Three things everyone should know about Vision Transformers

Hugo Touvron[⋆,†] Matthieu Cord[†] Alaaeldin El-Nouby[⋆,◇]
Jakob Verbeek[⋆] Hervé Jégou[⋆]

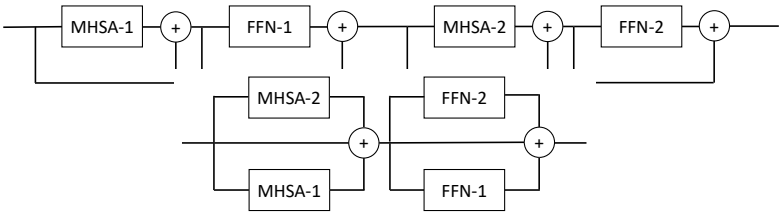
[⋆]Meta AI [†]Sorbonne University [◇]Inria

Abstract

After their initial success in natural language processing, transformer architectures have rapidly gained traction in computer vision, providing state-of-the-art results for tasks such as image classification, detection, segmentation, and video analysis. We offer three insights based on simple and easy to implement variants of vision transformers. (1) The residual layers of vision transformers, which are usually processed sequentially, can to some extent be processed efficiently in parallel without noticeably affecting the accuracy. (2) Fine-tuning the weights of the attention layers is sufficient to adapt vision transformers to a higher resolution and to other classification tasks. This saves compute, reduces the peak memory consumption at fine-tuning time, and allows sharing the majority of weights across tasks. (3) Adding MLP-based patch pre-processing layers improves Bert-like self-supervised training based on patch masking. We evaluate the impact of these design choices using the ImageNet-1k dataset, and confirm our findings on the ImageNet-v2 test set. Transfer performance is measured across six smaller datasets.

1 Introduction

Since its introduction the Transformer architecture [66] has become the dominant architecture in natural language processing tasks, replacing previously popular recurrent architectures. The vision transformer [16] (ViT) is a simple adaptation of transformers to computer vision tasks like image classification: the input image is divided into non-overlapping patches, which are fed to a vanilla transformer architecture, after a linear patch projection layer. In contrast to networks built from convolutional layers, transformers offer parallel processing and a complete field-of-view in a single layer. Along with other attention-based architectures, see e.g. [4, 7], transformers have recently substantially influenced the design of computer vision architectures. Many modern architectures in computer vision directly inherit parts of their design from this work, or are at least inspired by the recent findings resulting from transformers [7, 16, 62]. As a result, significant improvements have been observed on different computer vision tasks, ranging from ob-



ject detection and segmentation [18] and video analysis [1, 19] to image generation [9, 31].

While vision transformers have led to considerable progress, the optimization of their design and training procedures have only been explored to a limited extent. In this paper, we offer three insights on training vision transformers.

1. Parallel vision transformers. Several works [20, 75] advocate the interest shallower networks for reasons ranging from lower latency to easier optimization. We propose a very simple way to achieve this with ViTs. Let us denote by MHSA the multi-headed self-attention residual block, and by FFN the residual feedforward network. Starting from a sequential architecture depicted as follows,
we parallelize the architecture by reorganizing the same blocks by pairs,
which can be done for any different numbers of parallel blocks. This produces an architecture with the same number of parameters and compute, while being wider and shallower. This design allows for more parallel processing, easing optimization and reducing latency depending on the implementation.

In Section 3, we experimentally analyse the performance of this parallel construction, and in particular how it affects the accuracy in comparison to the sequential baseline. The parallel version becomes a compelling option if deep enough. In some cases, we observe improvements in accuracy resulting from an easier optimization. Regarding the latency on GPUs, we observe reductions in the case of small batch sizes.[1]

2. Fine-tuning attention is all you need. It is common practice to pre-train networks before fine-tuning them on a target task. This is the standard approach underpinning transfer learning, where one leverages a large generic dataset like ImageNet [56] when the number of images is limited for the target task [50, 73]. Another context is the one of changing resolution. Typically one would train at a lower resolution than the one employed at inference time. This saves resources, but additionally it reduces the discrepancy of scale between train and test images that results from data augmentation [65]. In Section 4 we show that, in the case of ViT, it is mostly sufficient to fine-tune only the multi-head attention layers and freeze the feedforward network (FFN) layers. This saves compute and reduces the memory peak during training. Importantly this allows the same FFN weights, which dominate the number of parameters, to be used for multiple tasks. The impact on accuracy is statistically not significant when fine-tuning for different image resolutions. For large models, the impact on accuracy is limited when considering transfer to other classification tasks.

3. Patch preprocessing with masked self-supervised learning. The first layers of a transformer have a relatively local span [11], suggesting that they mostly be-

[1] We have not found any papers in the literature analyzing the effect of width versus depth for ViT on common GPUs and CPUs.

have like convolutions. Some recent hybrid architectures [18, 21, 23] preprocess their input images with a convolutional stem, to improve accuracy and training stability [71]. However, preprocessing images with convolutions is *a priori* not compatible with the recent and successful mask-based self-supervised learning approaches, like BeiT [3] or MAE [24]. The convolutions propagate information across patches, impeding the masked prediction task.

In Section 5, we propose a simple way to adapt mask-based self-supervised training methods with patch pre-processing, by applying the masking after the patch pre-processing. However, our analysis reveals that existing convolutional stems are not effective when combined with BeiT. To address this issue, we introduce a hierarchical MLP (hMLP) stem that interleaves MLP layers and patch aggregation operations, and prohibits any communication between patches. Our experiments show that this choice is effective and able to leverage the benefit of both BeiT self-supervised pre-training and patch pre-processing. Moreover, our hMLP-stem is also effective for ViT in the supervised case: it is on par with the best convolutional stem of our comparison [21].

2 Background

In this section, we discuss related work in common with our different contributions. We also introduce the baseline ViT models considered in this study and how they are trained. In subsequent sections, we discuss related work that is more specific to each of our three specific contributions.

2.1 Related work

Attention-based models, and in particular transformers [66], have been rapidly adopted in neural networks handling text [6, 12, 40, 52, 66], speech [33, 44], and even for more complex tasks such as function integration or solving differential equation [37]. In computer vision, DeTR [7] and Vision Transformers [16] (ViT) have deeply influenced the design of architectures in a short period of time. Most of the architectures introduced since ViT can be regarded as some form of hybridisation of transformers with convolutional neural networks, as illustrated by the hierarchical transformers [19, 21, 41, 67], or conversely by convolutional neural networks with design elements inspired from ViT [42, 63], or even multi-layer perceptrons adopting designs inspired by transformers [14, 39, 47, 60, 61].

In our case we build upon the basic ViT design of Dosovitskiy. Its design is governed by a small hyper-parameter space, and as such is less engineered than some recent follow-up architectures. With a proper training procedure [58, 62, 69], it achieves interesting performance/complexity trade-offs. It is also versatile: it can be effectively combined with hierarchical detection or segmentation frameworks [18]. Importantly, in spite of limited built-in priors, it has demonstrated great potential when combined with self-supervised learning, either with contrastive methods [8, 10] or for reconstruction-based techniques like BeiT [3] or other forms of masked auto-encoders [15, 17, 24, 68, 72, 76].

Table 1: Baseline models and their performance on ImageNet1k-val top1 accuracy at resolution 224×224. We adopt common models with their default parametrization: Vit-B and Vit-L [16] and Vit-Ti and ViT-S [62], all with patch size of 16×16. Baseline results trained with LayerScale or not (LS) [64], and for 300 or 400 epochs of training.

Model	depth	width	heads	params ($\times 10^6$)	Flops ($\times 10^9$)	speed (im/s)	300 epochs val	300 epochs v2	400 ep.+LS val	400 ep.+LS v2
ViT-Ti/16	12	192	3	5.7	1.3	3796	72.7	60.3	73.5	61.4
ViT-S/16	12	384	6	22.1	4.6	1827	79.7	68.5	80.7	69.3
ViT-B/16	12	768	12	86.6	17.6	799	82.2	71.2	82.7	72.2
ViT-L/16	24	1024	16	304.4	61.6	277	83.0	72.4	84.0	73.7

2.2 Experimental setting

ViT models. We consider the vanilla ViT models initially introduced by Dosovitskiy et al. [16] as well as the smaller ones proposed by Touvron et al. [62]. Therefore we use the initial pooling method that is based on a so-called class token. We only consider transformers operating on 16×16 patches. Decreasing this patch size improves the results but significantly increases the model complexity.

Training procedure. To prevent overfitting, we adopt an existing training setting, namely the A2 procedure of Wightman et al. [69]. It uses a binary cross entropy loss and fixes the setting of most of the hyper-parameters.Wightman et al.'s A2 procedure was originally designed for training ResNet-50 models, and requires a few modifications when adopting it for ViTs to get strong performance and ensure sufficient stability:

- *The learning rate* should be reduced compared to ResNet-50. We set it to $lr = 4.10^{-3}$ for ViT-Ti and ViT-S and to $lr = 3.10^{-3}$ for ViT-B and ViT-L.
- *Stochastic depth drop-rate sd:* we adjust it per model following Touvron et al. [64]. It is not used for ViT-Ti. We fix $sd = 0.05$ for Vit-S, $sd = 0.1$ for ViT-B and $sd = 0.4$ for Vit-L.

We observe that LayerScale [64] significantly improves the performance when training large models, and that in that case a longer training is also beneficial. Therefore in addition to our main baseline where we train during 300 epochs without LayerScale, like in DeiT and in the A2 procedure of Wightman et al. [69], we consider another one that is trained for 400 epochs with LayerScale (LS).

Evaluation. Unless specified otherwise, we train our models on the ImageNet-1k dataset [56], and evaluate the top-1 accuracy on its validation set. All experiments are carried with seed 0. Since we have adjusted a low number of hyper-parameters, and since we share them across models except stochastic depth, we do not expect much overfitting. Nevertheless we also evaluate our models with the same metric on ImageNet-V2 [55] (matched frequency), which provides a separate test set, to provide a complementary view on the results.

2.3 Baselines

We report the results of our baseline in Table 1. With the few adaptations that we have done, our training procedure outperforms existing ones for supervised training for the model sizes that we consider, see Appendix A (Table 8). Note that all our models use a patch size of 16×16 as in Dosovitskiy et al. [16]. Unless specified, our experiments are carried out with images of size 224×224.

3 Depth vs Width: Parallel ViT

A recurrent debate in neural architecture design is on how to balance width versus depth. The first successful neural networks on Imagenet [35, 57] were not very deep, for instance the 22-layer GoogleNet [59] was regarded as deep in 2014's standards. This has changed with ResNets [25, 26], for which going deeper was hindering significantly less the optimization due to the residual connections. After its introduction, some researchers have investigated alternative choices for trading depth against width [13, 30, 75], like Wide Residual Networks [75].

Recently, there has been a renewed interest for wider architectures with attention [20, 38]. For instance the Non-deep Networks [20] proposes an architecture with several parallel branches whose design is more complex. In our work, we aim at proposing a much simpler and flexible alternative that builds upon a regular ViT in a more straightforward manner.

3.1 Preliminary discussion on width versus depth for ViT

The ViT architecture of Dosovitskiy et al. [16] is parametrized by three quantities: the width (i.e., the working dimensionality d), the depth, and the number of heads. We do not discuss the latter. Increasing depth or width increases the capacity of the model and usually its accuracy. For the most common ViT models that we report in Table 1 [16, 62], width and height are scaled together. Below, we discuss the different pros and cons for favoring width versus depth.

Parametrization & Optimization. The compositionality of the layers is better with deeper networks. This was one of the decisive advantage of ResNet once optimization issues were solved by residual connections. Yet too much depth hinders optimization, even with residual connections. Some solutions have been proposed to address this issue for ViTs [64], showing that transformers benefit from depth when trained with improved optimization procedure.

Separability. In image classification, the spatial features are ultimately projected [35] or pooled [25] into a high-dimensional latent vector that is subsequently fed to a linear classifier. The dimensionality of this vector should be high enough so that the classes are linearly separable. Hence it is typically larger for tasks involving many classes. For instance in ResNet-50 it has dimension 512 when applied to CIFAR, but 2048 for ImageNet. In ViT, the width is identical to the working dimensionality of each patch, and is typically smaller than with ResNet, possibly limiting the separation capabilities. Besides, a larger dimension of the latent vector tend to favor overfitting. In this regard the compromise between capacity and overfitting is subtle and depends size of the training set [58].

Complexity. In ViT, the different complexity measures are affected differently by width and depth. Ignoring the patch pre-processing and final classification layer, which contribute to complexity in a negligible manner, then we have:

- *The number of parameters* is proportional to depth and a quadratic function of the width.
- *The compute,* as determined by FLOPS, is similarly proportional to the depth and quadratic in width.
- *The peak memory usage at inference time* is constant when increasing the depth for a fixed width, but it is quadratic as a function of width.
- *The latency* of wide architectures is in theory better as they are more parallel, but actual speedups depend on implementation and hardware.

3.2 Parallelizing ViT

We propose and analyze flattening vision transformers by grouping layers following the scheme presented in the introduction. Let us consider a sequence of transformer blocks defined by the functions $\mathrm{mhsa}_l(\cdot)$, $\mathrm{ffn}_l(\cdot)$, $\mathrm{mhsa}_{l+1}(\cdot)$ and $\mathrm{ffn}_{l+1}(\cdot)$. Instead of sequentially processing the input x_l in four steps as done in the usual implementation:

$$\begin{aligned} x'_{l+1} &= x_l + \mathrm{mhsa}_l(x_l), & x_{l+1} &= x'_{l+1} + \mathrm{ffn}_l(x'_{l+1}), \\ x'_{l+2} &= x_{l+1} + \mathrm{mhsa}_{l+1}(x_{l+1}), & x_{l+2} &= x'_{l+2} + \mathrm{ffn}_{l+1}(x'_{l+2}), \end{aligned} \tag{1}$$

we replace this composition by two parallel operations:

$$\begin{aligned} x_{l+1} &= x_l + \mathrm{mhsa}_{l,1}(x_l) + \mathrm{mhsa}_{l,2}(x_l), \\ x_{l+2} &= x_{l+1} + \mathrm{ffn}_{l,1}(x_{l+1}) + \mathrm{ffn}_{l,2}(x_{l+1}). \end{aligned} \tag{2}$$

This reduces the number of layers by two for a given number of MHSA and FFN blocks. Conversely, there is twice the amount of processing in parallel. The intuition behind this parallelization is as follows: as networks get deeper, the contribution of any residual block $r(\cdot)$, be it $\mathrm{mhsa}(\cdot)$ or $\mathrm{ffn}(\cdot)$, becomes increasingly smaller with respect to the overall function. Therefore, the approximation $\forall r, r' \quad r'(x + r(x)) \approx r'(x)$ becomes increasingly satisfactory, and it is easy to check that if this approximation is true, eq. (1) and (2) are equivalent.

Our strategy is different from taking transformers with a larger working dimensionality, which leads to different trade-offs between accuracy, parameters, memory and FLOPS, as discussed in our experiments. In contrast to increasing the working dimension, which increases the complexity quadratically as discussed above, our modification is neutral with respect to parameter and compute.

Depending on whether we effectively parallelize the processing, the peak memory usage at inference time and the latency are modified. Note that rather than just two, we can choose to process any number of blocks in parallel; falling back to the sequential design if we process a single block in each layer.

3.3 Experiments

Notation. We adopt the standard naming convention of previous work [16, 62] to use the postfixes Ti/S/B/L to identify the working dimensionality of the models, i.e., the column "width" in Table 1. We append the depth N to indicate variations on the number of pairs of layers (MHSA,FFN) [64]. For instance, ViT-B24 has the same width as a ViT-B12 but with twice the depth, i.e., 24 pairs of MHSA and FFN layers instead of 12. For our parallel models, we specify both the depth and the number of parallel branches: ViT-B12×2 has twice the number of residual modules as a ViT-B12. It includes a total of 12×2=24 pairs of MHSA and FFN layers. Therefore it has the same complexity as the ViT-B24 model (a.k.a. ViT-B24×1).

Comparison of sequential and parallel ViTs. In Figure 1, we compare the performance of sequential and parallel models of a fixed complexity. We fix the total number of blocks, i.e. pairs of MHSA and FFN layers, which determines the number of parameters and FLOPS, and we consider different possible of branches that leads to the same total number of blocks. For instance 36 can be obtained as the sequential ViT 36×1, or the parallel ViTs 18×2, 12×3 or 9×4.

We observe that, amongst the parallel and sequential models, the best performance is obtained with two parallel branches for all tested model capacities. The performance is comparable between the S20×3 and S30×2 for ViT-S60, but generally using more than two parallel branches is not favorable in terms of accuracy and we do not discuss them further. Note that Figure 1 compares ViT models with a relatively large number of blocks (36 and 60). This is the case where sequential models are relatively difficult to optimize due to their depth. The parallel models with two branches are easier to train, while being deep enough to benefit from layer compositionality.

In Figure 2, we consider models with only 24 pairs (MHSA,FFN) and a varying width. Here we observe that the smallest models ViT-Ti and ViT-S are better in their sequential version. This is because are easy to optimize up to 24 layers. The B24×1 and B12×2 achieve comparable performance. In contrast, the ViT-L12×2 is stronger than its sequential counterpart, which is more difficult to optimize even though we used LS for this size; without LS its performance is 83% at 300 epochs.

In Figure 3, we compare the performance of sequential and parallel as a function of the number of blocks for ViT-S and ViT-B. Our observations concur with our previous findings: the parallel version is more helpful for the deeper and higher capacity models that are more difficult to optimize; our parallelization scheme alleviates this issue.

Impact of optimization. In Table 2, we provide results with LayerScale [64], which helps the optimization of the biggest models. It improves the performance of both sequential and parallel models, which end up approximately on par. Hence, for models big enough and with proper optimization, sequential and parallel ViTs are roughly equivalent.

Increasing the number of modules or the working dimensionality? Table 3 provides a comparison between different ViT architectures: sequential, parallel, and with larger working dimensionality. We approximately adjust the complexity in terms of parameters and FLOPS, yet this means that ViT models with larger working dimensionality have a higher peak memory usage with typical implementa-

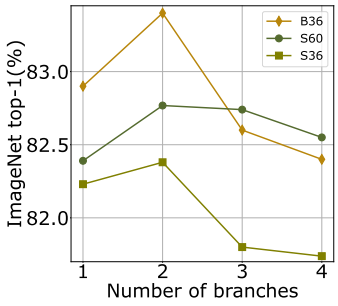


Figure 1: Impact of the parallelism on performance for a given model size (ViT-S36, -S60 and -B36) and 1–4 parallel branches.

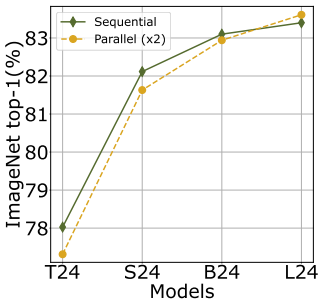


Figure 2: Impact of model width (T:192, S:384, B:768, L:1024). We train the two L24 with LS to avoid optimization issues.

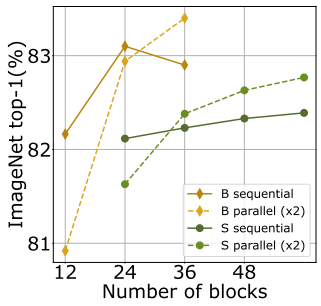


Figure 3: Sequential vs. parallel ViT-S and -B when varying the number of blocks.

Table 2: Impact of the training on parallel and sequential models.

Model	Number of Epochs	LS	ImNet top1 -val	-v2
sequential: ViT-B36x1	300	✗	82.9	72.2
	300	✓	83.9	73.2
	400	✗	83.4	72.5
	400	✓	84.1	73.9
parallel: ViT-B18x2	300	✗	83.3	72.4
	300	✓	83.8	73.3
	400	✗	83.4	73.1
	400	✓	84.1	73.5

Table 3: Comparison of parallel models with more blocks with models with a higher working dimensionality. L24×1, B36×1 and B18×2 trained with LS.

Model	#params ($\times 10^6$)	Flops ($\times 10^9$)	Mem. (MB)	ImNet top1 -val	-v2
B12x1	86.6	17.6	2077	$82.2_{\pm 0.06}$	$71.0_{\pm 0.26}$
S48x1	85.9	18.3	1361	82.3	72.0
S24x2	85.9	18.3	1433	82.6	72.3
L24x1	304.4	61.6	3788	83.4	73.3
B36x1	256.7	52.5	3071	83.9	73.2
B18x2	256.7	52.5	3217	83.8	73.3

Table 4: Throughput for ViT-S18×2 and ViT-B18×2 (im/s). With parallel ViT, the residual blocks can be processed either sequentially (seq) or in parallel (par).

batch	ViT-S18x2				ViT-B18x2			
size	seq	par	best	gain	seq	par	best	gain
1	44	61	61	38%	42	61	61	45%
2	84	123	123	46%	80	117	117	47%
4	168	245	245	46%	155	187	187	21%
8	334	474	474	42%	230	211	230	0%
16	569	518	569	0%	266	231	266	0%
32	616	556	616	0%	276	245	276	0%
64	647	575	647	0%	286	248	286	0%

tion. In both tested settings the sequential and parallel models yield substantially higher accuracy than the models with larger working dimensionality. The sequential and parallel models are comparable with 36 blocks. The parallel model is better in the case of 48 blocks due to the increased depth of the sequential model.

Latency. On a commodity V100 GPUs, we observe a significant speed-up in the case of per-sample processing, with also some gains for small batch sizes with relatively small models, see Table 4. This comparison is based on a simple implementation of our parallel architecture, which is suboptimal due to the lack of a specific CUDA kernel. Overall our measurements suggest specific hardware or kernels are required to obtain compelling benefits in terms of throughput.

4 Fine-tuning attention is all you need

In this section we focus on fine-tuning ViT models, either to adapt the the model to larger image resolutions or to address different downstream classification tasks. In particular, we consider an approach where we only fine-tune the weights corresponding to the MHSA layer, see Figure 4. We analyse the impact in terms of prediction accuracy and savings in processing complexity, peak memory usage and parameter count. As we will see, our choice is significantly better than alternative ones, such as fine-tuning the parameter-heavy FFN layers.

It is common to train networks at lower resolution and fine-tuning it at a higher target resolution. This saves a significant amount of compute at training time, and typically also improves the accuracy of the network at the target resolution [65]. This is because it reduces the discrepancy between the scale of the images seen at train and at test time that is induced by common data augmentation. Fine-tuning is also the paradigm associated with foundation models in general and to the concept of transfer learning itself [22, 50, 73]. A recent line of work explores adaptation of pre-trained models with various types of adapter modules with a small amount of task-specific parameters [5, 29, 45, 46, 51, 54]. In our work, instead, we focus on fine-tuning vanilla ViTs.

Fine-tuning at different resolutions. In Table 5, we report results with fine-tuning ViT-S, ViT-B and ViT-L at 384×384 resolution for models pre-trained at 224×224. Solely fine-tuning the MHSA weights provides results that are within standard deviation (± 0.1) from a full fine-tuning both on ImageNet-val and ImageNet-V2. This is not the case when fine-tuning the FFN layers, while these contain twice the number of parameters of MHSA. Note, our pre-trained models have been trained long enough (400 epochs) to ensure convergence.

There are only advantages to use this approach when fine-tuning at higher resolution as opposed to doing a full fine-tuning, as we get substantial savings in terms of parameters, latency, and peak memory usage for free, see Figure 4 (right panels). First, the fine-tuning stage requires 10% less memory on the GPU, which is especially interesting in the context of high-resolution fine-tuning where the higher images require more memory. The training is also 10% faster, as less gradients are computed. Finally, the attention weights correspond to approximately one third of the weights. Therefore, if one wants to use multiple models fine-tuned for different input resolutions, we save 66% of the storage for each additional model.

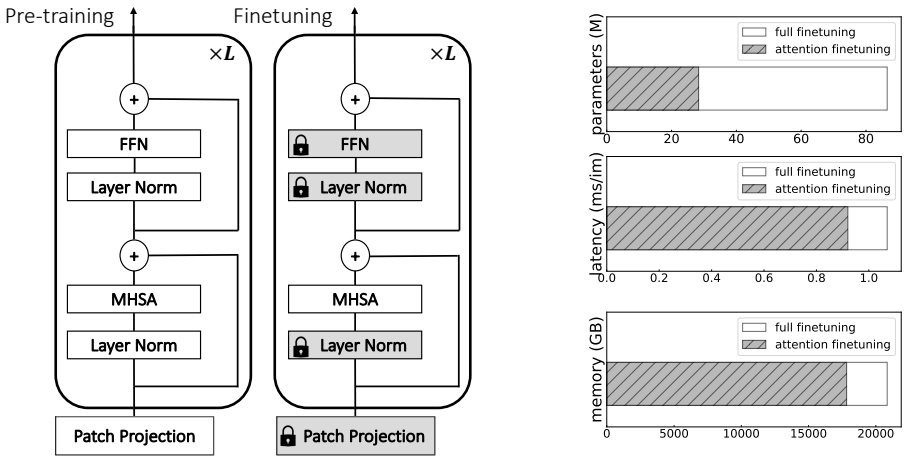


Figure 4: Fine-tuning the weights of the self-attention layer only (middle panel) leads to savings during fine-tuning in peak memory usage and computational cost. It also leads to important savings in the number of parameters when a model is fine-tuned for multiple resolutions or multiple downstream classification tasks.

Table 5: Comparison of full finetuning of all weight (full), finetuning of the MHSA layer weights only (attn) and of the FFN layer only (ffn) when adapting models at resolution 384×384 on ImageNet-1k from model pre-trained at 224×224. We compare finetuning with the SGD and AdamW [43] optimizers.

ImageNet1k-val top1 acc.						
Model	AdamW↑384			SGD↑384		
	full	attn	ffn	full	attn	ffn
ViT-S	**82.7**	82.5	82.2	82.6	82.3	82.0
ViT-B	**84.3**	**84.3**	84.1	84.3	84.2	84.0
ViT-L	**85.5**	**85.5**	85.2	85.4	85.3	85.1

ImageNet1k-V2 top1 acc.						
Model	AdamW↑384			SGD↑384		
	full	attn	ffn	full	attn	ffn
ViT-S	72.5	72.4	71.6	72.5	72.2	71.1
ViT-B	73.7	74.0	73.6	74.0	73.9	73.7
ViT-L	75.5	75.4	75.2	75.6	75.1	75.0

Fine-tuning on different datasets. We now evaluate our approach when transferring ViTs pre-trained on ImageNet to different downstream classification tasks by fine-tuning. We consider public benchmarks whose characteristics and references are given in Appendix B.

In Table 6 we report the performance for different fine-tuning strategies. Here we make different observations. First, for the **smallest datasets**, namely CARS and Flower, fine-tuning only the MHSA layers is an excellent strategy. It is even better than full-tuning. Our interpretation is that restricting the number of weights has a regularizing effect. The conclusion is more mixed with the **largest datasets**, in particular iNaturalist, where we observe a significant gap between the full fine-tuning and our solution for the ViT-S. This could be expected: in this case there are more images to learn from and new classes that were not seen before the fine-tuning stage. Restricting the fine-tuning to MHSA layer allows modifying only a relatively small number of parameters. FFN layers have twice more weights and leads to better results in that case. This limitation tends to disappear with the **larger ViT-L models**, for which the the capacity of the MHSA is much larger and therefore sufficient. Our strategy is therefore interesting in the typical use-cases of

Table 6: Transfer learning experiments: we compare full finetuning, finetuning of attention only and finetuning with ffn only on 6 transfer learning dataset with 3 differents ViT models pre-trained on ImageNet-1k only.

Model	Finetuning			INAT-18	INAT-19	CIFAR-10	CIFAR-100	CARS	Flowers
	full	attn	ffn						
ViT-S	✓	✗	✗	**68.0**	**73.9**	**98.9**	**90.5**	89.7	96.8
	✗	✓	✗	60.6	68.7	98.7	89.1	**89.8**	**96.9**
	✗	✗	✓	64.4	72.5	98.9	90.1	88.3	96.1
ViT-B	✓	✗	✗	**74.1**	**78.2**	**99.3**	**92.5**	92.7	97.8
	✗	✓	✗	71.1	75.7	99.2	91.8	**92.8**	**98.5**
	✗	✗	✓	73.3	77.3	**99.3**	92.1	88.9	97.5
ViT-L	✓	✗	✗	**75.9**	**79.7**	**99.3**	**93.2**	**93.8**	98.3
	✗	✓	✗	75.3	78.7	99.2	92.7	**93.8**	**98.4**
	✗	✗	✓	75.4	79.3	99.2	93.0	93.0	97.6

foundation models, which are very large models that are fine-tuned on a variety of downstream tasks.

5 Patch preprocessing for Bert-like self-supervised learning

The original ViT paper [16] considered to include convolution instead of patch projection in the network design. Several recent papers [21, 23, 67, 70, 71, 74] advocate this choice to include a small pre-processing network in the architecture, instead of a simple patch projection. Most of the pre-processing subnetworks that have been considered are based on convolutions, and are often referred to as "convolutional stems". Small transformers have also been considered [74].

While these patch pre-processing designs have been developed to improve accuracy and/or stability, there are some remaining questions regarding their design and flexibility. First, it is not clear which is the most effective when combined with a vanilla transformer. Second, to our knowledge there is no work addressing the problem of their compatibility with self-supervised methods based on patch masking, and in particular on Bert-like auto-encoders such as BeiT [3].

In this section we try to answer these questions. We compare several existing pre-processing designs in terms of accuracy and compute and evaluate them in combination with BeiT, using the codebase release by the authors of BeiT. The only change we make is to train the tokenizer on ImageNet-1k, rather than using the one from DALL-E [53] used in BeiT which is trained on a proprietary dataset comprised of 250 million images. In this manner, pre-training is based on ImageNet-1k only. This permits reproducible experimentation and fair comparison, and gives equivalent results [49]. Since existing convolutional designs are not satisfactory in combination with masking, we first introduce our own design.

Our hierarchical MLP (hMLP) stem is depicted in Figure 5. All patches are processed independently with linear layers interleaved with non-linearities and renormalization. Its design is guided by our motivation to remove any interac-

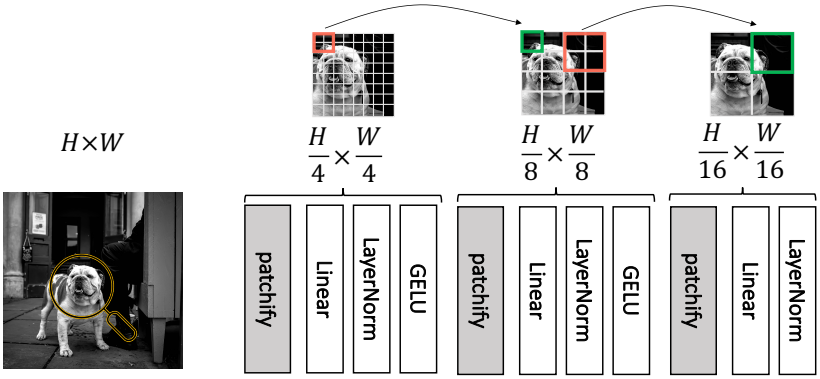


Figure 5: Design of our hMLP-stem: we start from subpatches and progressively merge them with linear layers interleaved by GELU non-linearities. The design of our stem is such that the patches are processed independently. Hence it commutes with masking.

tion between the different 16×16 patches during the pre-processing stage. Even if we mask a patch, it does not create any artifacts resulting from the convolution overlapping with other patches, as it is the case with existing designs. Therefore, with our hMLP solution, we can equivalently mask the patches before or after the patch-processing stage. Note that, although patches are processed independently, our hMLP-stem is equivalent to a convolutional stem in which the size of the convolutional kernel and its stride are matched, and in practice we implement it with convolutional layers, see our code in Appendix C.

In short, we start from small 2×2 patches, and gradually increase their size until they reach 16×16. Each increase of the patch size is denoted by "*patchify*" in Figure 5, in spirit of hierarchical transformer designs like Swin-Transformers [41]. The patches are projected with a linear projection and normalized before we apply a GELU non-linearity [27]. For the normalization, we consider and evaluate two choices: either we use batch-normalization [32] (BN) or layer-normalization (LN) [2]. While the BN offers better trade-offs, LN is of interest when used with small batch sizes: it works well even with a single image per batch, as often used in object detection.

In contrast with existing stems from the literature, our hMLP design does not significantly increase the compute requirement. For instance, ViT-B, requires FLOPS is 17.73 GFLOPS with our design. This adds less than 1% of compute compared to using the usual linear projection stem.

Stem comparison in supervised learning. In Table 7 we provide a comparison between different stem designs. We have selected several prototypical designs from the literature for which the code is available online. In addition to our hMLP stem, we have considered some variations over the standard linear projection to evaluate the influence of the non-linearities and normalization. For the standard linear stem, we also consider a ViT-B13 including an extra pair (MHSA, FFN) to allow more direct comparisons with other stems with more FLOPS. In this comparison the most effective existing design is the one of LeViT [21]. The improvements with respect to the linear baseline are significant considering the standard deviation, even when taking into account the extra layer of ViT-B13 to compare with an simi-

Table 7: **Patch pre-processing:** Performance in top1 accuracy with for a ViT-B12. All models are (1) trained 300 epochs in the supervised case; (2) pre-trained during 300 epochs and fine-tuned 100 epochs when used with BeiT. We report the result of a ViT-B13 to provide the performance of a vanilla transformer with more FLOPS. We measure the standard deviation for the two linear stem baselines and our hMLP stem on 5 runs. The other measurements are made with the fixed seed 0.

Stem type	norm.	NL	GFLOPS	ImNet1k supervised acc. -val	ImNet1k supervised acc. -v2	BeiT+FT ImNet-val
Linear: ViT-B12 [16]	–	–	17.58	82.20±0.06	71.0	83.05±0.08
	BN	–	17.58	82.31	71.0	82.98
	–	GELU	17.58	81.55	70.5	83.09
	BN	GELU	17.58	82.38	70.7	82.99
Linear: ViT-B13	–	–	19.04	82.35±0.12	71.3	83.26±0.06
Conv: Graham et al. [21]	BN	GELU	19.07	**82.57**	71.0	83.04
	LN	GELU	19.07	**82.50**	70.9	83.06
Local transformer [23]			19.12	82.26	70.6	82.38
hMLP (ours)	BN	GELU	17.73	**82.54**±0.09	**71.5**	**83.43**±0.10
	LN	GELU	17.73	**82.50**±0.07	71.0	83.24±0.09

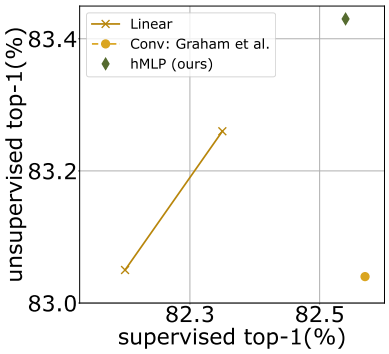


Figure 6: Performance of patch pre-processing in the supervised and BeiT+FT settings. Our hMLP stem performs well in both cases, improving the accuracy compared to linear projection (shown for B12 and B13) without significantly increasing the complexity (+0.8% FLOPS compared to the ViT-B12 in the bottom-left corner). In contrast, the convolutional stem only improves the performance in the supervised case, while significantly increasing complexity (+7.5% FLOPS).

lar number of FLOPS. Our hMLP stem obtains a comparable performance but with lower complexity, and without any interaction between the 16×16 patches.

Results with BeiT training. We report the results with BeiT, fine-tuned on ImageNet-val, in the right-most column of Table 7. We use the code of BeiT [3] with their training procedure, which includes LayerScale and a relatively elaborated fine-tuning procedure. As one can see, existing stems do not provide any improvement compared to the linear baseline, while adding compute. In contrast, our design is effective and provides an improvement of +0.3/+0.4 top1 accuracy compared to the baseline, which is significant considering the measure uncertainty. The interest of hMLP in the context of masked self-supervised learning is clear in Figure 6, where we plot the performance, averaged over 5 seeds for our method, in the supervised case versus the one with BeiT.

6 Conclusion

In this paper, we looked at three different topics related to Vision Transformers. First, we investigated a simple but effective way to parallelize them, showing a viable alternative to increase capacity without significantly increasing the working dimensionality. Whether this simple parallel design principle can be applied to other architectures is an exploration left for future work. Second, we considered different fine-tuning strategies and showed that fine-tuning the self-attention layer is sufficient in the context of resolution fine-tuning. This can also be interesting when transferring to other downstream classification tasks, especially when fine-tuning large models or/and transferring to a dataset with few training images. Last, we introduced a simple patch pre-processing stem, which processes patches independently across multiple linear layers interleaved with non-linearities and patch aggregation. It is especially useful when combined with mask-based self-supervised learning such as BeiT.

Acknowledgement. We thank Francisco Massa for valuable discussions and insights about optimizing the implementation of block parallelization.

References

[1] Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., Schmid, C.: ViVit: A video vision transformer. In: International Conference on Computer Vision (2021)

[2] Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)

[3] Bao, H., Dong, L., Wei, F.: BEiT: BERT pre-training of image transformers. arXiv preprint **arXiv:2106.08254** (2021)

[4] Bello, I., Zoph, B., Vaswani, A., Shlens, J., Le, Q.V.: Attention augmented convolutional networks. In: Conference on Computer Vision and Pattern Recognition (2019)

[5] Berriel, R., Lathuilière, S., Nabi, M., Klein, T., Oliveira-Santos, T., Sebe, N., Ricci, E.: Budget-aware adapters for multi-domain learning. In: International Conference on Computer Vision (2019)

[6] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. arXiv preprint **arXiv:2005.14165** (2020)

[7] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. In: European Conference on Computer Vision (2020)

[8] Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers. arXiv preprint **arXiv:2104.14294** (2021)

[9] Chang, H., Zhang, H., Jiang, L., Liu, C., Freeman, W.T.: MaskGIT: Masked generative image transformer. arXiv preprint **arXiv:2202.04200** (2022)

[10] Chen, X., Xie, S., He, K.: An empirical study of training self-supervised vision transformers. In: International Conference on Computer Vision (2021)

[11] d'Ascoli, S., Touvron, H., Leavitt, M.L., Morcos, A.S., Biroli, G., Sagun, L.: ConViT: Improving vision transformers with soft convolutional inductive biases. In: ICML (2021)

[12] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: NAACL (2019)

[13] Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., Sun, J.: Repvgg: Making vgg-style convnets great again. Conference on Computer Vision and Pattern Recognition (2021)

[14] Ding, X., Zhang, X., Han, J., Ding, G.: RepMLP: Re-parameterizing convolutions into fully-connected layers for image recognition. arXiv preprint **arXiv:2105.01883** (2021)

[15] Dong, X., Bao, J., Zhang, T., Chen, D., Zhang, W., Yuan, L., Chen, D., Wen, F., Yu, N.: Peco: Perceptual codebook for BERT pre-training of vision transformers. arXiv preprint **arXiv:2111.12710** (2021)

[16] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. In: International Conference on Learning Representations (2021)

[17] El-Nouby, A., Izacard, G., Touvron, H., Laptev, I., Jegou, H., Grave, E.: Are large-scale datasets necessary for self-supervised pre-training? arXiv preprint **arXiv:2112.10740** (2021)

[18] El-Nouby, A., Touvron, H., Caron, M., Bojanowski, P., Douze, M., Joulin, A., Laptev, I., Neverova, N., Synnaeve, G., Verbeek, J., et al.: XCiT: Cross-covariance image transformers. In: NeurIPS (2021)

[19] Fan, H., Xiong, B., Mangalam, K., Li, Y., Yan, Z., Malik, J., Feichtenhofer, C.: Multiscale vision transformers. arXiv preprint **arXiv:2104.11227** (2021)

[20] Goyal, A., Bochkovskiy, A., Deng, J., Koltun, V.: Non-deep networks. arXiv preprint **arXiv:2110.07641** (2021)

[21] Graham, B., El-Nouby, A., Touvron, H., Stock, P., Joulin, A., Jégou, H., Douze, M.: LeViT: a vision transformer in convnet's clothing for faster inference. arXiv preprint **arXiv:2104.01136** (2021)

[22] Guo, Y., Shi, H., Kumar, A., Grauman, K., Simunic, T., Feris, R.S.: Spottune: Transfer learning through adaptive fine-tuning. Conference on Computer Vision and Pattern Recognition (2019)

[23] Han, K., Xiao, A., Wu, E., Guo, J., Xu, C., Wang, Y.: Transformer in transformer. arXiv preprint **arXiv:2103.00112** (2021)

[24] He, K., Chen, X., Xie, S., Li, Y., Dollár, P., Girshick, R.: Masked autoencoders are scalable vision learners. arXiv preprint **arXiv:2111.06377** (2021)

[25] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Conference on Computer Vision and Pattern Recognition (2016)

[26] He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. arXiv preprint **arXiv:1603.05027** (2016)

[27] Hendrycks, D., Gimpel, K.: Gaussian error linear units (GELUs). arXiv preprint arXiv:1606.08415 (2016)

[28] Horn, G.V., Mac Aodha, O., Song, Y., Shepard, A., Adam, H., Perona, P., Belongie, S.J.: The inaturalist challenge 2017 dataset. arXiv preprint arXiv:1707.06642 (2017)

[29] Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., Gelly, S.: Parameter-efficient transfer learning for NLP. In: ICML (2019)

[30] Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K.Q.: Deep networks with stochastic depth. In: European Conference on Computer Vision (2016)

[31] Hudson, D.A., Zitnick, C.L.: Generative adversarial transformers. In: International Conference on Machine Learning (2021)

[32] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning (2015)

[33] Karita, S., Chen, N., Hayashi, T., et al.: A comparative study on transformer vs RNN in speech applications. arXiv preprint **arXiv:1909.06317** (2019)

[34] Krause, J., Stark, M., Deng, J., Fei-Fei, L.: 3d object representations for fine-grained categorization. In: IEEE Workshop on 3D Representation and Recognition (2013)

[35] Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: NeurIPS (2012)

[36] Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., CIFAR (2009)

[37] Lample, G., Charton, F.: Deep learning for symbolic mathematics. arXiv preprint **arXiv:1912.01412** (2019)

[38] Li, X., Wang, W., Hu, X., Yang, J.: Selective kernel networks. Conference on Computer Vision and Pattern Recognition (2019)

[39] Liu, H., Dai, Z., So, D.R., Le, Q.V.: Pay attention to MLPs. arXiv preprint **arXiv:2105.08050** (2021)

[40] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. arXiv preprint **arXiv:1907.11692** (2019)

[41] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. arXiv preprint **arXiv:2103.14030** (2021)

[42] Liu, Z., Mao, H., Wu, C.Y., Feichtenhofer, C., Darrell, T., Xie, S.: A convnet for the 2020s. arXiv preprint **arXiv:2201.03545** (2022)

[43] Loshchilov, I., Hutter, F.: Fixing weight decay regularization in adam. arXiv preprint **arXiv:1711.05101** (2017)

[44] Lüscher, C., Beck, E., Irie, K., et al.: RWTH ASR systems for LibriSpeech: Hybrid vs attention. In: Interspeech (2019)

[45] Mahabadi, R.K., Ruder, S., Dehghani, M., Henderson, J.: Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In: ACL/IJCNLP (2021)

[46] Mancini, M., Ricci, E., Caputo, B., Bulò, S.R.: Adding new tasks to a single network with weight transformations using binary masks. In: European Conference on Computer Vision Workshops (2018)

[47] Melas-Kyriazi, L.: Do you even need attention? A stack of feed-forward layers does surprisingly well on ImageNet. arXiv preprint **arXiv:2105.02723** (2021)

[48] Nilsback, M.E., Zisserman, A.: Automated flower classification over a large number of classes. In: Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing (2008)

[49] Nouby, A.E., Izacard, G., Touvron, H., Laptev, I., Jégou, H., Grave, E.: Are large-scale datasets necessary for self-supervised pre-training? arXiv preprint arXiv:2112.10740 (2021)

[50] Oquab, M., Bottou, L., Laptev, I., Sivic, J.: Learning and transferring mid-level image representations using convolutional neural networks. In: Conference on Computer Vision and Pattern Recognition (2014)

[51] Pfeiffer, J., Rücklé, A., Poth, C., Kamath, A., Vulic, I., Ruder, S., Cho, K., Gurevych, I.: AdapterHub: A framework for adapting transformers. In: EMNLP (2020)

[52] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2019)

[53] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., Sutskever, I.: Zero-shot text-to-image generation. arXiv preprint **arXiv:2102.12092** (2021)

[54] Rebuffi, S.A., Bilen, H., Vedaldi, A.: Efficient parametrization of multi-domain deep neural networks. International Conference on Computer Vision (2018)

[55] Recht, B., Roelofs, R., Schmidt, L., Shankar, V.: Do ImageNet classifiers generalize to ImageNet? In: International Conference on Machine Learning (2019)

[56] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet large scale visual recognition challenge. International Journal of Computer Vision **115**(3), 211–252 (2015)

[57] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)

[58] Steiner, A., Kolesnikov, A., Zhai, X., Wightman, R., Uszkoreit, J., Beyer, L.: How to train your ViT? data, augmentation, and regularization in vision transformers. arXiv preprint **arXiv:2106.10270** (2021)

[59] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Conference on Computer Vision and Pattern Recognition (2015)

[60] Tolstikhin, I., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Keysers, D., Uszkoreit, J., Lucic, M., Dosovitskiy, A.: MLP-Mixer: An all-MLP architecture for vision. arXiv preprint **arXiv:2105.01601** (2021)

[61] Touvron, H., Bojanowski, P., Caron, M., Cord, M., El-Nouby, A., Grave, E., Joulin, A., Synnaeve, G., Verbeek, J., Jégou, H.: ResMLP: feedforward networks for image classification with data-efficient training. arXiv preprint **arXiv:2105.03404** (2021)

[62] Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jégou, H.: Training data-efficient image transformers & distillation through attention. In: International Conference on Machine Learning (2021)

[63] Touvron, H., Cord, M., El-Nouby, A., Bojanowski, P., Joulin, A., Synnaeve, G., Jégou, H.: Augmenting convolutional networks with attention-based aggregation. arXiv preprint **arXiv:2112.13692** (2021)

[64] Touvron, H., Cord, M., Sablayrolles, A., Synnaeve, G., Jégou, H.: Going deeper with image transformers. International Conference on Computer Vision (2021)

[65] Touvron, H., Vedaldi, A., Douze, M., Jegou, H.: Fixing the train-test resolution discrepancy. In: NeurIPS (2019)

[66] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: NeurIPS (2017)

[67] Wang, W., Xie, E., Li, X., Fan, D.P., Song, K., Liang, D., Lu, T., Luo, P., Shao, L.: Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. arXiv preprint **arXiv:2102.12122** (2021)

[68] Wei, C., Fan, H., Xie, S., Wu, C.Y., Yuille, A., Feichtenhofer, C.: Masked feature prediction for self-supervised visual pre-training. arXiv preprint **arXiv:2112.09133** (2021)

[69] Wightman, R., Touvron, H., Jégou, H.: ResNet strikes back: An improved training procedure in timm. arXiv preprint **arXiv:2110.00476** (2021)

[70] Wu, H., Xiao, B., Codella, N.C.F., Liu, M., Dai, X., Yuan, L., Zhang, L.: Cvt: Introducing convolutions to vision transformers. arXiv preprint **arXiv:2103.15808** (2021)

[71] Xiao, T., Singh, M., Mintun, E., Darrell, T., Dollár, P., Girshick, R.B.: Early convolutions help transformers see better. arXiv preprint **arXiv:2106.14881** (2021)

[72] Xie, Z., Zhang, Z., Cao, Y., Lin, Y., Bao, J., Yao, Z., Dai, Q., Hu, H.: Simmim: A simple framework for masked image modeling. arXiv preprint **arXiv:2111.09886** (2021)

[73] Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? arXiv preprint **arXiv:1411.1792** (2014)

[74] Yuan, L., Chen, Y., Wang, T., Yu, W., Shi, Y., Tay, F., Feng, J., Yan, S.: Tokens-to-token vit: Training vision transformers from scratch on imagenet. arXiv preprint **arXiv:2101.11986** (2021)

[75] Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv preprint **arXiv:1605.07146** (2016)

[76] Zhou, J., Wei, C., Wang, H., Shen, W., Xie, C., Yuille, A., Kong, T.: ibot: Image bert pre-training with online tokenizer. International Conference on Learning Representations (2022)

Three things everyone should know about ViTs

– Supplemental material –

A Baselines

↓ Training procedure	#epochs	ViT-Ti	ViT-S	ViT-B	ViT-L
DeiT [62]	300	72.2	79.8	81.8	–
Steiner et al. [58]	300	69.6	76.0	78.7	74.0
He et al. [24]	300	–	–	82.1	81.5†
He et al. [24] with EMA	300	–	–	82.3	82.6†
Our baseline	300	72.7	79.7	82.2±0.06	83.0
Our baseline with LayerScale [64]	400	73.5	80.7	82.7	84.0

Table 8: Comparison our baseline with previous training procedures. We only include results that correspond to the vanilla ViT introduced by Dosovitskiy et al. [16] for Vit-B, Vit-L and Touvron et al. [62] for Vit-Ti and ViT-S. All models are train oned ImageNet-1k at resolution 224×224 without distillation. †200 epochs.

B Transfer Learning Datasets

Table 9: Datasets used in transfer experiments and corresponding references.

Dataset	Train size	Test size	#classes
ImageNet [56]	1,281,167	50,000	1000
iNaturalist 2018 [28]	437,513	24,426	8,142
iNaturalist 2019 [28]	265,240	3,003	1,010
Flowers-102 [48]	2,040	6,149	102
Stanford Cars [34]	8,144	8,041	196
CIFAR-100 [36]	50,000	10,000	100
CIFAR-10 [36]	50,000	10,000	10

C Pytorch code of our hMLP Stem

Algorithm 1 Pseudocode of hMLP stem in a PyTorch-like style.

```
import torch
import torch.nn as nn
class hMLP_stem(nn.Module):
   """ Image to Patch Embedding
   """
   def __init__(self, img_size=(224,224), patch_size=(16,16), in_chans=3, embed_dim
       =768):
      super().__init__()
      num_patches = (img_size[1] // patch_size[1]) * (img_size[0] // patch_size[0])
      self.img_size = img_size
      self.patch_size = patch_size
      self.num_patches = num_patches
      self.proj = torch.nn.Sequential(
         *[nn.Conv2d(in_chans, embed_dim//4, kernel_size=4, stride=4),
         nn.SyncBatchNorm(embed_dim//4),
         nn.GELU(),
         nn.Conv2d(embed_dim//4, embed_dim//4, kernel_size=2, stride=2),
         nn.SyncBatchNorm(embed_dim//4),
         nn.GELU(),
         nn.Conv2d(embed_dim//4, embed_dim, kernel_size=2, stride=2),
         nn.SyncBatchNorm(embed_dim),
         ])

   def forward(self, x):
      B, C, H, W = x.shape
      x = self.proj(x).flatten(2).transpose(1, 2)
      return x
```