 4, 4, 4, 4, 4, 3.5, 4, 4, 4, 3.5, 3.5]$, num_heads:$[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]$, depth: 14, embed_dim: 240

F.3.2 CIFAR100

50%-50% split mlp_ratio: [4,4,4,4,4,4,4,4,3.5,4,4,4,4,4], num_heads:[4,4,4,4,4,4,4,4,4,4,4,4,4,4], depth: 14, embed_dim: 216

80%-20% split mlp_ratio:[3.5,4,4,4,4,4,4,4,4,4,4,4,4,4], num_heads:[4,4,4,4,4,4,4,4,4,4,4,4,4,4, depth: 14, embed_dim: 240

F.3.3 IMAGENET1-K

mlp_ratio:[4,4,4,4,4,4,4,4,4,4,4,4,4,4], num_heads:[4,4,4,4,4,4,4,4,4,4,4,4,4,4, depth: 14, embed_dim: 240

F.4 MOBILENETV3

Kernel_sizes:[7,5,5,7,5,5,7,7,5,7,7,7,5,7,7,5,5,7,7,5],Channel_expansion_factor:[6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6], Depths : [4, 4, 4, 4, 4]

F.5 TOY SPACES

F.5.1 TOY CELL (OUR BEST ARCHITECTURE)

50%-50% split Genotype(normal=[('dil_conv_3x3', 0), ('dil_conv_3x3', 0), ('sep_conv_3x3', 1)], normal_concat=range(1, 3), reduce=[('sep_conv_3x3', 0), ('sep_conv_3x3', 0), ('dil_conv_3x3', 1)], reduce_concat=range(1, 3))

80%-20% split Genotype(normal=[('dil_conv_3x3', 0), ('dil_conv_5x5', 0), ('sep_conv_3x3', 1)], normal_concat=range(1, 3), reduce=[('sep_conv_3x3', 0), ('sep_conv_5x5', 0), ('dil_conv_3x3', 1)], reduce_concat=range(1, 3))

F.6 TOY CONV-MACRO (OUR BEST ARCHITECTURE)

50%-50% split: Channels = [32, 64, 128, 64], Kernel Sizes = [5, 5, 7, 7].

Train-Val fraction 80%-20%: Channels = [32, 64, 128, 64], Kernel Sizes = [5, 5, 7, 7].

G ARCHITECTURE REPRESENTATION ANALYSIS

CKA Centered Kernel Alignment (CKA) (Kornblith et al., 2019) is a metric, based on the Hilbert- Schmidt Independence Criterion (HSIC). This metric is primarily designed to model similarity between representations in neural networks. In this section we study the CKA between structurally identical layers in the inherited, fine-tuned and retrained networks in the AutoFormer-T space. Precisely, the goal here is to study how similar the inherited and fine-tuned representations are to the actual model trained from scratch. Table 25 Presents the average CKA values for a fixed subsample of the CIFAR10 and CIFAR100 dataset. We observe that the representations learned by the single-stage supernet are more similar to the architecture finetuned or trained from scratch. This further emphasizes the potential issues with using the inherited weights from the supernet as proxy for search for two-stage methods as highlighted in (Xu et al., 2022).

Model	Inherit v/s Retrain		Fine-Tune v/s Retrain	
	CIFAR10	CIFAR100	CIFAR10	CIFAR100
TangleNAS	**0.4630**	**0.5853**	0.57125	**0.65275**
SPOS+ES	0.45812	0.57932	0.569374	0.6309
SPOS+RS	0.44124	0.583	**0.5797**	0.638948

Table 25: CKA correlation between layers

H NAS BEST PRACTICES CHECKLIST

We now describe how we addressed the individual points of the NAS best practice checklist (Lindauer and Hutter, 2020).

1. **Best Practices for Releasing Code**

   For all experiments you report:

   (a) Did you release code for the training pipeline used to evaluate the final architectures? Yes
   (b) Did you release code for the search space? Yes
   (c) Did you release the hyperparameters used for the final evaluation pipeline, as well as random seeds? Yes
   (d) Did you release code for your NAS method? Yes
   (e) Did you release hyperparameters for your NAS method, as well as random seeds? Yes

2. **Best practices for comparing NAS methods**

   (a) For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code? Yes
   (b) Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)? Yes
   (c) Did you run ablation studies? Yes
   (d) Did you use the same evaluation protocol for the methods being compared? Yes
   (e) Did you compare performance over time? Yes
   (f) Did you compare to random search? Yes. Black box random search is very expensive in our case, hence we use the supernet from single-path-one-shot(SPOS) for random search.
   (g) Did you perform multiple runs of your experiments and report seeds? Yes
   (h) Did you use tabular or surrogate benchmarks for in-depth evaluations? Yes

3. **Best practices for reporting important details**

   (a) Did you report how you tuned hyperparameters, and what time and resources this required? N/A. We did not tune any hyperparameters.
   (b) Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)? Yes
   (c) Did you report all the details of your experimental setup? Yes