Going deeper with Image Transformers

Hugo Touvron Matthieu Cord Alexandre Sablayrolles Gabriel Synnaeve Hervé Jégou

Abstract

Transformers have been recently adapted for large scale image classification, achieving high scores shaking up the long supremacy of convolutional neural networks. However the optimization of vision transformers has been little studied so far. In this work, we build and optimize deeper transformer networks for image classification. In particular, we investigate the interplay of architecture and optimization of such dedicated transformers. We make two architecture changes that significantly improve the accuracy of deep transformers. This leads us to produce models whose performance does not saturate early with more depth, for instance we obtain 86.5% top-1 accuracy on Imagenet when training with no external data, we thus attain the current sate of the art with less floating-point operations and parameters. Our best model establishes the new state of the art on Imagenet with Reassessed labels and Imagenet-V2 / match frequency, in the setting with no additional training data. We share our code and models[1].

1. Introduction

Residual architectures are prominent in computer vision since the advent of ResNet [27]. They are defined as a sequence of functions of the form

$$x_{l+1} = g_l(x_l) + R_l(x_l), \tag{1}$$

where the function g_l and R_l define how the network updates the input x_l at layer l. The function g_l is typically identity, while R_l is the main building block of the network: many variants in the literature essentially differ on how this residual branch R_l is constructed or parametrized [51, 63, 73]. Residual architectures highlight the strong interplay between optimization and architecture design. As pointed out by He *et al.* [27], residual networks do not offer a better representational power. They achieve better performance because they are easier to train: shortly after their seminal work, He *et al.* discussed [28] the importance of having a clear path both forward and backward, and advocate setting g_l to the identity function.

[1] `https://github.com/facebookresearch/deit`

The vision transformers [19] instantiate a particular form of residual architecture: after casting the input image into a set x_0 of vectors, the network alternates self-attention layers (SA) with feed-forward networks (FFN), as

$$\begin{aligned} x'_l &= x_l + \mathrm{SA}(\eta(x_l)) \\ x_{l+1} &= x'_l + \mathrm{FFN}(\eta(x'_l)) \end{aligned} \tag{2}$$

where η is the LayerNorm operator [1]. This definition follows the original architecture of Vaswani *et al.* [67], except the LayerNorm is applied before the block (*pre-norm*) in the residual branch, as advocated by He *et al.* [28]. Child *et al.* [13] adopt this choice with LayerNorm for training deeper transformers for various media, including for image generation where they train transformers with 48 layers.

How to normalize, weigh, or initialize the residual blocks of a residual architecture has received significant attention both for convolutional neural networks [7, 8, 28, 76] and for transformers applied to NLP or speech tasks [2, 34, 76]. In Section 2, we revisit this topic for transformer architectures solving image classification problems. Examples of approaches closely related to ours include Fixup [76], T-Fixup [34], ReZero [2] and SkipInit [16].

Following our analysis of the interplay between different initialization, optimization and architectural design, we propose an approach that is effective to improve the training of deeper architecture compared to current methods for image transformers. Formally, we add a learnable diagonal matrix on the output of each residual block, initialized close to (but not at) 0. Adding this simple layer after each residual block improves the training dynamic, allowing us to train deeper high-capacity image transformers that benefit from depth. We refer to this approach as **LayerScale**.

Section 3 introduces our second contribution, namely **class-attention layers,** that we present in Figure 2. It is akin to an encoder/decoder architecture, in which we explicitly separate the transformer layers involving self-attention between patches, from class-attention layers that are devoted to extract the content of the processed patches into a single vector so that it can be fed to a linear classifier. This explicit separation avoids the contradictory objective of guiding the attention process while processing the class embedding. We refer to this new architecture as **CaiT** (Class-Attention in Image Transformers).

In the experimental Section 4, we empirically show the effectiveness and complementary of our approaches:

- LayerScale significantly facilitates the convergence and improves the accuracy of image transformers at larger depths. It adds a few thousands of parameters to the network at training time (negligible with respect to the total number of weights).
- Our architecture with specific class-attention offers a more effective processing of the class embedding.
- Our best CaiT models establish the new state of the art on Imagenet-Real [6] and Imagenet V2 matched frequency [53] with no additional training data. On ImageNet1k-val [55], our model is on par with the state of the art (86.5%) while requiring less FLOPs (329B vs 377B) and having less parameters than the best competing model (356M vs 438M).
- We also achieve competitive results on Transfer Learning, see Section C in supplemental material.

We discuss related works along this paper and in the dedicated Section 5, before we conclude in Section 6.

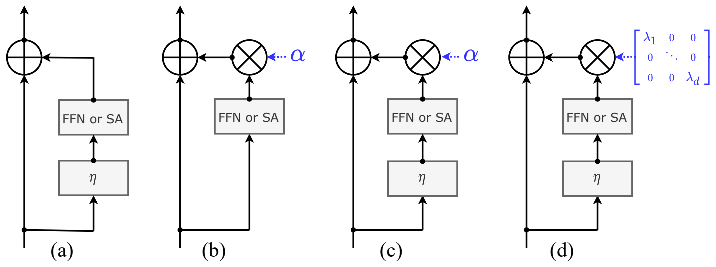
2. Deeper image transformers with LayerScale

Our goal is to increase the stability of the optimization when training transformers for image classification derived from the original architecture by Vaswani *et al.* [67], and especially when we increase their depth. We consider more specifically the vision transformer (ViT) architecture proposed by Dosovitskiy *et al.* [19] as the reference architecture and adopt the data-efficient image transformer (DeiT) optimization procedure of Touvron *et al.* [64]. In both works, there is no evidence that depth can bring any benefit when training on Imagenet only: the deeper ViT architectures have a lower performance, while DeiT only considers transformers with 12 blocks of layers. Section 4 will confirm that DeiT does not train deeper models effectively.

Figure 1 depicts the main variants we compare to facilitate the optimization. They cover recent choices from the literature: as discussed in the introduction, the architecture (a) of ViT and DeiT is a pre-norm architecture [19, 64], in which the layer-normalisation η occurs at the beginning of the residual branch. Note that the original architecture of Vaswani *et al.* [67] applies the normalization after the block, but in our experiments the DeiT training does not converge with post-normalization.

Fixup [76], ReZero [2] and SkipInit [16] introduce learnable scalar weighting α_l on the output of residual blocks, while removing the pre-normalization and the warmup, see Figure 1(b). This amounts to modifying Eqn. 2 as

$$
\begin{aligned}
x'_l &= x_l + \alpha_l \, \mathrm{SA}(x_l) \\
x_{l+1} &= x'_l + \alpha'_l \, \mathrm{FFN}(x'_l).
\end{aligned}
\tag{3}
$$



FFN or SA
η
α
$\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_d \end{bmatrix}$
(a)
(b)
(c)
(d)

Figure 1. Normalization strategies for transformer blocks. (a) The ViT image classifier adopts pre-normalization like Child *et al.* [13]. (b) ReZero/Skipinit and Fixup remove the η normalization and the warmup (i.e., a reduced learning rate in the early training stage) and add a learnable scalar initialized to $\alpha = 0$ and $\alpha = 1$, respectively. Fixup additionally introduces biases and modifies the initialization of the linear layers. Since these methods do not converge with deep vision transformers, (c) we adapt them by re-introducing the pre-norm η and the warmup. Our main proposal (d) introduces a per-channel weighting (i.e, multiplication with a diagonal matrix $\mathrm{diag}(\lambda_1, \ldots, \lambda_d)$, where we initialize each weight with a small value as $\lambda_i = \varepsilon$.

ReZero simply initializes this parameter to $\alpha = 0$. Fixup initializes this parameter $\alpha = 1$ and makes other modifications: it adopts different policies for the initialization of the block weights, and adds several weights to the parametrization. In our experiments, these approaches do not converge even with some adjustment of the hyper-parameters.

Our empirical observation is that removing the warmup and the layer-normalization is what makes training unstable in Fixup and T-Fixup. Therefore we re-introduce these two ingredients so that Fixup and T-Fixup converge with DeiT models, see Figure 1(c). As we see in the experimental section, these amended variants of Fixup and T-Fixup help with convergence. The main contributing factor is the learnable α_l, which when initialized at a small value do help the convergence when we increase the depth.

Our proposal LayerScale is a per-channel multiplication of the vector produced by each residual block, as opposed to a single scalar, see Figure 1(d). Our objective is to group the updates of the weights associated with the same output channel. Formally, LayerScale is a multiplication by a diagonal matrix on output of each residual block. In other terms, we modify Eqn. 2 as

$$
\begin{aligned}
x'_l &= x_l + \mathrm{diag}(\lambda_{l,1}, \ldots, \lambda_{l,d}) \times \mathrm{SA}(\eta(x_l)) \\
x_{l+1} &= x'_l + \mathrm{diag}(\lambda'_{l,1}, \ldots, \lambda'_{l,d}) \times \mathrm{FFN}(\eta(x'_l)),
\end{aligned}
\tag{4}
$$

where the parameters $\lambda_{l,i}$ and $\lambda'_{l,i}$ are learnable weights. The diagonal values are all initialized to a fixed small value ε: we set it to $\varepsilon = 0.1$ until depth 18, $\varepsilon = 10^{-5}$ for 24 and $\varepsilon = 10^{-6}$ for deeper networks. This formula is akin to other normalization strategies ActNorm [37] or LayerNorm but executed on output of the residual block. Yet we seek a different effect: ActNorm is a data-dependent initialization that calibrates activations so that they have zero-mean and unit variance, like batchnorm [35]. In contrast, we initialize the diagonal with small values so that the initial contribution of the residual branches to the function implemented by the transformer is small. In that respect our motivation is therefore closer to that of ReZero [2], SkipNorm [16], Fixup [76] and TFixup [34]: we start to train closer to the identity function and let the network integrate the additional parameters progressively during the training. However, LayerScale offers more diversity in the optimization than just adjusting the whole layer by a single learnable scalar as in ReZero/SkipNorm, Fixup and T-Fixup. As we will show empirically, offering the freedom to do so per channel is a decisive advantage of LayerScale over existing approaches.

Formally, adding these weights does not change the expressive power of the architecture since they can be integrated into the previous matrix of the SA and FFN layers.

3. Specializing layers for class attention

In this section, we introduce the CaiT architecture, depicted in Figure 2 (right). This design aims at circumventing one of the problems of the ViT architecture: the learned weights are asked to optimize two contradictory objectives: (1) guiding the self-attention between patches while (2) summarizing the information useful to the linear classifier. Our proposal is to explicitly separate the two stages.

Later class token. As an intermediate step towards our proposal, we insert the so-called class token, denoted by CLS, later in the transformer. This choice eliminates the discrepancy on the first layers of the transformer, which are therefore fully employed for performing self-attention between patches only. As a baseline that does not suffer from the contradictory objectives, we also consider **average pooling** of all the patches on output of the transformers, as typically employed in convolutional architectures.

Architecture. Our CaiT network consists of two distinct processing stages visible in Figure 2:

1. The *self-attention* stage is identical to the ViT transformer, but with no class embedding (CLS).

2. The *class-attention* stage is a set of layers that compiles the set of patch embeddings into a class embedding CLS that is subsequently fed to a linear classifier.

This class-attention alternates in turn a layer that we refer to as a multi-head class-attention (CA), and a FFN layer. In this stage, only the class embedding is updated. Similar to the one fed in ViT and DeiT on input of the transformer, it is a learnable vector. The main difference is that, in our architecture, we do no copy information from the class embedding to the patch embeddings during the forward pass.

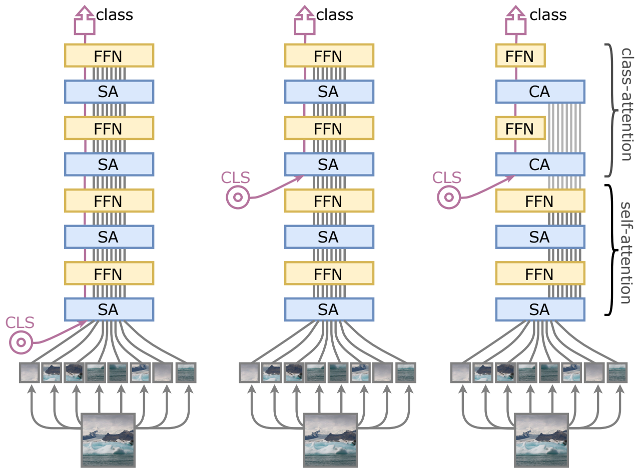


Figure 2. In the ViT transformer (*left*), the class embedding (CLS) is inserted along with the patch embeddings. This choice is detrimental, as the same weights are used for two different purposes: helping the attention process, and preparing the vector to be fed to the classifier. We put this problem in evidence by showing that inserting CLS later improves performance (*middle*). In the **CaiT** architecture (*right*), we further propose to freeze the patch embeddings when inserting CLS to save compute, so that the last part of the network (typically 2 layers) is fully devoted to summarizing the information to be fed to the linear classifier.

Only the class embedding is updated by residual in the CA and FFN processing of the summarize stage.

Multi-heads class attention. The role of the CA layer is to extract the information from the set of processed patches.It is identical to a SA layer, except that it relies on the attention between (i) the class embedding x_{class} (initialized at CLS in the first CA) and (ii) itself plus the set of frozen patch embeddings x_{patches}.

Considering a network with h heads and p patches, and denoting by d the embedding size, we parametrize the multi-head class-attention with several projection matrices, $W_q, W_k, W_v, W_o \in \mathbf{R}^{d\times d}$, and the corresponding biases $b_q, b_k, b_v, b_o \in \mathbf{R}^d$. With this notation, the computation of the CA residual block proceeds as follows. We first augment the patch embeddings (in matrix form) as $z = [x_{\text{class}}, x_{\text{patches}}]$, which makes the CA closer to a SA layer in that we guarantee that there is at least a key corresponding to the query. We then perform the projections:

$$Q = W_q\, x_{\text{class}} + b_q, \tag{5}$$
$$K = W_k\, z + b_k, \tag{6}$$
$$V = W_v\, z + b_v. \tag{7}$$

The class-attention weights are given by

$$A = \text{Softmax}(Q.K^T/\sqrt{d/h}) \tag{8}$$

where $Q.K^T \in \mathbf{R}^{h\times 1\times p}$. This attention is involved in the weighted sum $A \times V$ to produce the residual output vector

$$\mathrm{out}_{\mathrm{CA}} = W_o\, A\, V + b_o, \tag{9}$$

which is in turn added to x_{class} for subsequent processing.

The CA layers extract the useful information from the patches embedding to the class embedding. In preliminary experiments, we empirically observed that the first CA and FFN give the main boost, and a set of 2 blocks of layers (2 CA and 2 FFN) is sufficient to cap the performance. In the experimental section, we denote by 12+2 a transformer when it consists of 12 blocks of SA+FFN layers and 2 blocks of CA+FFN layers.

Complexity. The layers contain the same number of parameters in the class-attention and self-attention stages: CA is identical to SA in that respect, and we use the same parametrization for the FFNs. However the processing of these layers is much faster: the FFN only processes matrix-vector multiplications.

The CA function is also less expensive than SA in term of memory and computation because it computes the attention between the class vector and the set of patch embeddings: $Q \in \mathbf{R}^d$ means that $Q.K^T \in \mathbf{R}^{h\times 1\times p}$. In contrast, in the "regular self-attention" layers SA, we have $Q \in \mathbf{R}^{p\times d}$ and therefore $Q.K^T \in \mathbf{R}^{h\times p\times p}$. In other words, the initially quadratic complexity in the number of patches becomes linear in our extra CaiT layers.

4. Experiments

In this section, we report our experimental results related to LayerScale and CaiT. Note that we provide complementary results in supplemental material: in Appendix A we provide some experiments that have guided our architecture design and hyper-parameter optimization. Experiment in Transfer learning are reported in Appendix C and we show additional visualizations in Appendix D.

Experimental setting. Our implementation is based on the Timm library [69]. Unless specified otherwise, for this analysis we make minimal changes to hyper-parameters compared to the DeiT training scheme [64], except for minor adjustments of the learning rate in case we do not observe convergence. All our experiments are carried out on ImageNet [55], and also evaluated on two variations of it: ImageNet-Real [6] that corrects and give a more detailed annotation, and ImageNet-V2 [53] (matched frequency) that provides a separate test set.

4.1. Preliminary analysis with deeper architectures

We first carry out an empirical study of the normalization methods discussed in Section 2. At this stage we consider a

Table 1. **Improving convergence at depth** on ImageNet-1k. The baseline is DeiT-S. Several methods include a fix scalar learnable weight α per layer as in Figure 1(c). We have adapted Rezero, Fixup, T-Fixup, since the original methods do not converge: we have re-introduced the Layer-normalization η and warmup. We have adapted the drop rate d_r for all the methods, including the baseline (otherwise it does not converge). The column $\alpha = \varepsilon$ reports the performance when initializing the scalar with the same value as for LayerScale.

depth	baseline	scalar α weighting				LayerScale
	$[d_r]$	Rezero	T-Fixup	Fixup	$\alpha=\varepsilon$	
12	79.9 [*0.05*]	78.3	79.4	80.7	80.4	80.5
18	80.7 [*0.10*]	80.1	81.7	82.0	81.6	81.7
24	81.0 [*0.20*]	80.8	81.5	82.3	81.1	82.4
36	81.9 [*0.25*]	81.6	82.1	82.4	81.6	82.9

Deit-Small model during 300 epochs to allow a direct comparison with the results reports by Touvron *et al.* [64]. For all the variants we adjust the drop-rate of stochastic depth to the depth of the network, see Appendix A for a more detailed discussion. This is required to achieve convergence when increasing the depth to 36 layers. We measure the performance on the Imagenet1k [17, 55] classification dataset as a function of the depth.

4.1.1 Comparison of normalization strategies

As discussed in Section 2, Rezero, Fixup and T-Fixup do not converge when training DeiT off-the-shelf. However, if we re-introduce LayerNorm[2] and warmup, Fixup and T-Fixup achieve congervence and even improve training compared to the baseline DeiT. We report the results for these "adaptations" of Fixup and T-Fixup in Table 1.

The modified methods are able to converge with more layers without saturating too early. ReZero converges, we show (column $\alpha = \varepsilon$) that it is better to initialize α to a small value instead of 0, as in LayerScale. Fixup and t-Fixup are competitive with LayerScale in the regime of a relatively low number of blocks (12–18). However, they are more complex than LayerScale: they employ different initialization rules depending of the type of layers, and they require more changes of the transformer architecture. Therefore we only use LayerScale in subsequent experiments, which is much simpler as it only makes a single change and is parametrized by a single hyper-parameter ε.

Note, all the methods have a beneficial effect on convergence and they tend to reduce the need for stochastic depth, therefore we adjust these drop rate accordingly per method. Figure 3 provides the performances as the function of the drop rate d_r for LayerScale. We empirically use the following formula to set up the

[2]Bachlechner *et al*. report that batchnorm is complementary to ReZero, while removing LayerNorm in the case of transformers.

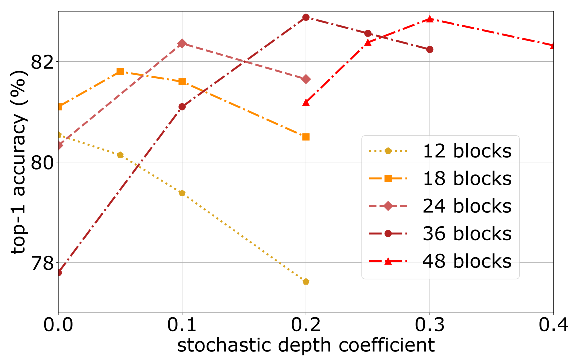


Figure 3. We measure the impact of stochastic depth on ImageNet with a DeiT-S with LayerScale for different depths. The drop rate of stochastic depth needs to be adapted to the network depth.

drop-rate for the CaiT-S models derived from on Deit-S: $d_r = \min\left(0.1 \times \frac{\text{depth}}{12} - 1, 0\right)$ .This formulaic choice avoids cross-validating this parameter and overfitting. We further increase (resp. decrease) it by a constant for larger (resp. smaller) working dimensionality d.

4.1.2 Analysis of Layerscale

Statistics of branch weighting. We evaluate the impact of Layerscale for a 36-layer transformer by measuring the ratio between the norm of the residual activations and the norm of the activations of the main branch $\|g_l(x)\|_2/\|x\|_2$. The results are shown in Figure 4. We can see that training a model with Layerscale makes this ratio more uniform across layers, and seems to prevent some layers from having a disproportionate impact on the activations. While translating this empirical observation to a provable conclusion, similar to prior works [2, 76] we hypothetize that the benefit is mostly the impact on optimization, which we try to support by the control experiment below.

Re-training. LayerScale makes it possible to get increased performance by training deeper models. At the end of training we obtain a specific set of scaling factors for each layer. Inspired by the lottery ticket hypothesis [23], one question that arises is whether what matters is to have the right scaling factors, or to include these learnable weights in the optimization procedure. In other terms, what happens if we re-train the network with the scaling factors obtained by a previous training?

The table below empirically answers that question.

Depth $\rightarrow$	12	18	24	36
LayerScale	80.5	81.7	82.4	82.9
Re-trained with fixed weights	80.6	81.5	81.2	81.6

In this experiment, we compare the performance (top-1 validation accuracy, %) on ImageNet-1k with DeiT-S architectures of differents depths. Everything being identical otherwise, in the first experiment we use LayerScale, i.e. we have learnable weights initialized at a small value ε. In the control experiment we use fixed scaling factors initialised at values obtained by the LayerScale training.

We can see that the control training with fixed weights also converges without suffering from instabilities with the deepest architectures. Nevertheless, the results are lower than those obtained with the learnable weighting factors. This suggests that the evolution of the parameters during training has a beneficial effect on the deepest models.

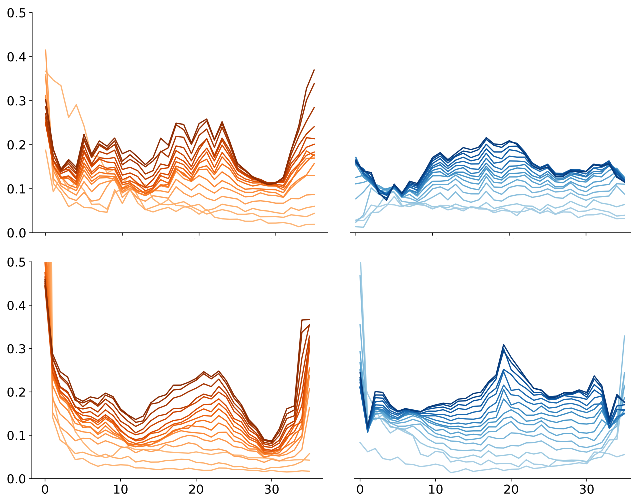


Figure 4. Analysis of the contribution of the residual branches (*Top:* Self-attention ; *Bottom:* FFN) for a network comprising 36 layers, without (red) or with (blue) Layerscale. The ratio between the norm of the residual and the norm of the main branch is shown for each layer of the transformer and for various epochs (darker shades correspond to the last epochs). For the model trained with layerscale, the norm of the residual branch is on average 20% of the norm of the main branch. We observe that the contribution of the residual blocks fluctuates more for the model trained without layerscale and in particular is lower for some of the deeper layers.

4.2. Class-attention layers

In Table 2 we study the impact on performance of the design choices related to class embedding. We depict some of them in Figure 2. As a baseline, average pooling of patches embeddings with a vanilla DeiT-Small achieves a better performance than using a class token. This choice, which does not employ any class embedding, is typical in convolutional networks, but possibly weaker with transformers when transferring to other tasks [20].

Late insertion. The performance increases when we insert the class embedding later in the transformer. It is maximized two layers before the output. Our interpretation is that the attention process is less perturbed in the 10 first lay-

Table 2. Variations on CLS with Deit-Small (no LayerScale): we change the layer at which the class embedding is inserted. In ViT and DeiT, it is inserted at layer 0 jointly with the projected patches. We evaluate a late insertion of the class embedding, as well as our design choice to introduce specific class-attention layers.

depth: SA+CA	insertion layer	top-1 acc.	#params	FLOPs
Baselines: DeiT-S and average pooling				
12: 12 + 0	0	79.9	22M	4.6B
12: 12 + 0	n/a	80.3	22M	4.6B
Late insertion of class embedding				
12: 12 + 0	2	80.0	22M	4.6B
12: 12 + 0	4	80.0	22M	4.6B
12: 12 + 0	8	80.0	22M	4.6B
12: 12 + 0	10	80.5	22M	4.6B
12: 12 + 0	11	80.3	22M	4.6B
DeiT-S with class-attention stage (SA+FFN)				
12: 9 + 3	9	79.6	22M	3.6B
12: 10 + 2	10	80.3	22M	4.0B
12: 11 + 1	11	80.6	22M	4.3B
13: 12 + 1	12	80.8	24M	4.7B
14: 12 + 2	12	80.8	26M	4.7B
15: 12 + 3	12	80.6	27M	4.8B

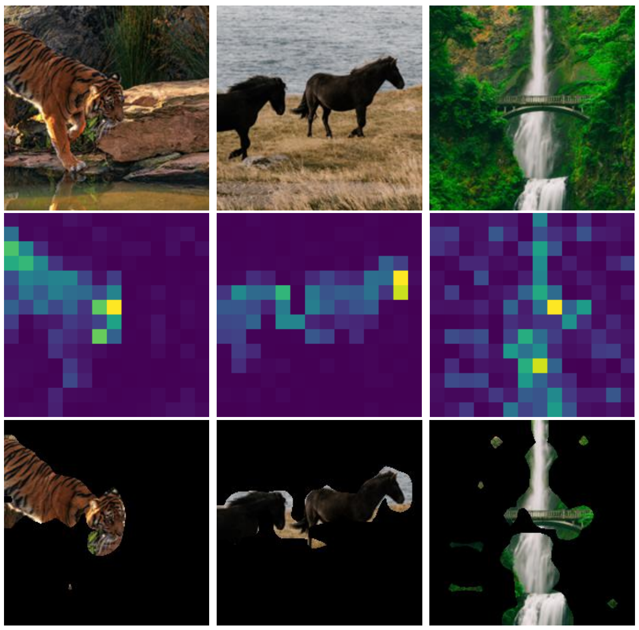


Figure 5. Visualization of the class-attention (first CA layer) obtained with a CaiT XXS-12 model. We show the attention map for each head in appendix. *top:* original image; *middle:* attention between all patches and class token; *bottom:* thresholded map.

ers, yet it is best to keep 2 layers for compiling the patches embedding into the class embedding via the class-attention, otherwise the processing gets closer to a weighted average.

Our class-attention layers are designed on the assumption that there is no benefit in copying information from the class embedding back to the patch embeddings in the forward pass. Table 2 supports that hypothesis: if we compare the performance for a total number of layers fixed to 12, the performance of CaiT with 10 SA and 2 CA layers is identical to average pooling and better than the DeiT-Small baseline with a lower number of FLOPs. If we set 12 layers in the self-attention stage, which dominates the complexity, we increase the performance significantly by adding two blocks of CA+FFN.

Visualization of class-attention maps. In Figure 5 we show the attention map related to our first class-attention layer. In CaiT, the first class-attention layer conveniently concentrates all the spatial-class relationship. The second class-attention layer seems to focus more on the context, see Appendix D for complementary visualizations.

4.3. Our CaiT models

Our CaiT models are built upon ViT: the only difference is that we incorporate LayerScale in each residual block (see Section 2) and the two-stages architecture with class-attention layers described in Section 3. Table 3 describes our different models. The design parameters governing the capacity are the depth and the working dimensionality d. In our case is related to the number of heads h as $d = 48 \times h$, since we fix the **number of components per head** to 48. This choice is a bit smaller than the value used in DeiT. We also adopt the **crop-ratio** of 1.0 optimized for DeiT by Wightman [69]. Table A.4 and A.5 in Appendix support these choices.

We incorporate talking-heads attention [56] into our model. It increases the performance on Imagenet of DeiT-Small from 79.9% to 80.3%.

The hyper-parameters are identical to those provided in DeiT [64], except mentioned otherwise. The main parameters are as follows: we use a batch size of 1024 samples and train during 400 epochs with repeated augmentation [5, 29]. The learning rate of the AdamW optimizer [44] is set to 0.001 and associated with a cosine training schedule, 5 epochs of warmup and a weight decay of 0.05.

We report in Table 3 the two hyper-parameters that we modify depending on the model complexity, namely the drop rate d_r associated with uniform stochastic depth, and the initialization value ε associated with LayerScale.

Fine-tuning at higher resolution ($\uparrow$) and distillation (Υ). We train all our models at resolution 224, and optionally fine-tune them at a higher resolution to trade performance against accuracy [19, 64, 65]: we denote the model by $\uparrow$384 models fine-tuned at resolution 384×384. We also train models with distillation (Υ) as suggested by Touvron *et*

Table 3. CaiT models: The design parameters are depth and d. The mem columns correspond to the memory usage. The speed is the throughput at inference time for a batch of 128 images with FP16 precision on one GPU V100 32GB (PyTorch 1.8 [48], CUDA 11). The only parameters that we adjust per model are the drop rate d_r of stochastic depth and the LayerScale initialization ε. All models are initially trained at resolution 224 during 400 epochs, the complexity measures (FLOPs, speed and mem) are reported for this resolution. We also fine-tune these models at resolution 384 (identified by $\uparrow$384) or train them with distillation (Υ).

CAIT	depth	d	#params	FLOPs ($\times 10^9$)		speed (im/s)		mem. (MB)		hparams		Top-1 acc. (%) on Imagenet1k-val			
model $\downarrow$	(SA+CA)		($\times 10^6$)	@224	@384	@224	@384	@224	@384	d_r	ε	@224	$\uparrow$384	@224Υ	$\uparrow$384Υ
XXS-24	24 + 2	192	12.0	2.5	9.6	1012.8	182.3	403	2126	0.1	10^{-5}	77.6	80.4	78.4	80.9
XXS-36	36 + 2	192	17.3	3.7	14.3	680.9	122.0	434	2156	0.1	10^{-6}	79.1	81.8	79.7	82.2
XS-24	24 + 2	288	26.6	5.4	19.3	737.6	134.8	614	3130	0.1	10^{-5}	81.8	83.8	82.0	84.1
XS-36	36 + 2	288	38.6	8.1	28.8	496.7	90.2	682	3198	0.2	10^{-6}	82.6	84.3	82.9	84.8
S-24	24 + 2	384	46.9	9.4	32.2	573.6	104.1	860	4165	0.1	10^{-5}	82.7	84.3	83.5	85.1
S-36	36 + 2	384	68.2	13.9	48.0	386.6	69.8	983	4287	0.2	10^{-6}	83.3	85.0	84.0	85.4
S-48	48 + 2	384	89.5	18.6	63.8	291.5	52.5	1106	4410	0.3	10^{-6}	83.5	85.1	83.9	85.3
M-24	24 + 2	768	185.9	36.0	116.1	262.9	38.3	2165	8634	0.2	10^{-5}	83.4	84.5	84.7	85.8
M-36	36 + 2	768	270.9	53.7	173.3	176.8	25.6	2661	9128	0.3	10^{-6}	83.8	84.9	85.1	86.1

Table 4. Ablation: we present the ablation path from DeiT-S to our CaiT models. We highlight the complementarity of our approaches and optimized hyper-parameters †: training failed.

Improvement	top-1 acc.	#params	FLOPs
DeiT-S [*d=384,300 epochs*]	79.9	22M	4.6B
+ More heads [*8*]	80.0	22M	4.6B
+ Talking-heads	80.5	22M	4.6B
+ Depth [*36 blocks*]	69.9†	64M	13.8B
+ Layer-scale [*init* $\varepsilon = 10^{-6}$]	80.5	64M	13.8B
+ Stochastic depth adaptation [d_r*=0.2*]	83.0	64M	13.8B
+ CA [CaiT]	83.2	68M	13.9B
+ Longer training [*400 epochs*]	83.4	68M	13.9B
+ Inference at higher resolution [*256*]	83.8	68M	18.6B
+ Fine-tuning at higher resolution [*384*]	84.8	68M	48.0B
+ Hard distillation [*teacher: RegNetY-16GF*]	85.2	68M	48.0B
+ Adjust crop ratio [*0.875 → 1.0*]	85.4	68M	48.0B

al. [64]. We use a RegNet-16GF [51] as teacher and adopt the simple "hard distillation" [64] for its simplicity.

4.4. Results

Table 3 provides different complexity measures for our models. As a general observation, we observe a subtle interplay between the width and the depth, both contribute to the performance as reported by Dosovitskiy *et al.* [19] with longer training schedules. But if one parameter is too small the gain brought by increasing the other is not worth the additional complexity.

Fine-tuning to size 384 ($\uparrow$) systematically offers a large boost in performance without changing the number of parameters. It also comes with a higher computational cost. In contrast, leveraging a pre-trained convnet teacher with hard distillation as suggested by Touvron *et al.* [64] provides a boost in accuracy without affecting the number of parameters nor the speed.

Comparison with the state of the art. In Table 5 we compare some of our models with the state of the art on Imagenet classification without external data. We focus on the models CaiT-S36 and CaiT-M36, at different resolutions and with or without distillation.

On Imagenet-val, we achieve 86.5% of top-1 accuracy, which is a significant improvement over DeiT (85.2%). It is the state of the art except for a very recent concurrent work [8] that also reports a top-1 accuracy of 86.5%. Our CaiT-M36$\uparrow$384Υ model obtains 85.9% top-1 accuracy on Imagenet-1k-val: this network is more efficient that the best performing NFNet convnets w.r.t. FLOPs and even more in terms of throughput (images processed per second on a V100 GPU), thanks to the lower memory usage.

Our approach is on par with the state of the art on Imagenet with reassessed labels, and outperforms it on Imagenet-V2, which has a distinct validation set which makes it harder to overfit.

Ablation. In Table 4 we present how to gradually transform the Deit-S [64] architecture to CaiT-36, and measure at each step the performance/complexity changes. One can see that CaiT is complementary with LayerScale and offers an improvement without significantly increasing the FLOPs. As already reported in the literature, the resolution is another important step for improving the performance and fine-tuning instead of training the model from scratch saves a lot of computation at training time. Last but not least, our models benefit from longer training schedules.

5. Related work

Since AlexNet [40], convolutional neural networks (CNN) are the standard in image classification [27, 63, 65], and more generally in computer vision. While a deep CNN can theoretically model long range interaction between pixels across many layers, there has been research in increas-

Table 5. **Complexity vs accuracy** on Imagenet [55], Imagenet Real [6] and Imagenet V2 matched frequency [53] for models trained without external data. We compare CaiT with DeiT [64], Vit-B [19], TNT [26], T2T [75] and to several state-of-the-art convnets: Regnet [51] improved by Touvron et al. [64], EfficientNet [14, 63, 72], Fix-EfficientNet [66] and NFNets [8]. Most reported results are from corresponding papers, and therefore the training procedure differs for the different models. For Imagenet V2 matched frequency and Imagenet Real we report the results provided by the authors. When not available (like NFNet), we report the results measured by Wigthman [69] with converted models, which may be suboptimal. The RegNetY-16GF is the teacher model that we trained for distillation. We report the best result in **bold** and the second best result(s) $\underline{\text{underlined}}$.

Network	nb of param.	nb of FLOPs	image size train	test	ImNet top-1	Real top-1	V2 top-1
RegNetY-16GF	84M	16B	224	224	82.9	88.1	72.4
EfficientNet-B5	30M	10B	456	456	83.6	88.3	73.6
EfficientNet-B7	66M	37B	600	600	84.3	_	_
EfficientNet-B5 RA	30M	10B	456	456	83.7	_	_
EfficientNet-B7 RA	66M	37B	600	600	84.7	_	_
Fix-EfficientNet-B8	87M	90B	672	800	85.7	$\underline{90.0}$	75.9
NFNet-F0	72M	12B	192	256	83.6	88.1	72.6
NFNet-F1	133M	36B	224	320	84.7	88.9	74.4
NFNet-F2	194M	62B	256	352	85.1	88.9	74.3
NFNet-F3	255M	115B	320	416	85.7	89.4	75.2
NFNet-F4	316M	215B	384	512	85.9	89.4	75.2
NFNet-F5	377M	290B	416	544	86.0	89.2	74.6
NFNet-F6+SAM	438M	377B	448	576	**86.5**	89.9	75.8
Transformers							
ViT-B/16	86M	55B	224	384	77.9	83.6	_
ViT-L/16	307M	191B	224	384	76.5	82.2	_
T2T-ViT t-14	21M	5B	224	224	80.7	_	_
TNT-S	24M	5B	224	224	81.3	_	_
TNT-S + SE	25M	5B	224	224	81.6	_	_
TNT-B	66M	14B	224	224	82.8	_	_
DeiT-S	22M	5B	224	224	79.8	85.7	68.5
DeiT-B	86M	18B	224	224	81.8	86.7	71.5
DeiT-B↑384	86M	55B	224	384	83.1	87.7	72.4
DeiT-B↑384Υ	87M	56B	224	384	85.2	89.3	75.2
Our deep transformers							
CaiT-S36	68M	14B	224	224	83.3	88.0	72.5
CaiT-S36↑384	68M	48B	224	384	85.0	89.2	75.0
CaiT-S48↑384	89M	64B	224	384	85.1	89.5	75.5
CaiT-S36Υ	68M	14B	224	224	84.0	88.9	74.1
CaiT-S36↑384Υ	68M	48B	224	384	85.4	89.8	76.2
CaiT-M36↑384Υ	271M	173B	224	384	86.1	$\underline{90.0}$	76.3
CaiT-M36↑448Υ	271M	248B	224	448	$\underline{86.3}$	**90.2**	$\underline{76.7}$
CaiT-M48↑448Υ	356M	330B	224	448	**86.5**	**90.2**	**76.9**

ing the range of interactions within a single layer. Some approaches adapt the receptive field of convolutions dynamically [15, 42]. At another end of the spectrum, attention can be viewed as a general form of non-local means, which was used in filtering (e.g. denoising [10]), and more recently in conjunction with convolutions [68]. Various other attention mechanism have been used successfully to give a global view in conjunction with (local) convolutions [4, 52, 12, 77, 79], most mimic *squeeze-and-excitate* [32] for leveraging global features. Lastly, LambdaNetworks [3] decomposes attention into an approximated content attention and a batch-amortized positional attention component.

Hybrid architectures combining CNNs and transformers blocks have also been used on ImageNet [58, 70] and on COCO [11]. Originally, transformers without convolutions were applied on pixels directly [47], even scaling to hundred of layers [13], but did not perform at CNNs levels. More recently, a transformer architecture working directly on small patches has obtained state of the art results on ImageNet [19]. Nevertheless, the state of the art has since returned to CNNs [8, 49]. While some small improvements have been applied on the transformer architecture with encouraging results [75], their performance is below the one of DeiT [64], which uses a vanilla ViT architecture.

Encoder/decoder architectures. Transformers were originally introduced for machine translation [67] with encoder-decoder models, and gained popularity as masked language model encoders (BERT) [18, 43]. They yielded impressive results as scaled up language models, e.g. GPT-2 and 3 [50, 9]. They became a staple in speech recognition too [45, 36], being it in encoder and sequence criterion or encoder-decoder seq2seq [61] conformations, and hold the state of the art to this day [74, 78] with models 36 blocks deep. Note, transforming only the class token with frozen trunk embeddings in CaiT is reminiscent of non-autoregressive encoder-decoders [25, 41], where a whole sequence (we have only one prediction) is produced at once by iterative refinements.

Deeper architectures usually lead to better performance [27, 57, 62], however this complicates their training process [59, 60]. One must adapt the architecture and the optimization procedure to train them correctly. Some approaches focus on the initialization schemes [24, 27, 71], others on multiple stages training [54, 57], multiple loss at different depth [62], adding components in the architecture [2, 76] or regularization [33]. As pointed in our paper, in that respect our LayerScale approach is more related to Rezero [2] and Skipinit [16], Fixup [76], and T-Fixup [34].

6. Conclusion

In this paper, we have shown how train deeper transformer-based image classification neural networks when training on Imagenet only. We have also introduced the simple yet effective CaiT architecture designed in the spirit of encoder/decoder architectures. Our work further demonstrates that transformer models offer a competitive alternative to the best convolutional neural networks when considering trade-offs between accuracy and complexity.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Thomas C. Bachlechner, Bodhisattwa Prasad Majumder, H. H. Mao, G. Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. *arXiv preprint arXiv:2003.04887*, 2020.

[3] Irwan Bello. Lambdanetworks: Modeling long-range interactions without attention. In *International Conference on Learning Representations*, 2021.

[4] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le. Attention augmented convolutional networks. In *Conference on Computer Vision and Pattern Recognition*, 2019.

[5] Maxim Berman, Hervé Jégou, Andrea Vedaldi, Iasonas Kokkinos, and Matthijs Douze. Multigrain: a unified image embedding for classes and instances. *arXiv preprint arXiv:1902.05509*, 2019.

[6] Lucas Beyer, Olivier J. Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aaron van den Oord. Are we done with imagenet? *arXiv preprint arXiv:2006.07159*, 2020.

[7] Andrew Brock, Soham De, and Samuel L Smith. Characterizing signal propagation to close the performance gap in unnormalized resnets. *arXiv preprint arXiv:2101.08692*, 2021.

[8] A. Brock, Soham De, S. L. Smith, and K. Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.

[9] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[10] Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. In *Conference on Computer Vision and Pattern Recognition*, 2005.

[11] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, 2020.

[12] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Conference on Computer Vision and Pattern Recognition*, 2020.

[13] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[14] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. *arXiv preprint arXiv:1909.13719*, 2019.

[15] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Conference on Computer Vision and Pattern Recognition*, 2017.

[16] Soham De and Samuel L Smith. Batch normalization biases residual blocks towards the identity function in deep networks. *arXiv e-prints*, pages arXiv–2002, 2020.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[19] 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. *International Conference on Learning Representations*, 2021.

[20] Alaaeldin El-Nouby, Natalia Neverova, Ivan Laptev, and Hervé Jégou. Training vision transformers for image retrieval. *arXiv preprint arXiv:2102.05644*, 2021.

[21] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019. ICLR 2020.

[22] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Hervé Jégou, and Armand Joulin. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320*, 2020.

[23] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[24] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[25] Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017.

[26] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *arXiv preprint arXiv:2103.00112*, 2021.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, June 2016.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

[29] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *Conference on Computer Vision and Pattern Recognition*, 2020.

[30] Grant Van Horn, Oisin Mac Aodha, Yang Song, Alexander Shepard, Hartwig Adam, Pietro Perona, and Serge J. Belongie. The inaturalist challenge 2018 dataset. *arXiv preprint arXiv:1707.06642*, 2018.

[31] Grant Van Horn, Oisin Mac Aodha, Yang Song, Alexander Shepard, Hartwig Adam, Pietro Perona, and Serge J. Belongie. The inaturalist challenge 2019 dataset. *arXiv preprint arXiv:1707.06642*, 2019.

[32] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[33] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, 2016.

[34] Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *International Conference on Machine Learning*, pages 4475–4483. PMLR, 2020.

[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

[36] Shigeki Karita, Nanxin Chen, Tomoki Hayashi, et al. A comparative study on transformer vs rnn in speech applications. *arXiv preprint arXiv:1909.06317*, 2019.

[37] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

[38] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, 2013.

[39] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, CIFAR, 2009.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[41] Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. *arXiv preprint arXiv:1802.06901*, 2018.

[42] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks. *Conference on Computer Vision and Pattern Recognition*, 2019.

[43] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

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

[45] Christoph Lüscher, Eugen Beck, Kazuki Irie, et al. Rwth asr systems for librispeech: Hybrid vs attention. *Interspeech 2019*, Sep 2019.

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

[47] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR, 2018.

[48] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[49] H. Pham, Qizhe Xie, Zihang Dai, and Quoc V. Le. Meta pseudo labels. *arXiv preprint arXiv:2003.10580*, 2020.

[50] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners.

[51] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. *Conference on Computer Vision and Pattern Recognition*, 2020.

[52] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, I. Bello, Anselm Levskaya, and Jonathon Shlens. Stand-alone self-attention in vision models. In *Neurips*, 2019.

[53] B. Recht, Rebecca Roelofs, L. Schmidt, and V. Shankar. Do imagenet classifiers generalize to imagenet? *arXiv preprint arXiv:1902.10811*, 2019.

[54] A. Romero, Nicolas Ballas, S. Kahou, Antoine Chassang, C. Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2015.

[55] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International journal of Computer Vision*, 2015.

[56] Noam Shazeer, Zhenzhong Lan, Youlong Cheng, N. Ding, and L. Hou. Talking-heads attention. *arXiv preprint arXiv:2003.02436*, 2020.

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

[58] A. Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, P. Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. *arXiv preprint arXiv:2101.11605*, 2021.

[59] R. Srivastava, Klaus Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[60] R. Srivastava, Klaus Greff, and J. Schmidhuber. Training very deep networks. In *NIPS*, 2015.

[61] Gabriel Synnaeve, Qiantong Xu, Jacob Kahn, Tatiana Likhomanenko, Edouard Grave, Vineel Pratap, Anuroop Sriram, Vitaliy Liptchinsky, and Ronan Collobert. End-to-end asr: from supervised to semi-supervised learning with modern architectures. *arXiv preprint arXiv:1911.08460*, 2019.

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

[63] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[64] Hugo Touvron, M. Cord, M. Douze, F. Massa, Alexandre Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020.

[65] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Herve Jegou. Fixing the train-test resolution discrepancy. *Neurips*, 2019.

[66] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy: Fixefficientnet. *arXiv preprint arXiv:2003.08237*, 2020.

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[68] X. Wang, Ross B. Girshick, A. Gupta, and Kaiming He. Non-local neural networks. *Conference on Computer Vision and Pattern Recognition*, 2018.

[69] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[70] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Masayoshi Tomizuka, Kurt Keutzer, and Peter Vajda. Visual transformers: Token-based image representation and processing for computer vision. *arXiv preprint arXiv:2006.03677*, 2020.

[71] L. Xiao, Y. Bahri, Jascha Sohl-Dickstein, S. Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10, 000-layer vanilla convolutional neural networks. *arXiv preprint arXiv:1806.05393*, 2018.

[72] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, A. Yuille, and Quoc V. Le. Adversarial examples improve image recognition. *Conference on Computer Vision and Pattern Recognition*, 2020.

[73] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *Conference on Computer Vision and Pattern Recognition*, 2017.

[74] Qiantong Xu, Alexei Baevski, Tatiana Likhomanenko, Paden Tomasello, Alexis Conneau, Ronan Collobert, Gabriel Synnaeve, and Michael Auli. Self-training and pre-training are complementary for speech recognition. *arXiv preprint arXiv:2010.11430*, 2020.

[75] L. Yuan, Y. Chen, Tao Wang, Weihao Yu, Yujun Shi, F. Tay, Jiashi Feng, and S. Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. *arXiv preprint arXiv:2101.11986*, 2021.

[76] Hongyi Zhang, Yann Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

[77] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi-Li Zhang, Haibin Lin, Yu e Sun, Tong He, Jonas Mueller, R. Manmatha, M. Li, and Alex Smola. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020.

[78] Yu Zhang, James Qin, Daniel S Park, Wei Han, Chung-Cheng Chiu, Ruoming Pang, Quoc V Le, and Yonghui Wu. Pushing the limits of semi-supervised learning for automatic speech recognition. *arXiv preprint arXiv:2010.10504*, 2020.

[79] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. In *Conference on Computer Vision and Pattern Recognition*, 2020.

Going deeper with Image Transformers

Supplementary Material

In this supplemental material, we first provide in Section A experiments that have guided our architecture design and hyper-parameter optimization. As mentioned in Introduction and our experimental section 4, we give the details of our experiments in Transfer learning in Appendix C and we show additional visualizations in Appendix D.

As mentioned in the main paper, we use the same hyper-parameters as in DeiT [64] everywhere except stated otherwise. In order to speed up training and optimize memory consumption we have used a sharded training thank to the Fairscale library (`https://pypi.org/project/fairscale/`) with fp16 precision.

A. Complementary analysis

In this section we provide several analysis related to optimization or architectural choices.

A.1. Train deeper networks with stochastic depth

In our early experiments, we observe that Vision Transformers become increasingly more difficult to train when we scale architectures. Depth is one of the main source of instability. For instance the DeiT procedure fails to properly converge above 18 layers without adjusting hyper-parameters. In the DeiT setting, we observe that the factors that provoke divergence are also those that bring more capacity to the models: depth, width, and the number of embeddings processed by the transformers. The latter directly relates to the image input resolution in our case because we use a fixed patch size of 16×16 pixels in our experiments.

Divergence can happen lately in the process, possibly almost at the end of training, even after reaching a decent temporary training accuracy. When the training diverges, we typically observe a rapid augmentation of the train loss and then NaN values due to fp16 precision. If we use fp32 instead with exactly the same seed, we do not observe NaN values, but the performance degrades.

Adjusting the drop-rate of stochastic depth. The first step to improve convergence is to adapt the hyper-parameters that interact the most with depth, in particular Stochastic depth [33]. This method is already popular in NLP [21, 22] to train deeper architectures. For ViT, it was first proposed by Wightman *et al.* [69] in the Timm implementation, and subsequently adopted in DeiT [64]. The per-layer drop-rate depends linearly on the layer depth, but in our experiments this choice does not provide an advantage compared to the simpler choice of a uniform drop-rate d_r. In Table A.1 we show that the default stochastic depth of DeiT allows us to train up to 18 blocks of SA+FFN. After that the training becomes unstable. By increasing the drop-rate hyper-parameter d_r, the performance increases until 24 layers. It saturates at 36 layers and drops.

Table A.1. Performance when increasing the depth. We compare different strategies and report the top-1 accuracy (%) on ImageNet-1k for the DeiT training (Baseline) with and without adapting the stochastic depth rate d_r (uniform drop-rate), and DeiT modified with Rezero and variants that were not working for ResNeT [28]. †: *failed before the end of the training. If it reached a reasonable performance during training, we report the last number obtained before divergence.*

depth	Baseline	Baseline with stch. adapt [d_r]	Rezero	constant scaling	exclusive gating	shortcut-only gating	linear shortcut	dropout shortcut
12	79.9	79.9 [*0.05*]	†	†	79.7	80.4	†	†
18	80.1	80.7 [*0.10*]	†	†	†	†	†	†
24	78.9†	81.0 [*0.20*]	†	†	†	†	†	†
36	78.9†	81.9 [*0.25*]	†	†	†	†	†	†
48	78.4†	80.7 [*0.30*]	†	†	†	†	†	†

A.2. Residual block normalization: negative results

In Table A.1 we also consider different architectural variants that He et al. [28] showed not to be working for ResNet, but this time we evaluate their interest for image transformers. As one can see, except for the exclusive gating method, all the attempts we have done with the ViT architecture fail to converge. As discussed in the main paper, the Rezero [2] method gives some good results if we re-introduce the warmup and Layer-normalization as a pre-norm operator.

A.3. Design of the class-attention stage

In this subsection we report some results obtained when considering alternative choices for the class-attention stage.

Table A.2. CaiT models with and without distillation token. All these models are trained with the same setting during 400 epochs.

Model	Distillation token ✗	Distillation token ✓
XXS-24Υ	78.4	78.5
M-24Υ	84.8	84.7

Not including class embedding in keys of class-attention. In our approach we chose to insert the class embedding in the class-attention: By defining

$$z = [x_{\text{class}}, x_{\text{patches}}], \tag{A.1}$$

we include x_{class} in the keys and therefore the class-attention includes attention on the class embedding itself in Eqn. 6 and Eqn. 7. This is not a requirement as we could simply use a pure cross-attention between the class embedding and the set of frozen patches.

If we do not include the class token in the keys of the class-attention layers, i.e., if we define $z = x_{\text{patches}}$, we reach 83.31% (top-1 acc. on ImageNet1k-val) with CaiT-S-36, versus 83.44% for the choice adopted in our main paper. This difference of +0.13% is likely not significant, therefore either choice is reasonable. In order to be more consistent with the self-attention layer SA, in the sense that each query has its key counterpart, we have kept the class embedding in the keys of the CA layers as stated in our paper.

Remove LayerScale in Class-Attention. If we remove LayerScale in the Class-Attention blocks in the CaiT-S-36 model, we reach 83.36% (top-1 acc. on ImageNet1k-val) versus 83.44% with LayerScale. The difference of +0.08% is not significant enough to conclude on a clear advantage. For the sake of consistency we have used LayerScale after all residual blocks of the network.

Distillation with class-attention In the main paper we report results with the hard distillation proposed by Touvron *et al.* [64], which in essence replaces the label by the average of the label and the prediction of the teacher output. This is the choice we adopted in our main paper, since it provides better performance than traditional distillation.

The DeiT authors also show the advantage of considering an additional "distillation token". In their case, employed with the ViT/DeiT architecture, this choice improves the performance compared to hard distillation. Noticeably it accelerates convergence.

In Table A.2 we report the results obtained when inserting a distillation token at the same layer as the class token, i.e., on input of the class-attention stage. In our case we do not observe an advantage of this choice over hard distillation when using class-attention layers. Therefore we have only considered hard distillation in our paper.

Table A.3. Comparison between different initialisation of the diagonal matrix for LayerScale. We report results with 0 initialization, Uniform initialisation and small constant initialisation. For all approaches (including the DeiT-S baseline), we have adapted the stochastic depth rate d_r.

depth	baseline $[d_r]$	ReZero $\alpha = 0$	LayerScale $[\varepsilon]$ $\lambda_i = 0$	LayerScale $[\varepsilon]$ $\lambda_i = \mathcal{U}[0, 2\varepsilon]$	LayerScale $[\varepsilon]$ $\lambda_i = \varepsilon$
12	79.9 [*0.05*]	78.3	79.7	80.2 [*0.1*]	80.5 [*0.1*]
18	80.7 [*0.10*]	80.1	81.5	80.8 [*0.1*]	81.7 [*0.1*]
24	81.0 [*0.20*]	80.8	82.1	82.1 [10^{-5}]	82.4 [10^{-5}]
36	81.9 [*0.25*]	81.6	82.7	82.6 [10^{-6}]	82.9 [10^{-6}]

A.4. Variations on LayerScale init

For the sake of simplicity and to avoid overfitting per model, we have chosen to do a constant initialization with small values depending on the model depth. In order to give additional insight on the importance of this initialization we compare in Table A.3 other possible choices.

LayerScale with 0 init. We initialize all coefficients of LayerScale to 0. This resembles Rezero, but in this case we have distinct learnable parameters for each channel. We make two observations. First, this choice, which also starts with residual branches that output 0 the beginning of the training, gives a clear boost compared to the block-wise scaling done by our adapted ReZero. This confirms the advantage of introducing a learnable parameter per channel and not only per residual layer. Second, LayerScale is better: it is best to initialize to a small ε different from zero.

Random init. We have tested a version in which we try a different initial weight per channel, but with the same average contribution of each residual block as in LayerScale. For this purpose we initialize the channel-scaling values with the Uniform law ($\mathcal{U}[0, 2\varepsilon]$). This simple choice choice ensures that the expectation of the scaling factor is equal to the value of the classical initialization of LayerScale. This choice is overall comparable to the initialization to 0 of the diagonal, and inferior to LayerScale.

A.5. Optimization of the number of heads

In Table A.4 we study the impact of the number of heads for a fixed working dimensionality. This architectural parameter has an impact on both the accuracy, and the efficiency: while the number of FLOPs remain roughly the same, the compute is more fragmented when increasing this number of heads and on typical hardware this leads to a lower effective throughput. Choosing 8 heads in the self-attention offers a good compromise between accuracy and speed. In Deit-Small, this parameter was set to 6.

A.6. Adaptation of the crop-ratio

In the typical ("center-crop") evaluation setting, most convolutional neural networks crop a subimage with a given

Table A.4. Deit-Small: for a fixed 384 working dimensionality and number of parameters, impact of the number of heads on the accuracy and throughput (im/s at inference time on a V100).

# heads	dim/head	throughput	GFLOPs	top-1 acc.
1	384	1079	4.6	76.80
2	192	1056	4.6	78.06
3	128	1043	4.6	79.35
6	64	989	4.6	79.90
8	48	971	4.6	80.02
12	32	927	4.6	80.08
16	24	860	4.6	80.04
24	16	763	4.6	79.60

Table A.5. We compare performance with image transformers with the defaut crop-ratio of 0.875 usually used with convnets, and the simple crop-ratio of 1.0 [69]. Note that, except in this experiment, all the results in our paper and supplemental are reported with a crop ratio of 0.875.

Network	Crop Ratio		ImNet	Real	V2
	0.875	1.0	top-1	top-1	top-1
S36	✓	-	83.4	88.1	73.0
	-	✓	83.3	88.0	72.5
S36↑384	✓	-	84.8	88.9	74.7
	-	✓	85.0	89.2	75.0
S36Υ	✓	-	83.7	88.9	74.1
	-	✓	84.0	88.9	74.1
M36Υ	✓	-	84.8	89.2	74.9
	-	✓	84.9	89.2	75.0
S36↑384Υ	✓	-	85.2	89.7	75.7
	-	✓	85.4	89.8	76.2
M36↑384Υ	✓	-	85.9	89.9	76.1
	-	✓	86.1	90.0	76.3
M36↑448Υ	✓	-	86.0	89.9	76.5
	-	✓	86.2	90.2	76.5

ratio, typically extracting a 224 × 224 center crop from a 256 × 256 resized image, leading to the typical ratio of 0.875. Wightman *et al.* [69] notice that setting this crop ratio to 1.0 for transformer models has a positive impact: the distilled DeiT-B↑ 384 reach a top1-accuracy on Imagenet1k-val of 85.42% in this setting, which is a gain of +0.2% compared to the accuracy of 85.2% reported by Touvron *et al.* [64].

Our measurements concur with this observation: We observe a gain for almost all our models and most of the evaluation benchmarks. For instance our best model M36↑448Υ increases to 86.2% top-1 accuracy on Imagenet-val1k. Note that in our main paper, unless stated otherwise, the accuracy with our models is measured with a crop ratio of 0.875.

A.7. Longer training schedules

As shown in Table 4 , increasing the number of training epochs from 300 to 400 improves the performance of CaiT-S-36. However, increasing the number of training epochs from 400 to 500 does not change performance significantly (83.44 with 400 epochs 83.42 with 500 epochs). This is consistent with the observation of the DeiT [64] paper, which notes a saturation of performance from 400 epochs for the models trained without distillation.

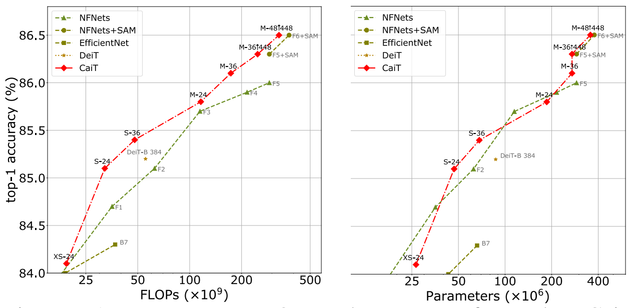


Figure B.1. We represent FLOPs and parameters for our best CaiT ↑384 and ↑448Υ models trained with distillation. They are competitive on ImageNet-1k-val with the sota in the high accuracy regime, from XS-24 to M-48. Convolution-based neural networks like NFNets and EfficientNet are better in low-FLOPS and low-parameters regimes.

B. More comparisons on Imagenet

Our main classification experiments are carried out on ImageNet [55], and also evaluated on two variations of this dataset: ImageNet-Real [6] that corrects and give a more detailed annotation, and ImageNet-V2 [53] (matched frequency) that provides a separate test set. In Table 5 we compare some of our models with the state of the art on Imagenet classification when training without external data. We focus on the models CaiT-S36 and CaiT-M36, at different resolutions and with or without distillation.

On Imagenet1k-val, CaiT-M48↑448Υ achieves 86.5% of top-1 accuracy, which is a significant improvement over DeiT (85.2%). It was the state of the art at the time of submission, on par with a recent concurrent work [8] that has a significantly higher number of FLOPs. Our approach outperforms the state of the art on Imagenet with reassessed labels, and on Imagenet-V2, which has a distinct validation set which makes it harder to overfit.

Table C.1. Datasets used for our different tasks.

Dataset	Train size	Test size	#classes
ImageNet [55]	1,281,167	50,000	1000
iNaturalist 2018 [30]	437,513	24,426	8,142
iNaturalist 2019 [31]	265,240	3,003	1,010
Flowers-102 [46]	2,040	6,149	102
Stanford Cars [38]	8,144	8,041	196
CIFAR-100 [39]	50,000	10,000	100
CIFAR-10 [39]	50,000	10,000	10

Table C.2. Results in transfer learning. All models are trained and evaluated at resolution 224 and with a crop-ratio of 0.875 in this comparison (see Table A.5 for the comparison of crop-ratio on Imagenet).

Model	ImageNet	CIFAR-10	CIFAR-100	Flowers	Cars	iNat-18	iNat-19
EfficientNet-B7	84.3	98.9	91.7	98.8	**94.7**	_	_
ViT-B/16	77.9	98.1	87.1	89.5	_	_	_
ViT-L/16	76.5	97.9	86.4	89.7	_	_	_
Deit-B 224	81.8	99.1	90.8	98.4	92.1	73.2	77.7
CaiT-S-36 224	83.4	99.2	92.2	98.8	93.5	77.1	80.6
CaiT-M-36 224	83.7	99.3	93.3	99.0	93.5	76.9	81.7
CaiT-S-36 Υ 224	83.7	99.2	92.2	99.0	94.1	77.0	81.4
CaiT-M-36 Υ 224	**84.8**	**99.4**	**93.1**	**99.1**	94.2	**78.0**	**81.8**

C. Transfer learning

We evaluated our method on transfer learning tasks by fine-tuning on the datasets in Table C.1.

Fine-tuning procedure. For fine-tuning we use the same hyperparameters as for training. We only decrease the learning rates by a factor 10 (for CARS, Flowers, iNaturalist), 100 (for CIFAR-100, CIFAR-10) and adapt the number of epochs (1000 for CIFAR-100, CIFAR-10, Flowers-102 and Cars-196, 360 for iNaturalist 2018 and 2019). We have not used distillation for this finetuning.

Results. Table C.2 compares CaiT transfer learning results to those of EfficientNet [63], ViT [19] and DeiT [64]. These results show the excellent generalization of the transformers-based models in general. Our CaiT models achieve excellent results, as shown by the overall better performance than EfficientNet-B7 across datasets.

D. Visualizations

D.1. Attention map

In Figure D.1 we show the attention maps associated with the individual 4 heads of a XXS CaiT model, and for the two layers of class-attention. In Figure 5 to save space we had only presented the first head and the first class-attention. We make two observations:

- The first class-attention layer clearly focuses on the object of interest, corresponding to the main part of the image on which the classification decision is performed (either correct or incorrect). In this layer, the different heads focus either on the same or on complementary parts of the objects. This is especially visible for the waterfall image;
- The second class-attention layer seems to focus more on the context, or at least the image more globally.

D.2. Illustration of saliency in class-attention

In figure D.2 we provide more vizualisations for a XXS model. They are just illustration of the saliency that one may extract from the first class-attention layer. As discussed previously this layer is the one that, empirically, is the most related to the object of interest. To produce these visual representations we simply average the attention maps from the different heads (depicted in Figure D.1), and upsample the resulting map to the image size. We then modulate the gray-level image with the strength of the attention after normalizing it with a simple rule of the form $(x - x_{\min})/(x_{\max} - x_{\min})$. We display the resulting image with `cividis` colormap.

For each image we show this saliency map and provides all the class for which the model assigns a probability higher than 10%. These visualizations illustrate how the model can focus on two distinct regions (like racket and tennis ball on the top row/center). We can also observe some failure cases, like the top of the church classified as a flagpole.

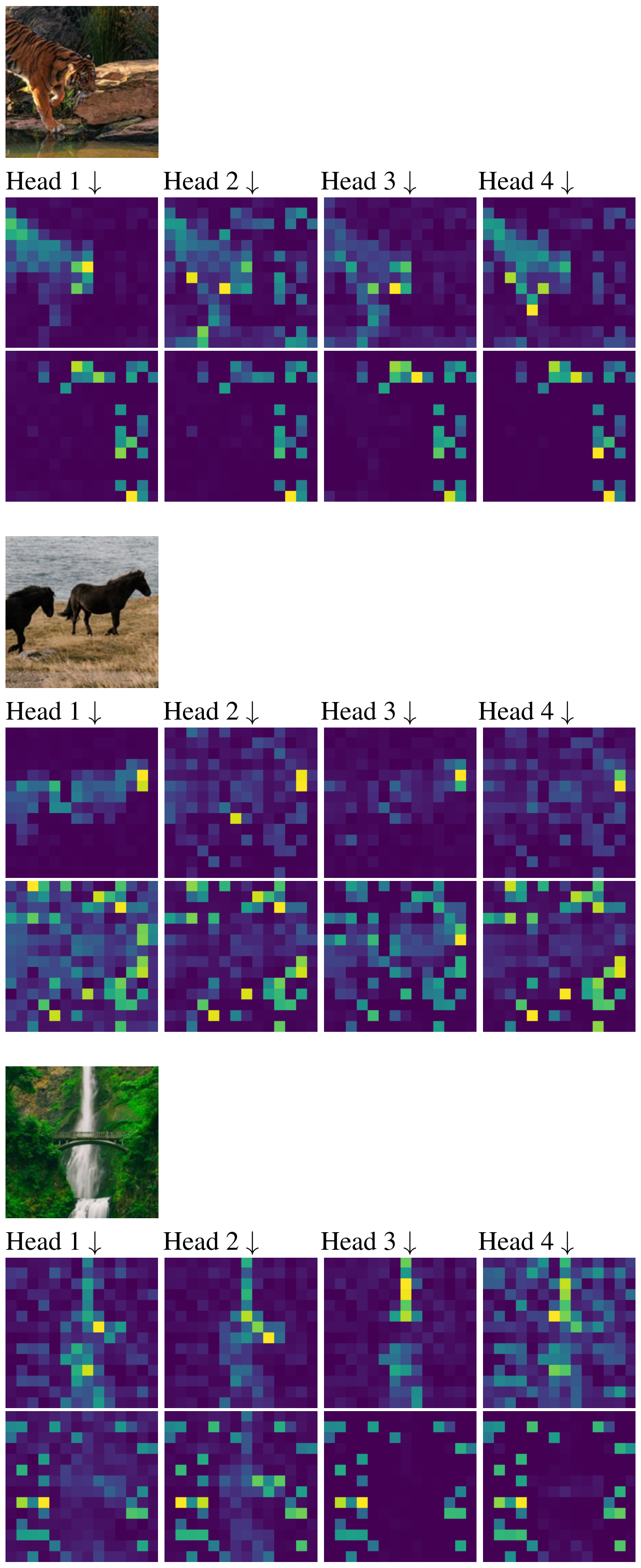


Figure D.1. Visualization of the attention maps in the class-attention stage, obtained with a XXS model. For each image we present two rows: the top row correspond to the four heads of the attention maps associated with the first CA layer. The bottom row correspond to the four heads of the second CA layer.

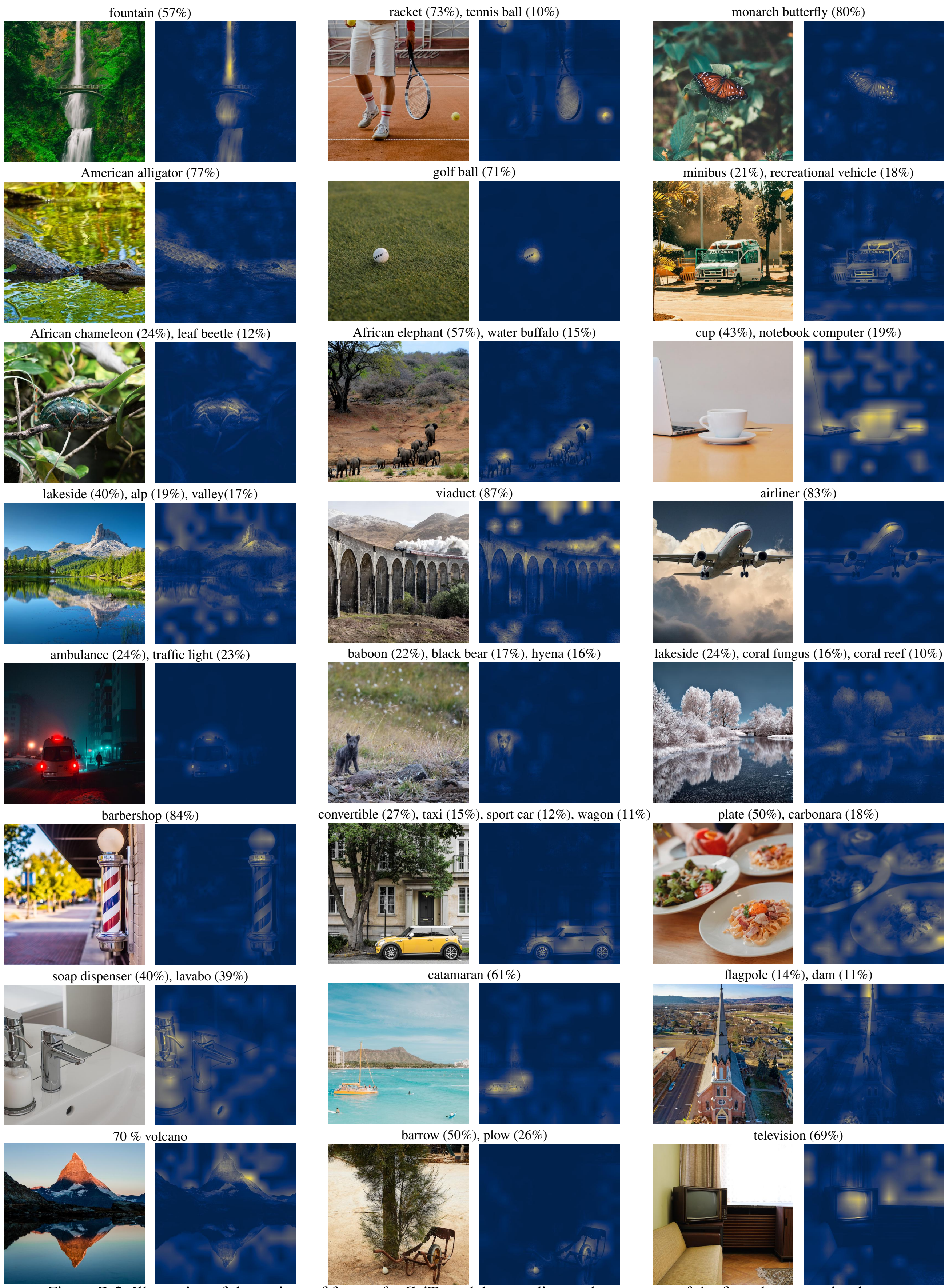

fountain (57%)
racket (73%), tennis ball (10%)
monarch butterfly (80%)
American alligator (77%)
golf ball (71%)
minibus (21%), recreational vehicle (18%)
African chameleon (24%), leaf beetle (12%)
African elephant (57%), water buffalo (15%)
cup (43%), notebook computer (19%)
lakeside (40%), alp (19%), valley(17%)
viaduct (87%)
airliner (83%)
ambulance (24%), traffic light (23%)
baboon (22%), black bear (17%), hyena (16%)
lakeside (24%), coral fungus (16%), coral reef (10%)
barbershop (84%)
convertible (27%), taxi (15%), sport car (12%), wagon (11%)
plate (50%), carbonara (18%)
soap dispenser (40%), lavabo (39%)
catamaran (61%)
flagpole (14%), dam (11%)
70 % volcano
barrow (50%), plow (26%)
television (69%)

Figure D.2. Illustration of the regions of focus of a CaiT model, according to the response of the first class-attention layer.