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 ✗	✓
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 [ε] $\lambda_i = 0$	$\lambda_i = \mathcal{U}[0, 2\varepsilon]$	$\lambda_i = \varepsilon$
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.

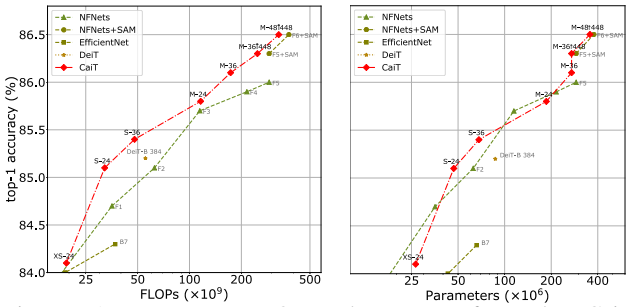

Figure B.1. We represent FLOPs and parameters for our best CaiT ↑384 and ↑448Υ models trained with distillation. They are competitive on ImageNet-1k-val with the sota in the high accuracy regime, from XS-24 to M-48. Convolution-based neural networks like NFNets and EfficientNet are better in low-FLOPS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

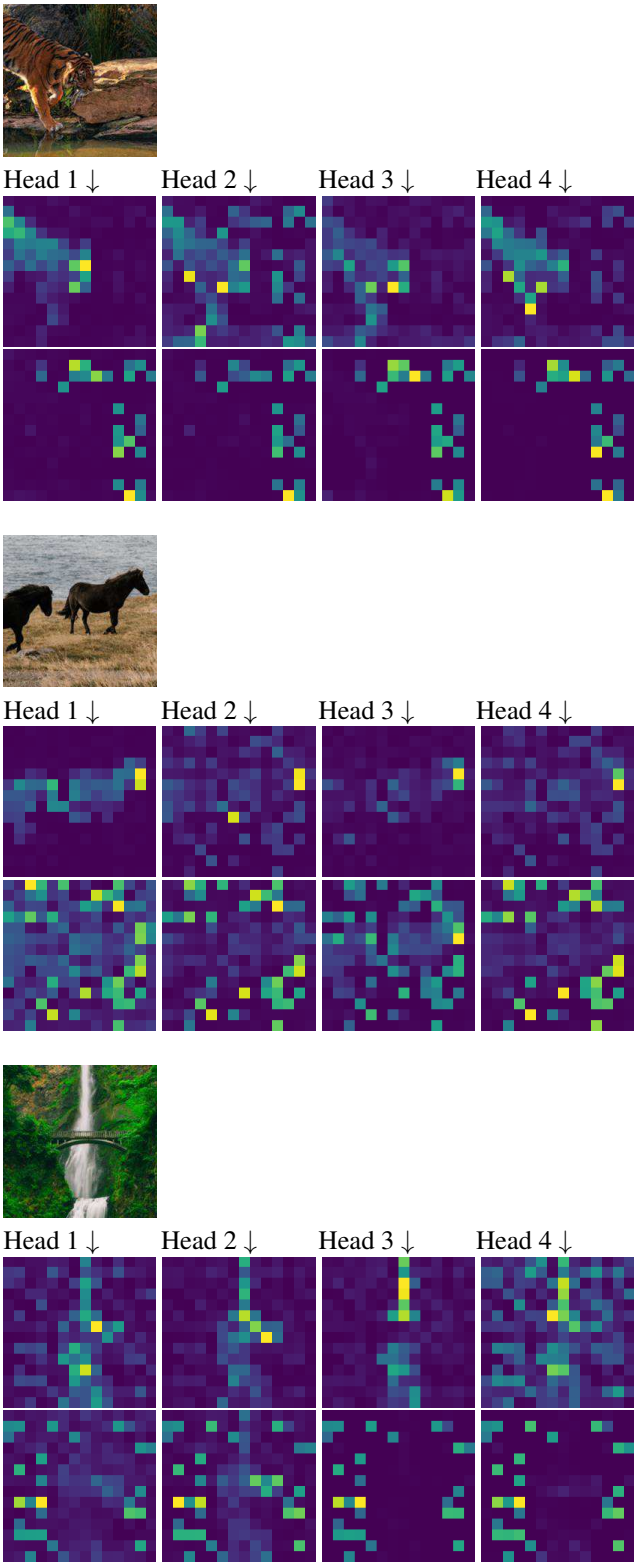

Figure D.1. Visualization of the attention maps in the class-attention stage, obtained with a XXS model. For each image we present two rows: the top row correspond to the four heads of the attention maps associated with the first CA layer. The bottom row correspond to the four heads of the second CA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

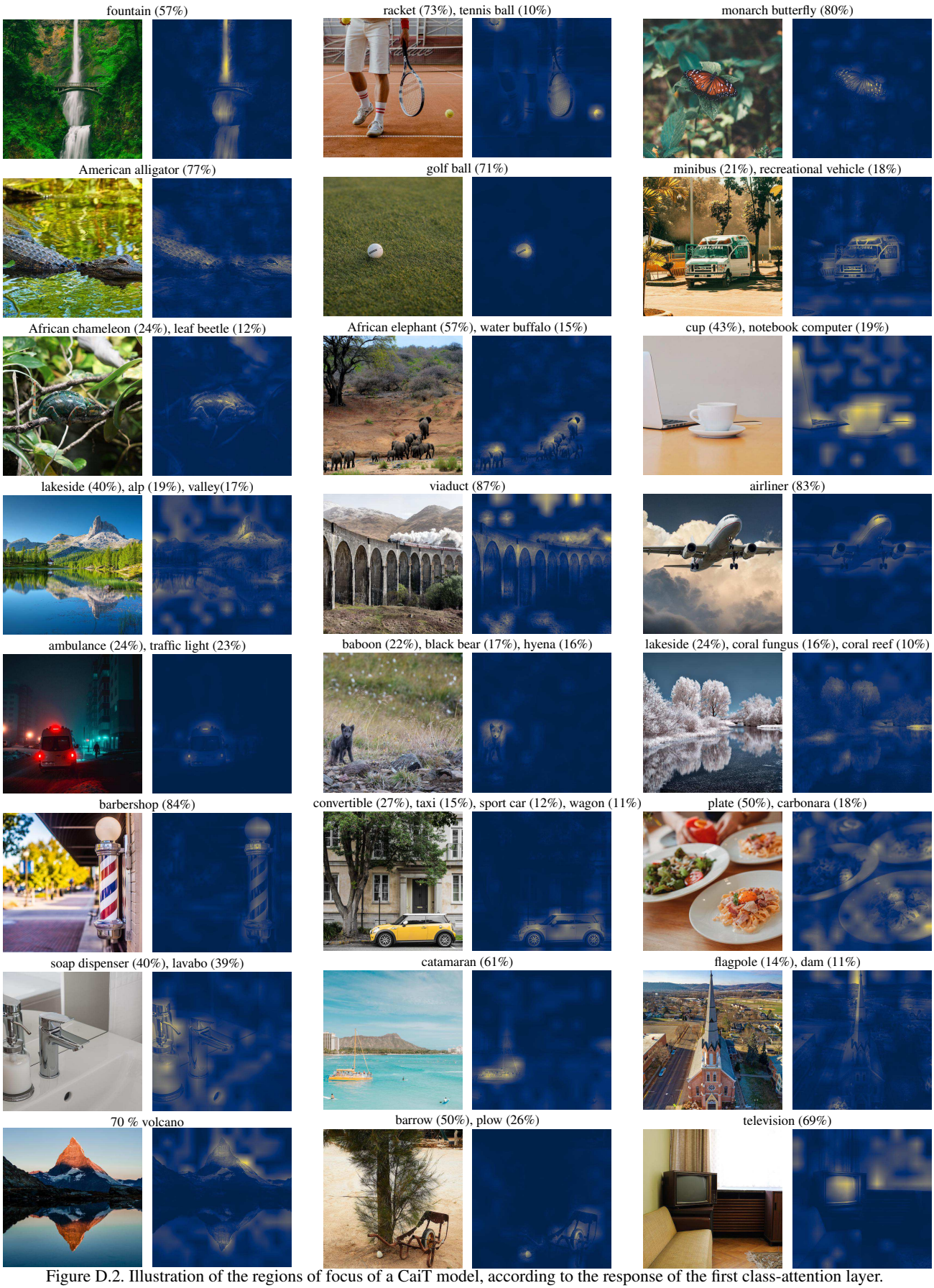

Figure D.2. Illustration of the regions of focus of a CaiT model, according to the response of the first class-attention layer.