PyramidTNT: Improved Transformer-in-Transformer Baselines with Pyramid Architecture

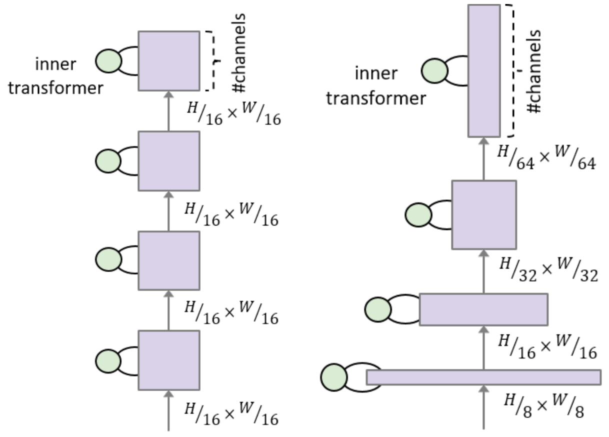
Kai Han, Jianyuan Guo, Yehui Tang, Yunhe Wang
Huawei Noah's Ark Lab
{kai.han,jianyuan.guo,tangyehui,yunhe.wang}@huawei.com

Abstract

Transformer networks have achieved great progress for computer vision tasks. Transformer-in-Transformer (TNT) architecture utilizes inner transformer and outer transformer to extract both local and global representations. In this work, we present new TNT baselines by introducing two advanced designs: 1) pyramid architecture, and 2) convolutional stem. The new "PyramidTNT" significantly improves the original TNT by establishing hierarchical representations. PyramidTNT achieves better performances than the previous state-of-the-art vision transformers such as Swin Transformer. We hope this new baseline will be helpful to the further research and application of vision transformer. Code will be available at https://github.com/huawei-noah/CV-Backbones/tree/master/tnt_pytorch.

1. Introduction

Vision transformer is providing a new type of neural network for computer vision. Starting from ViT [6], a series of works have been proposed to improve the architecture of vision transformer [11, 38, 34, 7, 22, 4, 29]. PVT [34] introduces pyramid network architecture for vision transformer. T2T-ViT-14 [38] recursively aggregates neighboring tokens into one token for extracting local structure and reducing the number of tokens. TNT [11] utilizes inner transformer and outer transformer to model word-level and sentence-level visual representations. Swin Transformer [22] proposes a hierarchical transformer whose representation is computed with Shifted windows. With the recent progress, the performance of vision transformer shows superiority over convolutional neural network (CNN) [9].



(a) TNT

(b) PyramidTNT

Figure 1: Comparison of TNT and PyramidTNT architectures.

This work establishes improved vision transformer baselines based on the TNT [11] framework. Inspired by the recent works [34, 36], we introduce two main architecture modifications: 1) pyramid architecture with gradual decreased resolution to extract multi-scale representations, and 2) convolutional stem for improving the patchify stem and stable training. We also include several other tricks [25, 33] to further improve the efficiency. The new transformer is named as PyramidTNT. The experiments on image classification and object detection demonstrate the superiority of PyramidTNT. Specifically, PyramidTNT-S yields 82.0% ImageNet classification top-1 accuracy with only 3.3B FLOPs, which is significantly better than the original TNT-S [11] and Swin-T [22]. For COCO detection, PyramidTNT-S achieves 42.0 mAP with fewer computational cost than othere transformer and MLP detection models. We hope this new baseline will be helpful to the further research and application of vision transformer.

2. Related Work

Transformer Backbone. Dosovitskiy *et al.* [6] firstly introduce the pure transformer architecture [32] to the vision tasks, which splits the input image into multiple patches and takes each patch as a 'word' in natural language. In

Table 1: Network architectures of PyramidTNT. Three instantiations with different complexity including tiny (Ti), small (S), middle (M) and base (B) versions are presented. The expansion ratio of MLP module is set as 4 by default. H_o and H_i denote the number of heads in outer transformer and inner transformer. R is the reduction ratio of the LSRA.

Stage	Output size	PyramidTNT-Ti		PyramidTNT-S		PyramidTNT-M		PyramidTNT-B	
Stem	$\frac{H}{8} \times \frac{W}{8}$	Conv×5		Conv×5		Conv×5		Conv×5	
		Outer	Inner	Outer	Inner	Outer	Inner	Outer	Inner
Stage 1	$\frac{H}{8} \times \frac{W}{8}$	$\begin{bmatrix} D = 80 \\ H_o = 2 \\ R = 4 \end{bmatrix} \times 2$	$\begin{bmatrix} C = 5 \\ H_i = 1 \\ R = 1 \end{bmatrix} \times 1$	$\begin{bmatrix} D = 128 \\ H_o = 4 \\ R = 4 \end{bmatrix} \times 2$	$\begin{bmatrix} C = 8 \\ H_i = 2 \\ R = 1 \end{bmatrix} \times 1$	$\begin{bmatrix} D = 192 \\ H_o = 4 \\ R = 4 \end{bmatrix} \times 2$	$\begin{bmatrix} C = 12 \\ H_i = 2 \\ R = 1 \end{bmatrix} \times 1$	$\begin{bmatrix} D = 256 \\ H_o = 4 \\ R = 4 \end{bmatrix} \times 2$	$\begin{bmatrix} C = 16 \\ H_i = 2 \\ R = 1 \end{bmatrix} \times 1$
Downsample	$\frac{H}{16} \times \frac{W}{16}$	Patch Merging		Patch Merging		Patch Merging		Patch Merging	
		Outer	Inner	Outer	Inner	Outer	Inner	Outer	Inner
Stage 2	$\frac{H}{16} \times \frac{W}{16}$	$\begin{bmatrix} D = 160 \\ H_o = 4 \\ R = 2 \end{bmatrix} \times 6$	$\begin{bmatrix} C = 10 \\ H_i = 2 \\ R = 1 \end{bmatrix} \times 1$	$\begin{bmatrix} D = 256 \\ H_o = 8 \\ R = 2 \end{bmatrix} \times 8$	$\begin{bmatrix} C = 16 \\ H_i = 4 \\ R = 1 \end{bmatrix} \times 2$	$\begin{bmatrix} D = 384 \\ H_o = 8 \\ R = 2 \end{bmatrix} \times 8$	$\begin{bmatrix} C = 24 \\ H_i = 4 \\ R = 1 \end{bmatrix} \times 2$	$\begin{bmatrix} D = 512 \\ H_o = 8 \\ R = 2 \end{bmatrix} \times 10$	$\begin{bmatrix} C = 32 \\ H_i = 4 \\ R = 1 \end{bmatrix} \times 2$
Downsample	$\frac{H}{32} \times \frac{W}{32}$	Patch Merging		Patch Merging		Patch Merging		Patch Merging	
		Outer	Inner	Outer	Inner	Outer	Inner	Outer	Inner
Stage 3	$\frac{H}{32} \times \frac{W}{32}$	$\begin{bmatrix} D = 320 \\ H_o = 8 \\ R = 1 \end{bmatrix} \times 3$	$\begin{bmatrix} C = 20 \\ H_i = 4 \\ R = 1 \end{bmatrix} \times 1$	$\begin{bmatrix} D = 512 \\ H_o = 16 \\ R = 1 \end{bmatrix} \times 4$	$\begin{bmatrix} C = 32 \\ H_i = 8 \\ R = 1 \end{bmatrix} \times 1$	$\begin{bmatrix} D = 768 \\ H_o = 16 \\ R = 1 \end{bmatrix} \times 6$	$\begin{bmatrix} C = 48 \\ H_i = 8 \\ R = 1 \end{bmatrix} \times 1$	$\begin{bmatrix} D = 1024 \\ H_o = 16 \\ R = 1 \end{bmatrix} \times 6$	$\begin{bmatrix} C = 64 \\ H_i = 8 \\ R = 1 \end{bmatrix} \times 1$
Downsample	$\frac{H}{64} \times \frac{W}{64}$	Patch Merging		Patch Merging		Patch Merging		Patch Merging	
		Outer	Inner	Outer	Inner	Outer	Inner	Outer	Inner
Stage 4	$\frac{H}{64} \times \frac{W}{64}$	$\begin{bmatrix} D = 320 \\ H_o = 8 \\ R = 1 \end{bmatrix} \times 2$	$\begin{bmatrix} C = 20 \\ H_i = 4 \\ R = 1 \end{bmatrix} \times 1$	$\begin{bmatrix} D = 512 \\ H_o = 16 \\ R = 1 \end{bmatrix} \times 2$	$\begin{bmatrix} C = 32 \\ H_i = 8 \\ R = 1 \end{bmatrix} \times 1$	$\begin{bmatrix} D = 768 \\ H_o = 16 \\ R = 1 \end{bmatrix} \times 2$	$\begin{bmatrix} C = 48 \\ H_i = 8 \\ R = 1 \end{bmatrix} \times 1$	$\begin{bmatrix} D = 1024 \\ H_o = 16 \\ R = 1 \end{bmatrix} \times 2$	$\begin{bmatrix} C = 64 \\ H_i = 8 \\ R = 1 \end{bmatrix} \times 1$
Head	1×1	Pooling & FC		Pooling & FC		Pooling & FC		Pooling & FC	
Input resolution		192×192		256×256		256×256		256×256	
Parameters (M)		10.6		32.0		85.0		157.0	
FLOPs (B)		0.6		3.3		8.2		16.0	

[32], extremely large training datasets (*e.g.*, JFT-300M and ImageNet-21k) are usually required for high performance. Touvron *et al.* [31] improve the training recipe and train vision transformers from scratch on ImageNet. Wang *et al.* [34] introduce a hierarchical architecture, which reduce the sequence length of transformer as the network deepens, which can extract the high-level semantic information and reduce the computational cost. Liu *et al.* [22] restrict the self-attention operation in non-overlapping local windows and realize the cross-window connection by shifting these windows. Yuan *et al.* [38] propose a layer-wise tokens-to-token transformation to replace the simple tokenization of input images. Wu *et al.* [35] introduce convolutional projections into vision transformers to bring the desirable properties of CNNs. To capture both global and local information in an image, Han *et al.* [11] present a nested architecture by further dividing each patch into smaller ones, which enhances the representation ability significantly. Notice that this nested design is a general methodology, which can be also combined with the hierarchical architectures (*e.g.*, [34, 22]) for further improving performance.

MLP Backbone. Tolstikhin *et al.* [30] construct a MLP-Mixer model by only stacking multi-layer perceptrons (MLPs), showing that neither convolutions and attention are necessary for good performance. Channel-mixing and token-mixing MLPs are two core blocks, which extract features of each token (patch) and aggregate information from different tokens, respectively. Recently, various variants are developed to achieve a better trade-off between accuracy and computational cost. For example, shift operation is introduced in S^2-MLP [37] and AS-MLP [18] to exchange information across different tokens. Hire-MLP [8] present a hierarchical rearrangement operation, where the inner-region rearrangement and cross-region rearrangement capture local information and global context, respectively. Wave-MLP [28] takes each tokens as a wave and model the relationship between different tokens by considering their amplitude and phase information simultaneously.

3. Method

Convolutional Stem. Given a input image $X \in \mathbb{R}^{H \times W}$, vanilla TNT model first split the image into a number of patches and further view each patch as a sequence of sub-patches. A linear layer is applied to project the sub-patch into a visual word vector (a.k.a., token). These visual words are concatenated and transformed into a visual sentence vector. Xiao *et al.* [36] find that using several convolutions as stem in ViT increases optimization stability and also improves the performance. Based on the observation, we construct a convolutional stem for PyramidTNT. A stack of 3×3 convolutions is utilized to produce visual words $Y \in \mathbb{R}^{\frac{H}{2} \times \frac{W}{2} \times C}$ where C is the visual word dimension. Similarly, we can obtain visual sentences $Z \in \mathbb{R}^{\frac{H}{8} \times \frac{W}{8} \times D}$ where D is the visual sentence dimension. The word-level and sentence-level position encodings are added on visual words and sentences respectively, as in the original TNT [11].

Pyramid Architecture. The original TNT network maintains the same number of tokens in every block, following ViT [6]. The numbers of visual words and visual sentences are kept unchanged from bottom to top. Inspired by PVT [34], we construct four stages with different number of tokens for TNT, as shown in Figure 1 (b). For the four stages, the spatial shape of visual words are set as $\frac{H}{2} \times \frac{W}{2}$, $\frac{H}{4} \times \frac{W}{4}$, $\frac{H}{8} \times \frac{W}{8}$, and $\frac{H}{16} \times \frac{W}{16}$. The spatial shape of visual sentences are set as $\frac{H}{8} \times \frac{W}{8}$, $\frac{H}{16} \times \frac{W}{16}$, $\frac{H}{32} \times \frac{W}{32}$, and $\frac{H}{64} \times \frac{W}{64}$. The downsample operation is implemented using a convolution with stride 2. Each stage is composed by several TNT blocks, and the TNT block operates on word-level and sentence-level features as described in [11]. Finally, the output visual sentences are fused into a vector as the image representation using the global average pooling operation.

Other Tricks. In apart from the network architecture modification, several advanced tricks for vision transformer are also adopted. Relative position encoding [25] is added on self-attention module to better represent relative position between tokens. Linear spatial reduction attention (LSRA) [33] is utilized in the first two stages to reduce the computation cost of self-attention for long sequence.

4. Experiment

4.1. Image Classification

Table 2: Training hyperparameters for ImageNet-1K.

PyramidTNT	Ti	S	M	B
Epochs	300			
Batch size	1024			
Optimizer	AdamW [23]			
Start learning rate (LR)	1e-3			
LR decay	Cosine			
Warmup epochs	20			
Weight decay	0.05			
Label smoothing [26]	0.1			
Drop path [17]	0.1	0.1	0.15	0.3
Repeated augment [15]	✓			
RandAugment [5]	✓			
Mixup prob. [40]	0.8			
Cutmix prob. [39]	1.0			
Erasing prob. [41]	0.25			
Exponential moving average	0.99996			

Settings. We conduct image classification experiments on the large-scale ImageNet-1K dataset [24]. ImageNet-1K consists of about 1.28M training images and 50K validation images belonging to 1,000 classes. We utilize the same training strategy as in DeiT [31] and TNT [11], as described in Table 2. All PyramidTNT models are implemented using PyTorch and trained on 8 NVIDIA V100 GPUs.

Table 3: ImageNet-1K classification results of representative CNN, MLP and transformer models. Following [31, 22], the throughput is measured on an NVIDIA V100 GPU and PyTorch.

Model	Params (M)	FLOPs (B)	Throughput (image/s)	Top-1 (%)
CNN				
ResNet-50 [13, 38]	25.6	4.1	1226	79.1
ResNet-101 [13, 38]	44.7	7.9	753	79.9
ResNet-152 [13, 38]	60.2	11.5	526	80.8
EfficientNet-B0 [27]	5.3	0.39	2694	77.1
EfficientNet-B1 [27]	7.8	0.7	1662	79.1
EfficientNet-B2 [27]	9.2	1.0	1255	80.1
EfficientNet-B3 [27]	12	1.8	732	81.6
EfficientNet-B4 [27]	19	4.2	349	82.9
S-GhostNet-B1 [21, 10]	16.2	0.67	-	80.9
S-GhostNet-B4 [21, 10]	32.9	4.4	-	84.3
MLP				
AS-MLP-T [18]	28	4.4	862	81.3
AS-MLP-S [18]	50	8.5	473	83.1
AS-MLP-B [18]	88	15.2	308	83.3
CycleMLP-B2 [3]	27	3.9	635	81.6
CycleMLP-B3 [3]	38	6.9	371	82.4
CycleMLP-B4 [3]	52	10.1	259	83.0
Hire-MLP-Small [8]	33	4.2	807	82.1
Hire-MLP-Base [8]	58	8.1	441	83.2
Hire-MLP-Large [8]	96	13.4	290	83.8
Wave-MLP-T [28]	17	2.4	1208	80.6
Wave-MLP-S [28]	30	4.5	720	82.6
Wave-MLP-M [28]	44	7.9	413	83.4
Transformer				
DeiT-Ti [6, 31]	5	1.3	2536	72.2
DeiT-S [6, 31]	22	4.6	940	79.8
DeiT-B [6, 31]	86	17.6	292	81.8
T2T-ViT-14 [38]	21.5	5.2	-	81.5
T2T-ViT-19 [38]	39.2	8.9	-	81.9
T2T-ViT-24 [38]	64.1	14.1	-	82.3
PVT-Small [34]	24.5	3.8	820	79.8
PVT-Medium [34]	44.2	6.7	526	81.2
PVT-Large [34]	61.4	9.8	367	81.7
PVTv2-B0 [33]	3.4	0.6	-	70.5
PVTv2-B2 [33]	25.4	4.0	-	82.0
PVTv2-B4 [33]	62.6	10.1	-	83.6
Swin-T [22]	29	4.5	755	81.3
Swin-S [22]	50	8.7	437	83.0
Swin-B [22]	88	15.4	278	83.3
TNT-S [11]	23.8	5.2	428	81.5
TNT-S-2 [11]	22.4	4.7	704	81.4
TNT-B [11]	65.6	14.1	246	82.9
PyramidTNT-Ti	**10.6**	**0.6**	**2423**	**75.2**
PyramidTNT-S	**32.0**	**3.3**	**721**	**82.0**
PyramidTNT-M	**85.0**	**8.2**	**413**	**83.5**
PyramidTNT-B	**157.0**	**16.0**	**263**	**84.1**

Results. We show the ImageNet-1K classification results in Table 3. Compared to the original TNT, PyramidTNT

Table 4: Object detection and instance segmentation results on COCO val2017. We compare the proposed PyramidTNT-S and PyramidTNT-M with other backbones based on RetinaNet [19] and Mask R-CNN [12] frameworks, all models are trained in "1x" schedule. FLOPs is calculated on 1280×800 input.

Backbone	RetinaNet 1×						Mask R-CNN 1×						
	# FLOPs	AP	AP_{50}	AP_S	AP_M	AP_L	# FLOPs	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
ResNet50 [13]	239.3G	36.3	55.3	19.3	40.0	48.8	260.1G	38.0	58.6	41.4	34.4	55.1	36.7
PVT-Small [34]	226.5G	40.4	61.3	25.0	42.9	55.7	245.1G	40.4	62.9	43.8	37.8	60.1	40.3
CycleMLP-B2 [3]	230.9G	40.6	61.4	22.9	44.4	54.5	249.5G	42.1	64.0	45.7	38.9	61.2	41.8
Swin-T [22]	244.8G	41.5	62.1	25.1	44.9	55.5	264.0G	42.2	64.6	46.2	39.1	61.6	42.0
Hire-MLP-Small [8]	237.6G	41.7	-	**25.3**	**45.4**	54.6	256.2G	42.8	65.0	46.7	39.3	62.0	42.1
PyramidTNT-S	225.9G	**42.0**	**63.1**	25.0	44.9	**57.7**	255.9G	**43.4**	**65.3**	**47.3**	**39.5**	**62.3**	**42.2**

Table 5: Instance segmentation results on COCO val2017. Mask R-CNN [12] and Cascade Mask R-CNN [1] are trained in "3x" schedule with multi-scale strategy.

Backbone	Mask R-CNN 3×							Cascade Mask R-CNN 3×						
	# FLOPs	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}	# FLOPs	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
ResNet50 [13]	260.1G	41.0	61.7	44.9	37.1	58.4	40.1	738.7G	46.3	64.3	50.5	40.1	61.7	43.4
AS-MLP-T [18]	260.1G	46.0	67.5	50.7	41.5	64.6	44.5	739.0G	50.1	68.8	54.3	43.5	66.3	46.9
Swin-T [22]	264.0G	46.0	68.2	50.2	41.6	65.1	44.8	742.4G	50.5	69.3	54.9	43.7	66.6	47.1
Hire-MLP-S [8]	256.2G	46.2	68.2	50.9	42.0	65.6	45.3	734.6G	50.7	69.4	55.1	**44.2**	66.9	**48.1**
PyramidTNT-S	255.9G	**47.1**	**68.9**	**51.6**	**42.2**	**65.8**	**45.4**	794.1G	**51.0**	**69.7**	**55.3**	**44.2**	**67.0**	**48.1**

achieves much better image classification accuracy. For instance, top-1 accuracy of PyramidTNT-S is 0.5% higher by using 1.9B fewer FLOPs compared to TNT-S. We also compare PyramidTNT with other representative CNN, MLP and transformer based models. From the results, we can see that PyramidTNT is the state-of-the-art vision transformer.

4.2. Object Detection

Settings. The object detection and instance segmentation experiments are conducted on challenging COCO 2017 benchmark [20], which contains 118K training images and 5K validation images. Following PVT [34] and Swin Transformer [22], we consider three typical object detection frameworks: RetinaNet [19], Mask R-CNN [12] and Cascade Mask R-CNN [1] in mmdetection [2]. Noted that the four spatial shapes of our PyramidTNT are set as $\frac{H}{8} \times \frac{W}{8}$, $\frac{H}{16} \times \frac{W}{16}$, $\frac{H}{32} \times \frac{W}{32}$, and $\frac{H}{64} \times \frac{W}{64}$, in contrast to the multi-scale feature maps produced by typical backbones. To address this discrepancy, we employ four simple upsample layers consisted of a stride-two 2×2 transposed convolution, followed by batch normalization [16] and GeLU [14], and a stride-one 3×3 convolution, followed by another batch normalization and GeLU. Therefore our PyramidTNT can generate feature maps with strides of 4, 8, 16, and 32 pixels, w.r.t. the input image.

In order to compare with PVT [34], CycleMLP [3] and Hire-MLP [8], we conduct experiments based on RetinaNet [19] and Mask R-CNN [12]. We use AdamW optimizer with a batch size of 2 images per GPU, the initial learning rate is set to 1e-4 and divided by 10 at the 8th and the 11th epoch. The weight decay is set to 0.05. All models are trained in "1x" schedule (*i.e.*, 12 epochs), with single-scale strategy on 8 Tesla V100 GPUs. The input image is resized such that its shorter side has 800 pixels while its longer side does not exceed 1333 pixels during training.

In addition, we adopt another setting following [22, 18, 8], *i.e.*, multi-scale training strategy and "3x" schedule, based on Mask R-CNN [12] and Cascade Mask R-CNN [1]. During training, the input image is resized such that its shorter side is between 480 and 800 pixels while its longer side does not exceed 1333. In the testing phase, the shorter side of the input image is fixed to 800 pixels. We also use AdamW optimizer with batch size 16 on 8 Tesla V100 GPUs. The initial learning rate is set to 1e-4 and divided by 10 at the 27th and the 33rd epoch.

Results. Table 4 reports the results of object detection and instance segmentation under "1x" training schedule. PyramidTNT-S significantly outperforms other backbones on both one-stage and two-stage detectors with similar computational cost. For example, PyramidTNT-S based RetinaNet archive 42.0 AP and 57.7 AP_L, surpassing the models with Swin Transformer [22] by 0.5 AP and 2.2 AP_L, respectively. These results indicate that the pyramid architecture of TNT can help capture better global information for large objects. We conjecture that the simple upsample strategy and smaller spatial shape of PyramidTNT withhold the AP_S from a large promotion.

We also report the detection results under multi-scale strategy and "3x" training schedule in Table 5.

PyramidTNT-S can obtain much better AP^b and AP^m than all other counterparts on Mask R-CNN [12] and Cascade Mask R-CNN [1], showing its better feature representation ability. For example, PyramidTNT-S surpasses Hire-MLP-S [8] by 0.9 AP^b on Mask R-CNN with less FLOPs constrain.

References

[1] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018. 4, 5

[2] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, et al. Mmdetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 4

[3] Shoufa Chen, Enze Xie, Chongjian Ge, Ding Liang, and Ping Luo. Cyclemlp: A mlp-like architecture for dense prediction. *arXiv preprint arXiv:2107.10224*, 2021. 3, 4

[4] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. In *NeurIPS*, 2021. 1

[5] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR Workshops*, 2020. 3

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 1, 3

[7] Ben Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *ICCV*, 2021. 1

[8] Jianyuan Guo, Yehui Tang, Kai Han, Xinghao Chen, Han Wu, Chao Xu, Chang Xu, and Yunhe Wang. Hire-mlp: Vision mlp via hierarchical rearrangement. *arXiv preprint arXiv:2108.13341*, 2021. 2, 3, 4, 5

[9] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer. *arXiv preprint arXiv:2012.12556*, 2020. 1

[10] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *CVPR*, pages 1580–1589, 2020. 3

[11] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. In *NeurIPS*, 2021. 1, 2, 3

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, 2017. 4, 5

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 3, 4

[14] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv:1606.08415*, 2016. 4

[15] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *CVPR*, 2020. 3

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 4

[17] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016. 3

[18] Dongze Lian, Zehao Yu, Xing Sun, and Shenghua Gao. As-mlp: An axial shifted mlp architecture for vision. *arXiv preprint arXiv:2107.08391*, 2021. 2, 3, 4

[19] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. 4

[20] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755, 2014. 4

[21] Chuanjian Liu, Kai Han, An Xiao, Yiping Deng, Wei Zhang, Chunjing Xu, and Yunhe Wang. Greedy network enlarging. *arXiv preprint arXiv:2108.00177*, 2021. 3

[22] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021. 1, 2, 3, 4

[23] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 3

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 3

[25] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018. 1, 3

[26] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 3

[27] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 3

[28] Yehui Tang, Kai Han, Jianyuan Guo, Chang Xu, Yanxi Li, Chao Xu, and Yunhe Wang. An image patch is a wave: Phase-aware vision mlp. *arXiv preprint arXiv:2111.12294*, 2021. 2, 3

[29] Yehui Tang, Kai Han, Chang Xu, An Xiao, Yiping Deng, Chao Xu, and Yunhe Wang. Augmented shortcuts for vision transformers. In *NeurIPS*, 2021. 1

[30] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. Mlp-mixer: An all-mlp architecture for vision. *arXiv:2105.01601*, 2021. 2

[31] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2020. 2, 3

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 1, 2

[33] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pvtv2: Improved baselines with pyramid vision transformer. *arXiv preprint arXiv:2106.13797*, 2021. 1, 3

[34] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *ICCV*, 2021. 1, 2, 3, 4

[35] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. *arXiv:2103.15808*, 2021. 2

[36] Tete Xiao, Piotr Dollar, Mannat Singh, Eric Mintun, Trevor Darrell, and Ross Girshick. Early convolutions help transformers see better. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. 1, 2

[37] Tan Yu, Xu Li, Yunfeng Cai, Mingming Sun, and Ping Li. S2-mlp: Spatial-shift mlp architecture for vision. *arXiv preprint arXiv:2106.07477*, 2021. 2

[38] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *ICCV*, 2021. 1, 2, 3

[39] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019. 3

[40] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 3

[41] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, volume 34, pages 13001–13008, 2020. 3