A Appendix

A.1 Details of Training and Search with ViTAS

In this section, we present the details of training recipe w.r.t. different models and datasets. Concretely, we uniformly partitioned transformer space with 10 groups to implement the search of dimensions.

Search with ViTAS on ImageNet-1k dataset. We use the same training recipe of superformer for both DeiT- and Twins-based transformer space. As illustrated in Table 5, all superformers are trained for 300 epochs with a batch size of $1,024$ with the AdamW optimizer [28]. The learning rate is initialized to be 0.001 and decayed to zero with a cosine strategy. Besides, we also leverage a linear warm-up in the first five epochs same as [5]. Note that we do not leverage any further extra data augmentation/regularization for training the superformer.

Training recipe of searched models on ImageNet-1k dataset. For DeiT-based architectures, we follow the same recipe as [41]. In detail, we use a weight decay of 0.05 and batch size of $1,024$, and we train the models by 300 epochs with the learning rate decayed with cosine strategy from initial value 0.05 to 0. Except for repeated augmentation, we adopt all other recipes same as DeiT [41], *i.e.*, 5 epochs for warmup, 0.1 of label smoothing, 0.1 of stochastic depth, rand augmentation, mixup, cutmix, and random erasing. For Twins-based architectures, compare to DeiT recipe, we use the stochastic depth augmentation of 0.1, 0.2, 0.3, 0.5 for tiny, small, base, and large models, respectively. We also use gradient clipping with a max norm of 5.0 to stabilize the training process for twins as in [5].

Training recipe of models on COCO2017 dataset. We transfer the searched architectures to COCO2017 [13] and ADE20k [67] based on MMDetection [3] and with the same recipes as [5]. We train all the models with AdamW optimizer of 12 epochs and batch size of 16. The learning rate is initialized as 1×10^{-4} and 2×10^{-4} for RetinaNet and Mask R-CNN, respectively. The learning rate is started with 500-iteration warmup and decayed by $10\times$ at the 8-th and 11-th epoch, respectively. We set stochastic drop path regulation as 0.2 and weight decay of 0.0001.

Training recipe of models on ADE20k dataset. For the semantic FPN framework, we leverage the same setting as [44,5]. We train models with AdamW and 80K steps with a batch size of 16. The learning rate is set as 1×10^{-4} and decayed with "poly strategy" with the power coefficient of 0.9. We use the drop-path rate of 0.2 for small, base models while 0.4 for large model to avoid over-fitting. For the UperNet framework, we leverage the recipe provided in [46,5]. In detail, we train models with AdamW optimizer for 160 iterations and a batch size of 16. The initial learning rate is set to 6×10^{-5} and linearly decay to zero. The drop-path is set to 0.2 for backbone and weight decay is of 0.01 for the whole model.

A.2 Transformer Space of ViTAS

In this section, we present the constitution of transformer space w.r.t. different sizes for the Twins and DeiT. We incorporate all the essential elements in our transformer space,

including *head number*, *patch size*, *output dimension of each layer*, and *depth of the architectures*.

Table 10 describes the defined Twins-based ViTAS transformer space. The block setting (the default order and numbers of MHSA and FC layers) is inspired by [11,41]. The maximum depth of each stage is set as original setting of Twins plus two for ID search and the head number h in the MHSA is chosen within the set $\{3, 6, 12, 16\}$. Meanwhile, the way of sharing FC_1 in the MHSA is to evenly divide its output features into h groups (*i.e.*, h heads), while different h leads to different dimensions D/h of a group. Furthermore, in order to accommodate the output dimensions w.r.t. each stage, the maximum dimension of each layer is set to $2\times$ of baseline method, and the dimension can be selected from a group of ten settings, *i.e.*, $\{\frac{i}{10}\}_{i=1}^{10}$. Given the defined transformer space, we encourage each block in the ViT to freely select their own optimal head numbers and output dimensions. Therefore, with the Twins-small based transformer space as an example, the size of transformer space amounts to 1.1×10^{54} and the FLOPs (parameters) ranges from 0.02G (0.16M) to 11.2G (86.1M).

Table 10: Macro transformer space for the ViTAS of Twins-based architecture."TBS" indicates that layer type is searched from parametric operation and identity operation for depth search. "Embedding" represents the patch embedding layer. "$\mathrm{Max_a}$" and "$\mathrm{Max_m}$" indicates the max dimension of attention layer ("$\mathrm{Max_a}$" also used for patch embedding layer) and MLP layer, respectively. "Ratio" means the reduction ratio from the "Max Output Dim". A larger "Ratio" indicates a larger dimension.

(a) Twins tiny transformer space.

Number	OP	Type	Patch size / #Heads	$\mathrm{Max_a}$	$\mathrm{Max_m}$	Ratio
1	False	Embeding	4	128	-	$\{i/10\}_{i=1}^{10}$
4	TBS	Local Global	$\{2, 4, 8, 16\}$	480	512	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	256	-	$\{i/10\}_{i=1}^{10}$
4	TBS	Local Global	$\{2, 4, 8, 16\}$	960	1024	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	512	-	$\{i/10\}_{i=1}^{10}$
12	TBS	Local Global	$\{2, 4, 8, 16\}$	1920	2048	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	1024	-	$\{i/10\}_{i=1}^{10}$
6	TBS	Local Global	$\{2, 4, 8, 16\}$	3840	4096	$\{i/10\}_{i=1}^{10}$

(b) Twins small transformer space.

Number	OP	Type	Patch size / #Heads	$\mathrm{Max_a}$	$\mathrm{Max_m}$	Ratio
1	False	Embeding	4	128	-	$\{i/10\}_{i=1}^{10}$
4	TBS	Local Global	$\{2, 4, 8, 16\}$	480	512	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	256	-	$\{i/10\}_{i=1}^{10}$
4	TBS	Local Global	$\{2, 4, 8, 16\}$	960	1024	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	512	-	$\{i/10\}_{i=1}^{10}$
12	TBS	Local Global	$\{2, 4, 8, 16\}$	1920	2048	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	1024	-	$\{i/10\}_{i=1}^{10}$
6	TBS	Local Global	$\{2, 4, 8, 16\}$	3840	4096	$\{i/10\}_{i=1}^{10}$

(c) Twins base transformer space.

Number	OP	Type	Patch size / #Heads	$\mathrm{Max_a}$	$\mathrm{Max_m}$	Ratio
1	False	Embeding	4	192	-	$\{i/10\}_{i=1}^{10}$
4	TBS	Local Global	$\{2, 4, 8, 16\}$	480	768	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	384	-	$\{i/10\}_{i=1}^{10}$
4	TBS	Local Global	$\{2, 4, 8, 16\}$	960	1536	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	768	-	$\{i/10\}_{i=1}^{10}$
20	TBS	Local Global	$\{2, 4, 8, 16\}$	1920	3072	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	1536	-	$\{i/10\}_{i=1}^{10}$
4	TBS	Local Global	$\{2, 4, 8, 16\}$	3840	6144	$\{i/10\}_{i=1}^{10}$

(d) Twins large transformer space.

Number	OP	Type	Patch size / #Heads	$\mathrm{Max_a}$	$\mathrm{Max_m}$	Ratio
1	False	Embeding	4	256	-	$\{i/10\}_{i=1}^{10}$
4	TBS	Local Global	$\{2, 4, 8, 16\}$	960	1024	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	512	-	$\{i/10\}_{i=1}^{10}$
4	TBS	Local Global	$\{2, 4, 8, 16\}$	1920	2048	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	1024	-	$\{i/10\}_{i=1}^{10}$
20	TBS	Local Global	$\{2, 4, 8, 16\}$	3840	4096	$\{i/10\}_{i=1}^{10}$
1	False	Embeding	2	2048	-	$\{i/10\}_{i=1}^{10}$
4	TBS	Local Global	$\{2, 4, 8, 16\}$	7680	8192	$\{i/10\}_{i=1}^{10}$

Similarly, Table 11 describes the definition of DeiT-based transformer space. "Max Dim" indicates the output dimensions of both attention and MLP blocks. With the DeiT-small transformer space, the size of transformer space amounts to 5.4×10^{34} and the FLOPs (parameters) ranges from 0.1G (0.5M) to 20.0G (97.5M).

Table 11: Macro transformer space for the ViTAS of DeiT-based architecture."TBS" indicates that layer type is searched from vanilla ViT block or identity operation for depth search. "Ratio" means the reduction ratio from the "Max Dim". A larger "Ratio" means a larger dimension.

(a) DeiT tiny transformer space.

Number	OP	Type	Patch size / #Heads	Max Dim	Ratio
1	False	Linear	$\{14, 16, 32\}$	384	$\{i/10\}_{i=1}^{10}$
14	TBS	MHSA	$\{3, 6, 12, 16\}$	1440	$\{i/10\}_{i=1}^{10}$
		MLP	-	1440	$\{i/10\}_{i=1}^{10}$

(b) DeiT small transformer space.

Number	OP	Type	Patch size / #Heads	Max Dim	Ratio
1	False	Linear	$\{14, 16, 32\}$	768	$\{i/10\}_{i=1}^{10}$
14	TBS	MHSA	$\{3, 6, 12, 16\}$	2880	$\{i/10\}_{i=1}^{10}$
		MLP	-	2880	$\{i/10\}_{i=1}^{10}$

A.3 Evolutionary Search

To avoid the exhausted search from the enormous (*e.g.*, 1.1×10^{54} for Twins small) transformer space and boost the search efficiency, we leverage the multi-objective NSGA-II [6] algorithm for evolutionary search, which is easy to accommodate the constraint budgets (*e.g.*, FLOPs, GPU throughput). Concretely, we set the population size and generation number as 50 and 40, respectively, which amounts to $2,000$ searched paths in ViTAS. To implement the search, we randomly select 50 paths within the pre-set FLOPs as the initial population. Then, we select the top 20 performance architectures as the parents to generate new generalization architectures via mutation and crossover. After the search, we only leverage the architecture with the highest performance during search to train from scratch and report its performance.

A.4 Algorithm of ViTAS

The details about ViTAS are presented in Algorithm 1.

A.5 Coefficient Factors w.r.t. Kendall, Pearson, and Spearman

In Section 6.4, we provide a detailed comparison between ordinal, bilateral, and our cyclic weight sharing paradigm on $2,000$ searched paths w.r.t. three coefficient factors. Indeed, the Pearson ρ_S coefficient aims to evaluate to what degree a monotonic function fits the relationship between two random variables. Besides, the Spearman ρ_S is defined as the Pearson correlation coefficient between the rank variables. Therefore, Pearson and Spearman coefficients share the same formulated equation Eq. (14) but with different value types (*e.g.*, original value and ranks for Pearson and Spearman coefficients, respectively). Defining $\boldsymbol{r}$ and $\boldsymbol{s}$ as two groups of data, the Spearman coefficient ρ_S can be computed by

$$\rho_S = \frac{\mathrm{cov}(r, s)}{\sigma_r \sigma_s}, \tag{14}$$

Algorithm 1 Vision Transformer Architecture Search

Require: The number of groups $\mathcal{G}$. The training epochs of superformer $\mathcal{N}$. Searching methods $\mathcal{A}$. Training dataset $\mathcal{D}_{tr}$. Validation dataset $\mathcal{D}_{val}$.

1: initialize the superformer $\mathcal{N}$ and Cyclic weight sharing paradigm with Eq.(3) $\sim$ Eq.(13) and groups $\mathcal{G}$
2: initialize the search space with identity shifting.
3: **while** epochs $< \mathcal{E}$ **do**
4: randomly assign the training and backwards path
5: updating the corresponding path of groups with dataset $\mathcal{D}_{tr}$.
6: **end while**
7: search the optimized architecture within superformer $\mathcal{N}$ by searching methods $\mathcal{A}$ (as illustrated in section A.3).
8: train the optimized architecture from scratch for evaluation.

Ensure: the optimized architecture with evaluation results =0

where $\mathrm{cov}(\cdot, \cdot)$ is the covariance of two variables, and σ_r and σ_s are the standard deviations of r and s, respectively. In our experiments, the ranks are distinct integers. Therefore, the Eq. (14) can be also reformulated as

$$\rho_S = 1 - \frac{6 \sum_{i=1}^{n} (\boldsymbol{r}_i - \boldsymbol{s}_i)^2}{n(n^2 - 1)}, \tag{15}$$

where $n = 2000$ defines the number of overlapped elements between variables.

The Kendall τ coefficient aims to evaluate the pairwise ranking performance. Given a pair of $(\boldsymbol{r}_i, \boldsymbol{r}_j)$ and $(\boldsymbol{s}_i, \boldsymbol{s}_j)$, if we have either both $\boldsymbol{r}_i > \boldsymbol{r}_j$ and $\boldsymbol{s}_i > \boldsymbol{s}_j$, or both $\boldsymbol{r}_i < \boldsymbol{r}_j$ and $\boldsymbol{s}_i < \boldsymbol{s}_j$, these two pairs are considered as **concordant**. Otherwise, it is said to be **disconcordant**. With the concordant and disconcordant pairs, the Kendall τ can be formulated as

$$K_\tau = \frac{n_{\text{con}} - n_{\text{discon}}}{n_{\text{all}}}, \tag{16}$$

where n_{con} and n_{discon} represent the number of concordant and disconcordant pairs, and n_{all} is the total number of pairs.

A.6 Ablation Studies of Private Tokens with DeiT-based Superformer

For DeiT-based architecture, it usually leverages the trainable vector named class token to perform the prediction task. The class token is appended to the patch tokens before the first layer and go through the architecture for the classification task. However, for the superformer, different architectures assume to share the weights with the weight sharing paradigm, which blurs the performance gap with the shared class token. Indeed, we propose to private the class tokens to cater for the variance of different paths in superformer.

To evaluate the effect of private class tokens on superformer, we implement the search with/without private tokens and retrain the searched architectures from scratch and report the performance as in Table 12.

Table 12: Ablation studies of the proposed ViTAS w.r.t. private token. †: architectures that are searched with private token.

models	FLOPs (G)	Throughput (image/s)	Params (M)	Top-1(%)	Top-5(%)
ViTAS_DeiT_A	1.4	2,842.7	6.8	75.1	92.3
ViTAS_DeiT_A†	1.4	2,831.1	6.6	75.6	92.5
ViTAS_DeiT_B	5.0	1,134.8	24	79.9	95.0
ViTAS_DeiT_B†	4.9	1,189.4	23	80.2	95.1

A.7 Effect of ViTAS as a superformer with DeiT search space.

To investigate the effect of ViTAS w.r.t. baseline methods with DeiT search space, we perform the search with AutoFormer (ordinal) [60,4], BCNet (Bilateral) [37], ViTAS (Cyclic), respectively. As in Table 13, with only Cyclic pattern, our ViTAS (73.9%) achieves 1.1% or 1.6% on Top-1 accuracy gain compare to bilateral (72.8%) and ordinal (72.3%) pattern. Besides, when adopting all strategies in this paper, our ViTAS (75.6%) can attain 0.4% or 0.9% performance gain compare to baselines of bilateral (75.2%) and ordinal (74.7%) weight sharing mechanism.

Table 13: More detailed experiments w.r.t. weak_aug (WA), private token (PT), and Identity shifting (IF) in DeiT space on ImageNet.

	None	PT	WA	IF	PT+WA	PT+IF	WA+IF	PT+WA+IF
Ordinal	72.3	72.5	73.5	72.6	73.7	73.0	74.1	74.7
Bilateral	72.8	73.0	73.9	73.5	74.2	73.7	74.6	75.2
Cyclic	73.9	74.1	74.8	74.5	74.9	74.4	75.1	75.6

A.8 Comparisons between ViTAS and AutoFormer [4] with the same DeiT space as AutoFormer

In AutoFormer [4], it adopts some addition tricks during retraining for a high accuracy performance. For example, AutoFormer adopts the global average pooling rather than the class token for the classification result, which is the same as Twins search space. Besides, in AutoFormer, it also leverages a different formation of q, k, v in self- attention. Moreover, it includes additional relative positional embeddings of k and v to boost the retraining result. All these strategies contribute to a better retraining result compair to the original DeiT search space.

To provide a fair comparison with AutoFormer, we also use the strategies in AutoFormer to retrain our searched ViTAS-DeiT-A and ViTAS-DeiT-B from Table 6. With the settings same as AutoFormer, the performance of ViTAS in DeiT space can be updated as in Table 14. In this paper, only results of ViTAS in Table 14 leverage the same strategies as AutoFormer.

Table 14: Searched ViT architectures that do not involve inductive bias w.r.t. different FLOPs and GPU throughput on ImageNet-1k. $\star$ indicates that the re-implementation results of important baseline methods with our recipe. Our results are highlighted in bold.

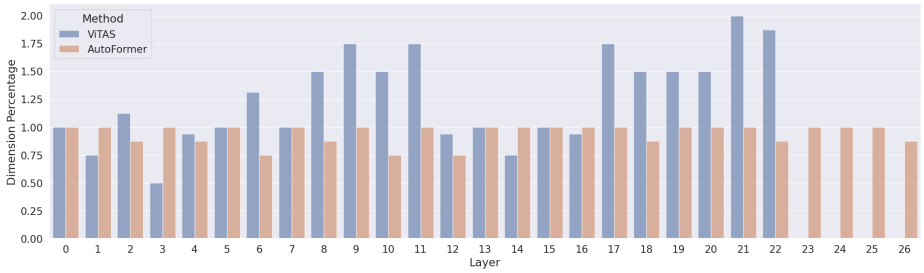
Method	FLOPs (G)	Throughput (image/s)	Params (M)	Top-1 (%)	Top-5 (%)
ResNet-18 [17]	1.8	4458.4	12	69.8	89.1
DeiT-T* [41]	1.3	2728.5	5	72.3	91.4
AutoFormer-T* [4]	1.3	2955.4	5.7	74.7	91.9
ViTAS-DeiT-A	1.4	2827.4	6.8	**75.7**	**92.7**
ResNet-50 [17]	4.1	1226.1	25	76.2	91.4
DeiT-S* [41]	4.6	940.4	22	79.9	95.0
AutoFormer-S* [4]	5.1	1231.7	22.9	81.7	96.8
ViTAS-DeiT-B	5.0	1165.7	2.5	**82.2**	**97.4**

A.9 Comparisons of searched architectures between ViTAS and AutoFormer [4]

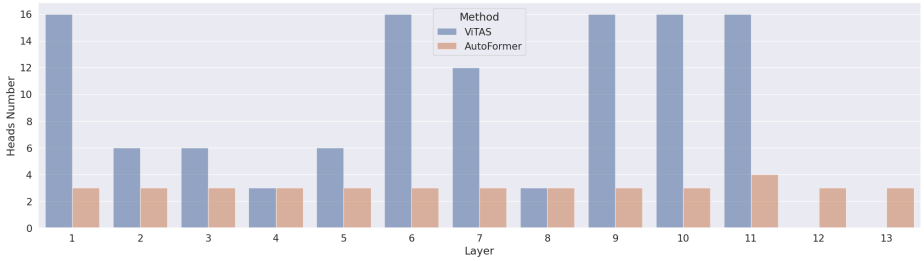
To intuitively check the effect of ViTAS with another baseline method, *i.e.* AutoFormer, we visualize the network width searched by ViTAS and AutoFormer for 1.4G FLOPs DeiT-based architecture in Figure 4. The dimension percentage is computed based on the DeiT-T, *e.g.*, 1.0 means that keep the same channels in the corresponding layer (*i.e.*, patch embedding, attention or MLP) as DeiT-T.

Concretely, in the dimension level (*i.e.*, see Figure 4(a)), ViTAS keeps smaller dimension in the first few layers while a bit more dimensions in the last few layers. Besides, in the first few layers, the searched ViT architecture tend to keep smaller attention dimension than MLP output dimension, while larger attention dimension in the last few layers. We think this may be because attention is performed based on the extracted information of features, which may be more useful after MLP layer extracted enough information from the input. With tight budget (*i.e.*, 1.4G), our searched architecture tend to stack less blocks (*i.e.*, 11) than AutoFormer. In general, AutoFormer keeps almost the same dimensions for all layers as DeiT-T.

When it comes to the heads number, our searched architectures tend to have larger heads number in the first and last few blocks. We think that this may help the architecture to deal with sophisticated information with more heads number in the last few blocks. The AutoFormer keeps almost the same heads number as DeiT-T in all blocks.



(a) Dimension percentage of ViTAS and AutoFormer w.r.t. DeiT small setting.



(b) Heads number of searched DeiT-T based architecutre w.r.t. ViTAS and AutoFormer.

Fig. 4: Visualization of dimensions and heads number of searched DeiT-T based architecture w.r.t. ViTAS and AutoFormer on ImageNet-1k dataset.

Moreover, to promote the fair comparison of ViTAS and AutoFormer, we retrain the released structure of AutoFormer with the same retraining recipe of ours, as shown in Table 15. With the same retraining recipe and same FLOPs budgets, ViTAS achieves 0.8% higher on top-1 accuracy than AutoFormer with 1.3G FLOPs budget DeiT-based architecture, which indicates the effectiveness of our method.

Table 15: Performance comparison with AutoFormer [4] of DeiT tiny based architectures on ImageNet-1k by the same training recipe. $\star$ indicates the re-implementation results of important baseline methods with our recipe. †: we uniformly scale the searched models to the same FLOPs of 1.3G w.r.t. AutoFormer-T.

Method	FLOPs (G)	Throughput (image/s)	Params (M)	Top-1 (%)	Top-5 (%)
DeiT-T* [41]	1.3	2728.5	5	72.3	91.4
AutoFormer-T* [4]	1.3	2955.4	5.7	74.7	91.9
ViTAS-DeiT-A†	1.3	2965.7	6.1	75.5	92.4
ViTAS-DeiT-A	1.4	2831.1	6.6	75.6	92.5

A.10 Implementing the Search with CNN-based Search Space.

To comprehensively check the effect of cyclic weight sharing mechanism w.r.t. CNN based search space, we implement the dimension search with ResNet50 on ImageNet-1k dataset. We leverage the same recipe and search space as [37]. As shown in Table 16, our searched architecture achieves the superior performance w.r.t. baseline methods of DS[10], AutoSlim [60], and BCNet [37], which indicates the effectiveness of the proposed cyclic weight sharing mechanism.

Table 16: Performance comparison of ResNet50 on ImageNet-1k. $\star$ indicates the re-implementation results of important baseline methods with our recipe.

Groups	Methods	FLOPs(G)	Params(M)	accuracy(%)	Groups	Methods	FLOPs(G)	Params(M)	accuracy(%)
1.4G	DS-ResNet-S	1.2	-	74.6	2.4G	DS-ResNet-M	2.2	-	76.1
	AutoSlim*	1.4	15.3	73.8		AutoSlim*	2.4	21.8	75.7
	BCNet*	1.4	16.3	75.4		BCNet*	2.4	22.6	77.0
	ViTAS	1.4	15.7	75.8		ViTAS	2.4	22.1	77.2

[10] The DS-ResNet is from the paper of "Dynamic slimmable network", CVPR 2021 (oral).

A.11 Re-implementation Results of Baseline Methods

To intuitively check the performance of ViTAS, we present the re-implement results w.r.t baseline methods on ImageNet-1k, COCO2017, and ADE20k datasets with our training recipe, as shown in Table 17∼19.

Table 17: Searched Twins-based ViT architectures w.r.t. different FLOPs and GPU throughput on ImageNet-1k. We abbreviate the name of tiny, short, base, and large for T, S, B, and L, respectively. Re-Top-1, Re-Top-5: indicates that the re-implementation results of baseline methods with our recipe. Top-1, Top-5: performance of baseline methods that is reported from papers. Our results are highlighted in bold.

Method	FLOPs (G)	Throughput (image/s)	Params (M)	Top-1 (%)	Top-5 (%)	Re-Top-1 (%)	Re-Top-5 (%)
DeiT-T [41]	1.3	2728.5	5	72.2	91.3	72.3	91.4
Twins-T [5]	1.4	1580.7	11.5	-	-	77.8	94.1
AutoFormer-T [4]	1.3	3055.4	5.7	74.7	92.6	74.7	91.9
ViTAS-Twins-T	1.4	1686.3	13.8	**79.4**	**94.8**	-	-
DeiT-S [41]	4.6	437.0	22.1	79.8	-	79.9	95.0
Twins-SVT-S [5]	2.9	1059	24	81.7	-	81.6	95.9
AutoFormer-S [4]	5.1	1231.7	22.9	81.7	95.7	79.8	95.0
Twins-PCPVT-S [5]	3.8	815	24.1	81.2	-	81.2	95.6
Swin-T [25]	4.5	766	29	81.3	-	81.2	95.5
ViTAS-Twins-S	3.0	958.6	30.5	**82.0**	**95.7**	-	-
Swin-S [25]	8.7	444	50	83.0	-	83.0	96.2
Twins-SVT-B [5]	8.6	469	56	83.2	-	83.2	96.3
ViTAS-Twins-B	8.8	362.7	66.0	**83.5**	**96.5**	-	-
DeiT-B [41]	17.6	292	86.6	81.8	-	81.8	95.7
Twins-SVT-L [5]	15.1	288	99.2	83.7	-	83.6	96.6
ViTAS-Twins-L	16.1	260.7	124.8	**84.0**	**96.9**	-	-

Table 18: Object detection and instance segmentation performance with searched backbones on the COCO2017 dataset with Mask R-CNN framework and RatinaNet framework. We followed the same training and evaluation setting as [5]. "FLOPs" and "Param" are in giga and million, respectively. $\star$ indicates the re-implementation results of important baseline methods with our recipe. Methods without $\star$ indicates that the performance is reported from papers. Since Twins do not provide the experiment results for Twins-SVT-L, we only report our re-implement results as Twins-SVT-L*. Our results are highlighted in bold.

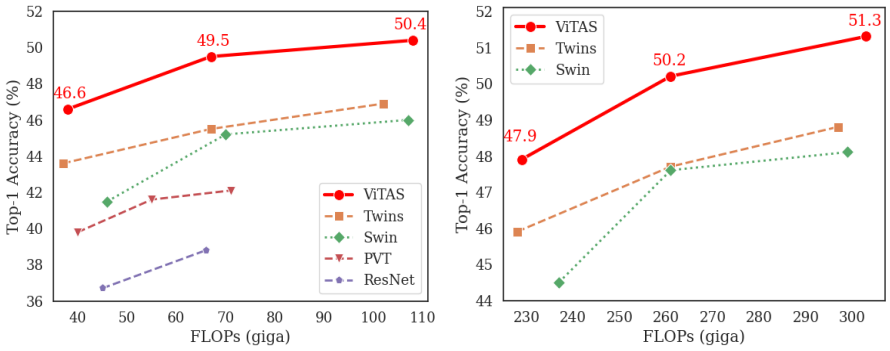
Backbone	Mask R-CNN 1× [16]							RetinaNet 1× [22]							
	FLOPs	Param	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}	FLOPs	Param	AP^b	AP^b_{50}	AP^b_{75}	AP_S	AP_M	AP_L
Twins-SVT-S [5]	164	44.0	43.4	66.0	47.3	40.3	63.2	43.4	104	34.3	43.0	64.2	46.3	28.0	46.4	57.5
Twins-SVT-S*	164	44.0	43.5	66.0	47.8	40.1	62.9	43.1	104	34.3	42.2	63.3	44.9	26.4	45.6	57.0
ViTAS-Twins-S	168	44.2	**45.9**	**67.8**	**50.3**	**41.5**	**64.7**	**45.0**	108	41.3	**44.4**	**65.3**	**47.6**	**27.5**	**48.3**	**60.0**
Twins-SVT-B [5]	224	76.3	45.2	67.6	49.3	41.5	64.5	44.8	163	67.0	45.3	66.7	48.1	28.5	48.9	60.5
Twins-SVT-B*	224	76.3	45.5	67.4	50.0	41.4	64.5	44.5	163	67.0	44.4	65.6	47.4	28.5	47.9	59.5
ViTAS-Twins-B	227	85.4	**47.6**	**69.2**	**52.2**	**42.9**	**66.3**	**46.5**	167	76.2	**46.0**	**66.7**	**49.6**	**29.1**	**50.2**	**62.0**
Twins-SVT-L* [5]	292	119.7	45.9	67.9	49.9	41.6	65.0	45.0	232	110.9	45.2	66.6	48.4	29.0	48.6	60.9
ViTAS-Twins-L	301	144.1	**48.2**	**69.9**	**52.9**	**43.3**	**66.9**	**46.7**	246	135.5	**47.0**	**67.8**	**50.3**	**29.6**	**50.9**	**62.4**

Table 19: Performance comparisons with searched backbones on ADE20K validation dataset. Architectures were implemented with the same training recipe as [5]. All backbones were pretrained on ImageNet-1k, except for SETR, which was pretrained on ImageNet-21k dataset. $\star$ indicates the re-implementation results of important baseline methods with our recipe. Methods without $\star$ indicates that the performance is reported from papers. mAcc: mean accuracy for all categories. aAcc: accuracy of all pixels. Our results are highlighted in bold.

Backbone	Semantic FPN 80k [5]					Upernet 160k [25]				
	FLOPs(G)	Param(M)	mIoU	mAcc	aAcc	FLOPs(G)	Param(M)	mIoU	mAcc	aAcc
Twins-SVT-S [5]	37	28.3	43.2	-	-	228	54.4	46.2	-	-
Twins-SVT-S*	37	28.3	43.6	55.4	80.6	228	54.4	45.9	57.3	81.5
ViTAS-Twins-S	38	35.1	**46.6**	**57.6**	**82.2**	229	61.7	**47.9**	**59.0**	**82.6**
Twin-SVT-B [5]	67	60.4	45.3	-	-	261	88.5	47.7	-	-
Twin-SVT-B*	67	60.4	45.5	57.0	81.5	261	88.5	47.7	59.1	82.7
ViTAS-Twins-B	67	69.6	**49.5**	**60.5**	**83.4**	261	97.7	**50.2**	**61.1**	**83.5**
Twins-SVT-L [5]	102	103.7	46.7	-	-	297	133	48.8	-	-
Twins-SVT-L*	102	103.7	46.9	58.3	81.8	297	133	49.2	60.5	82.8
ViTAS-Twins-L	108	128.2	**50.4**	**61.6**	**83.6**	303	158.7	**51.3**	**61.9**	**84.4**

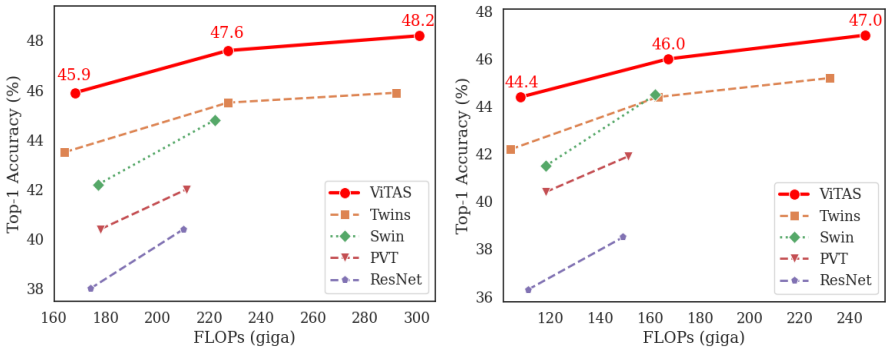
A.12 Comparisons of ViTAS with Baseline Methods w.r.t. Transferability to COCO2017 and ADE20k Datasets

To intuitively compare the transferability of ViTAS with other baseline methods, we evaluated the generalization ability of the ViTAS by transferring the searched architectures to COCO2017 and ADE20k datasets. As shown in Figure 5, we visualize the performance of ViTAS w.r.t. other baseline methods. Indeed, with similar FLOPs budget, our ViTAS achieves the superiority than other baseline methods on the both datasets w.r.t. the tasks of segmentation and detection.



(a) Semantic FPN 80k framework on ADE20k dataset.

(b) Upernet 160k framework on ADE20k dataset.



(c) Mask R-CNN framework on COCO2017 dataset.

(d) RetinaNet framework on COCO2017 dataset.

Fig. 5: The comparisons between ViTAS and other baseline methods with on ADE20k and COCO2017 dataset.

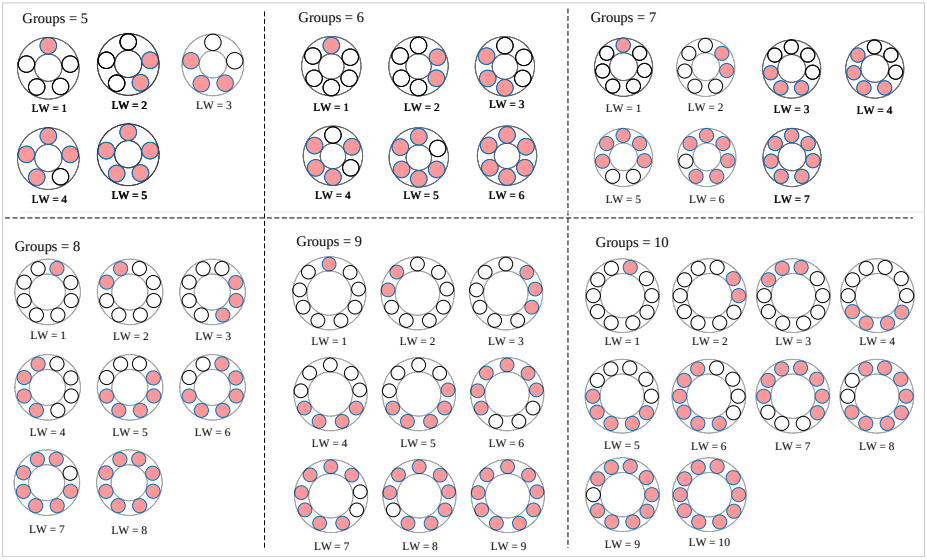


Fig. 6: Visualization of cyclic weight sharing mechanism w.r.t. groups number from 5 to 10.

A.13 Visualization of Cyclic Weight Sharing Mechanism w.r.t. Different Groups Number

In this section, we visualize the channels w.r.t. different groups number in cyclic weight sharing mechanism. Indeed, to promote the usage of cyclic weight sharing mechanism, we constraint the channels to be continuous w.r.t. each group, as illustrated in Figure 6.

Moreover, with Eq. (13), we compare the *influence uniformity* of our cyclic pattern with ordinal and bilateral weight sharing mechanism. As shown in Table 20, our cyclic pattern achieves almost zero influence gap w.r.t. channels and much smaller than others, and bilateral pattern has about the half influence gap than ordinal pattern. It indicates that our ViTAS can fairly train all channels in superformer, and thus can perform better than baseline methods.

Table 20: Comparison of cyclic weight sharing mechanism with other baseline methods w.r.t. different groups number.

methods \ Groups	5	6	7	8	9	10
Ordinal	30.3	48.1	70.6	99.1	130.0	167.4
Bilateral	10.0	19.0	30.6	46.1	66.1	88.7
Cyclic	0.6	0.3	0.7	0.5	0.8	0.5

A.14 Analysis of Weak Augmentation in ViTAS

As in Table 21, we empirically find weak augmentation boost the architecture search of ViTAS. However, in AutoFormer [4], it adopts the same training recipe of superformer as DeiT [41] training from scratch. This is due to the difference with the size of search space. To explore the possibly optimal ViT architecture with no expert manually bias, we leverage a very large search space for ViTAS.

Table 21: Comparison of ViTAS and AutoFormer w.r.t. search space, dimension range, and augmentation type of superformer. "DR" indicates the dimension range of transformer blocks within search space.

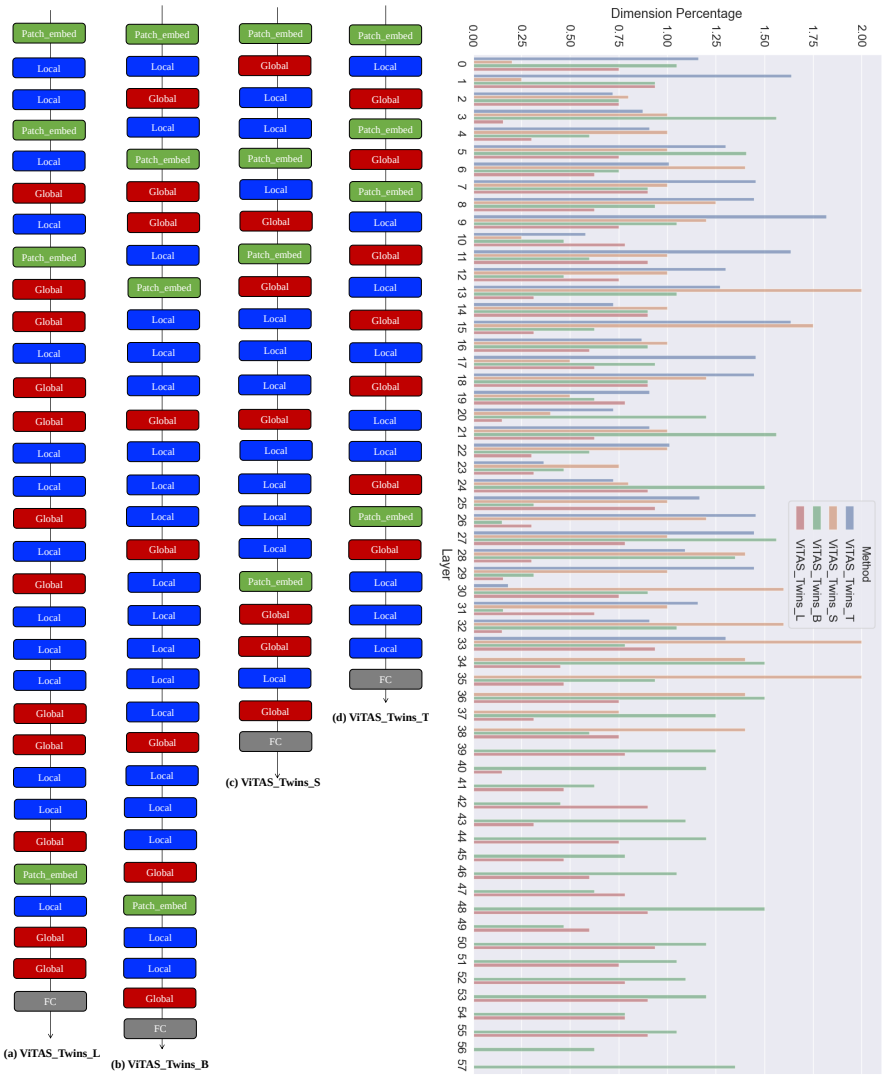
Methods	DeiT-T	DeiT-S	DR of DeiT-T	DR of DeiT-S	Aug type
AutoFormer	4.2×10^{15}	1.2×10^{18}	(192, 256)	(320, 448)	Strong aug
ViTAS	5.4×10^{34}	5.4×10^{34}	(144, 1440)	(288, 2880)	Weak aug

Our ViTAS search space in Table 21 is exponentially ($\times 10^{19}$) larger than that of AutoFormer. As a result, in ViTAS, the superformer is supposed to cover the weights for all the architectures ($> 10^{34}$ paths). Its training thus obtains more challenging when involving stronger augmentations, even affecting the training stability and convergence of superformer. While for AutoFormer, it only includes a manually designed small search space and thus can use a strong augmentation for training the superformer. In addition, the dimension range of the architectures varies greatly ($\times 10$ gap between the largest and smallest dimensions) in ViTAS. While in the AutoFormer, the architectures have very similar dimensions, which boost the training of the superformer while limited the search for the optimal ViT architectures.

A.15 Visualization and Interpretation of Searched ViT Architectures

In this section, we discuss the searched ViT architectures. For intuitively understanding, we visualize our searched three ViT architectures with various FLOPs in Figure 7 as examples. From Figure 7, we summarize the three experiential results for further ViT design. For the last stage of ViT architectures, the downsampling size is equal to the feature size, which indicates that "Local" blocks perform similarly to "Global" blocks.

- The optimal architecture generally tends to follow several local operations after the global blocks.
- The optimal architecture has a bit more local operations than the global operations.
- The dimension between layers changes smaller in Twins-based architectures than in DeiT-based architectures (*i.e.*, see Figure 4), which indicates that the work in [5] performs as a strong baseline w.r.t. the provided ViT architectures.



(a) Visualization of operations w.r.t. searched architectures.

(b) Visualization of dimensions w.r.t. searched architectures.

Fig. 7: Visualization of searched architectures on Twins transformer space.