Spatial Attention in DL Architectures

Overview of existing approaches
Mila Computer Vision Reading Group

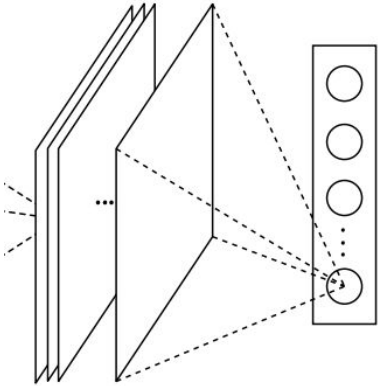
Oren Kraus - June 2022

Mind the GAP (Global Average Pool)

Global Average Pooling is a pooling operation designed to replace fully connected layers in classical CNNs. The idea is to generate one feature map for each corresponding category of the classification task in the last mlpconv layer.

Introduced by Lin et al. in Network In Network

[papers with code: global average pooling]

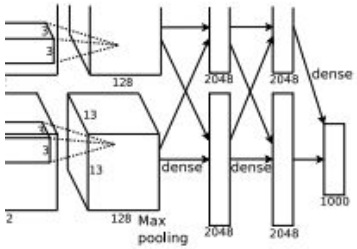


Discussion

- Why GAP?
- What alternatives are there?
- Is our use case different?

Background on GAP

- Replaced dense output layers in *Network in Network (ICLR, 2014)*
  - Introduced GAP (section 3.2) to replace dense layers in output
    - enforces “correspondences between feature maps and categories”
    - “no parameter to optimize in the global average pooling thus overfitting is avoided at this layer” - 0.39% improvement in Cifar10 error rate
    - GAP layer directly connected to cost - Class activation maps can be visualized
    - Also adopted by GooglenNet (CVPR, 2015), VGGNet, Resnet



AlexNet (2012)- w/ dense output layer

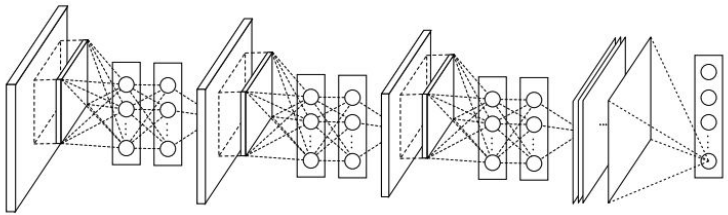
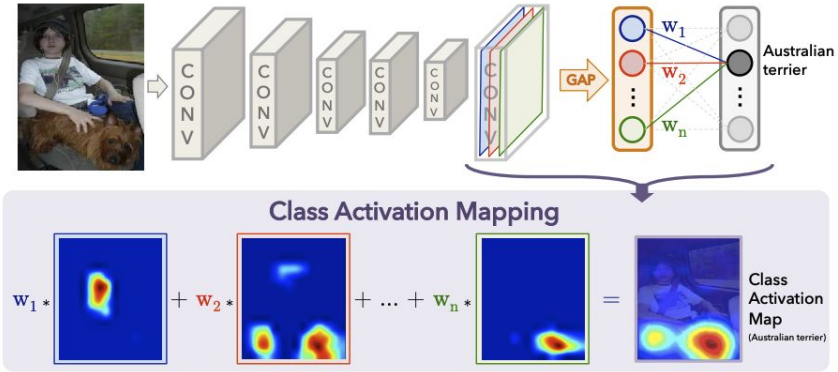


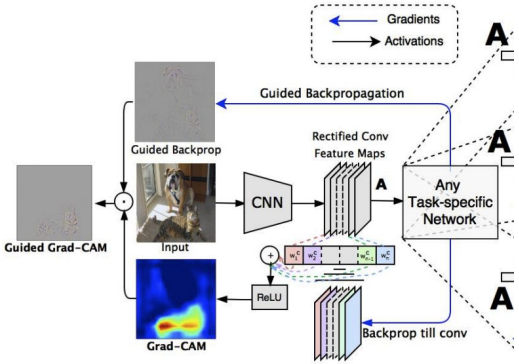
Figure 2: The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer.

Background on GAP

- [Learning Deep Features for Discriminative Localization](#) *(CVPR, 2016)*
  - Introduced class activation maps (**CAM**) derived from GAP layers
  - 37.1% top-5 test error weakly supervised object localization on ILSVRC benchmark
  - [Grad-Cam](#) extends this work by replacing the weight from the final classification layer with the GAP gradients to each feature map



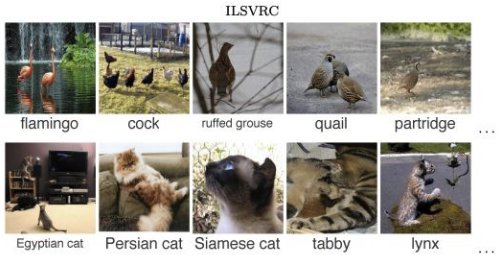
CAM



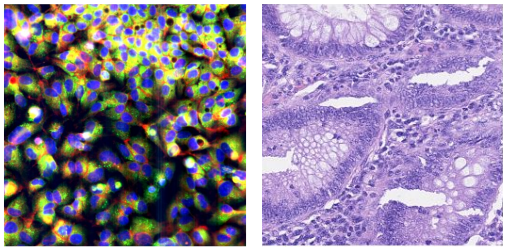
grad-CAM

[side-note]

- Most of these models were trained on and validated on Imagenet when developed. However, “Real-world scenes are likely to contain multiple instances of some objects, and nearby object instances are particularly difficult to delineate. The average object category in ILSVRC has 1.61 target object instances on average per positive image, with each instance having on average 0.47 neighbors (adjacent instances of the same object category).” (Russakovsky, O. *et al.* ImageNet Large Scale Visual Recognition Challenge, 2014)
- Is this the best approach for image datasets with lots of instances?



ImagNet



Cell microscopy and histology data

Background on Spatial Attention

- [Learn to Pay Attention](#) *(ICLR, 2018)*
  - Use global feature vector (***g***, i.e. post GAP vector) as query vector
  - Compute attention weights (a_i^s) for final conv feature maps based on compatibility score (c_i^s) based on dot product between ***g*** and spatial features (ℓ_i^s)
  - (g_a^s) are either concatenated or used for separate predictions that are aggregated
  - Improvements on CIFAR-10/100, CUB-200-2011, SVHN

compatibility score: $c_i^s = \langle \ell_i^s, g \rangle,\ i \in \{1 \cdots n\}.$

attention weights: $a_i^s = \frac{\exp(c_i^s)}{\sum_j^n \exp(c_j^s)},\ i \in \{1 \cdots n\}.$

Weighted features: $g_a^s = \sum_{i=1}^n a_i^s \cdot \ell_i^s$

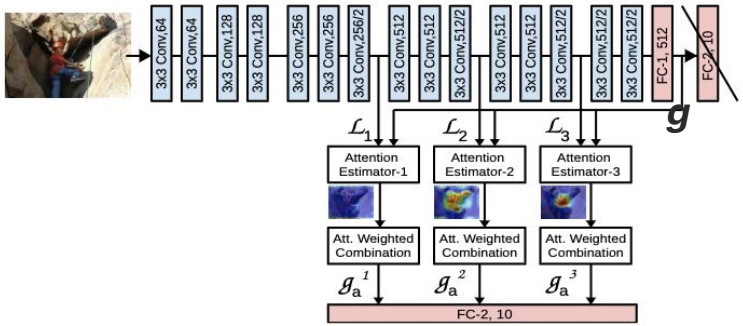


Figure 2: Attention introduced at 3 distinct layers of VGG. Lowest level attention maps appear to focus on the surroundings (*i.e.*, the rocky mountain), intermediate level maps on object parts (*i.e.*, harness and climbing equipment) and the highest level maps on the central object.

Background on Spatial Attention

- [Spatially Attentive Output Layer for Image Classification](#) *(CVPR, 2020)*
  - Compute spatial attention map final conv layer
  - Output is sum of spatial logits (class specific predictions) weighted by attention map
  - Introduce two additional losses - distillation based on GAP and SSL based on cutmix

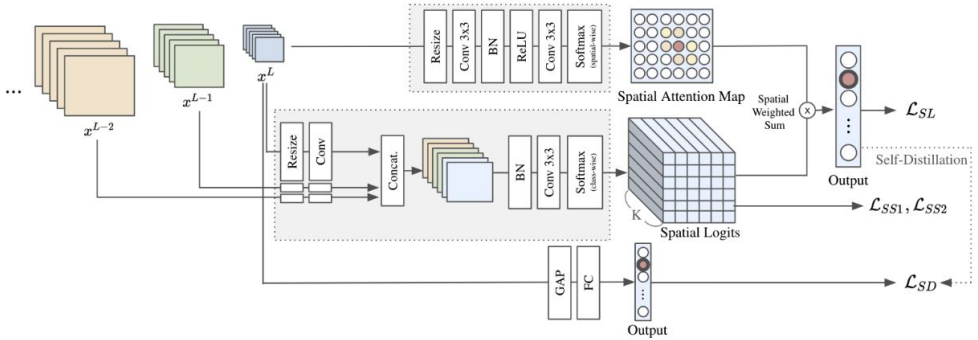


Figure 2: The detailed structure of the proposed SAOL. It produces the spatial attention map and spatial logits, separately. Note that we use additional self-annotated spatial labels to leverage our architecture further. We can also train the conventional GAP-FC based output layer jointly, using self-distillation.

Attention map

$$\mathbf{A} \in [0,1]^{H_o \times W_o} \qquad \sum_{i,j} \mathbf{A}_{ij} = 1, \forall k$$

Spatial logits

$$\mathbf{Y} \in [0,1]^{K \times H_o \times W_o} \qquad \sum_k (\mathbf{Y}_k)_{ij} = 1, \forall i, j.$$

Output

$$\hat{\mathbf{y}}_k = O_{\text{SAOL},k}(\mathbf{X}^L) = \sum_{i,j} \mathbf{A}_{ij} (\mathbf{Y}_k)_{ij}, \quad \forall k,$$

Main paper

- [Augmenting Convolutional networks with attention-based aggregation](#) *(arxiv, Dec 2021)*
  - Replace final pooling layer with attention based pooling
  - Can support multiple class attention maps by using multiple class tokens
  - Propose CNN arch (PatchConvNet) to retain spatial resolution in conv
  - Outperforms other transformer-based archs, but not some convnets

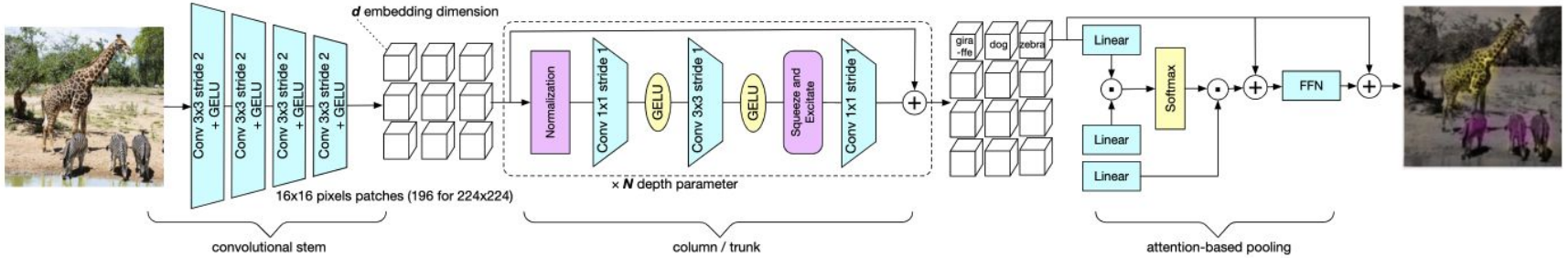


Figure 3. Detail of the full model, with the convolutional stem on the left, the convolutional main block in the middle, and here toppled with multi-class attention-based pooling on the right.

ViT comparison

- VIT - AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE *(ICLR 2021)*

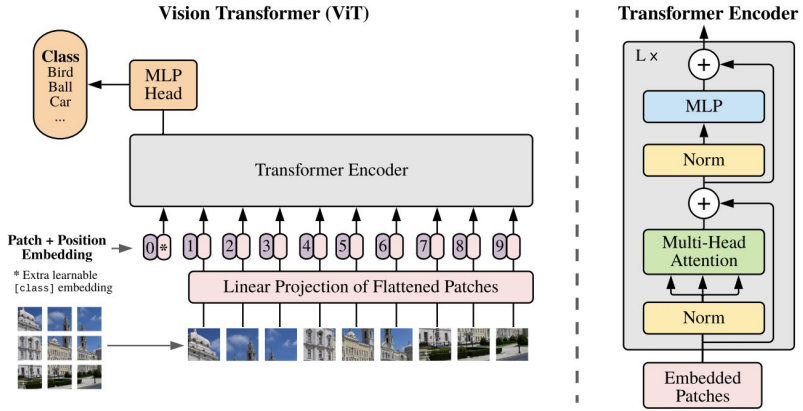
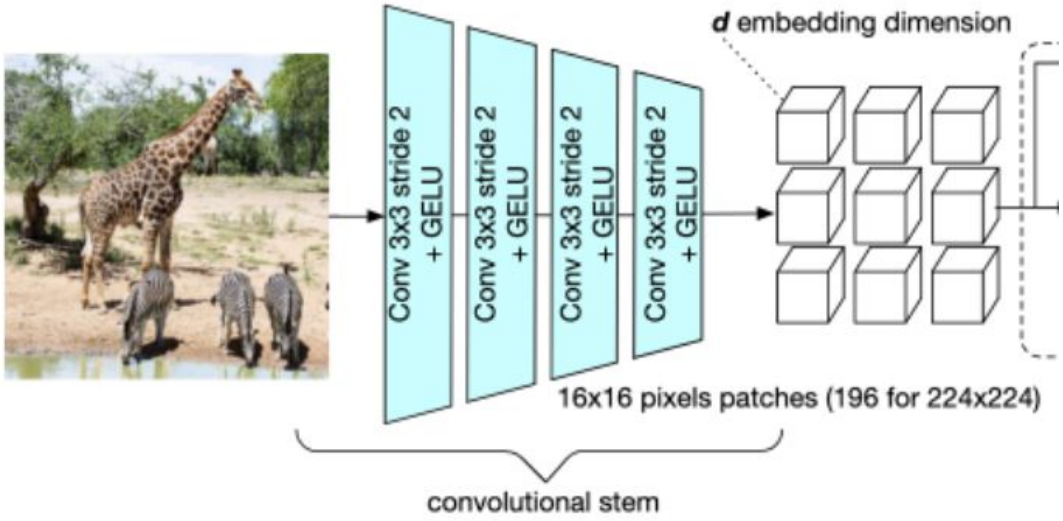


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

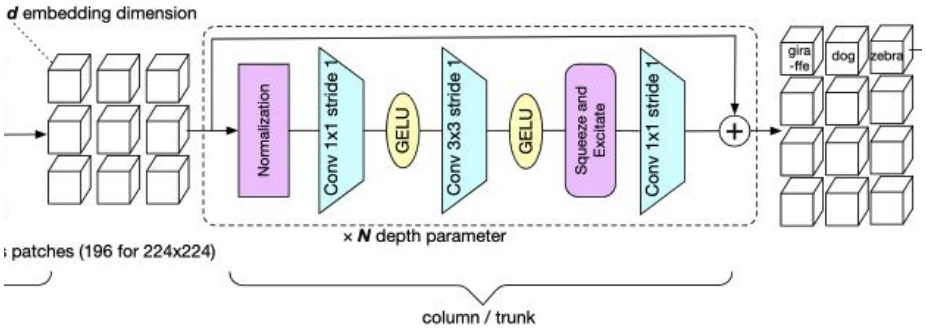
Main paper

- Replace patching and flattening pixels with simple convolutional stem that produces 16×16 patches



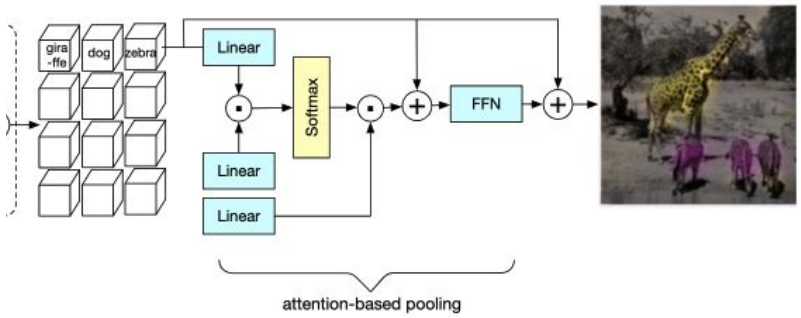
Main paper

- The trunk consists of *N* blocks that preserve embedding dimension and resolution (similar to transformer blocks without self-attention between tokens)



Main paper

- Attention-based pooling
  - A query class token attends to the projected patches and aggregates them as a weighted summation. The weights depend on the similarity of projected patches with a trainable vector (CLS) akin to a class token. The resulting d-dimensional vector is subsequently added to the CLS vector and processed by a feed-forward network (FFN).



Main paper: Results

Classification with Imagenet1k training.

Architecture	nb params ($\times 10^6$)	throughput (im/s)	FLOPs ($\times 10^9$)	Peak Mem (MB)	Top-1 Acc.	V2 Acc.
"Traditional" ConvNets						
ResNet-50 [24, 66]	25.6	2587	4.1	2182	80.4	68.7
RegNetY-4GF [46, 66]	20.6	1779	4.0	3041	81.5	70.7
RegNetY-8GF [46, 66]	39.2	1158	8.0	3939	82.2	71.1
RegNetY-16GF [46, 58]	83.6	714	16.0	5204	82.9	72.4
EfficientNet-B4 [55]	19.0	573	4.2	10006	82.9	72.3
EfficientNet-B5 [55]	30.0	268	9.9	11046	83.6	73.6
NFNet-F0 [7]	71.5	950	12.4	4338	83.6	72.6
NFNet-F1 [7]	132.6	337	35.5	6628	84.7	74.4
Vision Transformers and derivatives						
ViT: DeiT-S [58, 66]	22.0	1891	4.6	987	80.6	69.4
ViT: DeiT-B [58]	86.6	831	17.5	2078	81.8	71.5
Swin-T-224 [41]	28.3	1109	4.5	3345	81.3	69.5
Swin-S-224 [41]	49.6	718	8.7	3470	83.0	71.8
Swin-B-224 [41]	87.8	532	15.4	4695	83.5	-
Vision MLP						
Mixer-L/16 [56]	208.2	322	44.6	2614	71.8	56.2
Mixer-B/16 [56]	59.9	993	12.6	1448	76.4	63.2
ResMLP-S24 [57]	30.0	1681	6.0	844	79.4	67.9
ResMLP-B24 [57]	116.0	1120	23.0	930	81.0	69.0
Patch-based ConvNets						
ResMLP-S12 conv3x3 [57]	16.7	3217	3.2	763	77.0	65.5
ConvMixer-768/32 [2]	21.1	271	20.9	2644	80.2	-
ConvMixer-1536/20 [2]	51.6	157	51.4	5312	81.4	-
Ours-S60	25.2	1125	4.0	1321	82.1	71.0
Ours-S120	47.7	580	7.5	1450	83.2	72.5
Ours-B60	99.4	541	15.8	2790	83.5	72.6
Ours-B120	188.6	280	29.9	3314	84.1	73.9

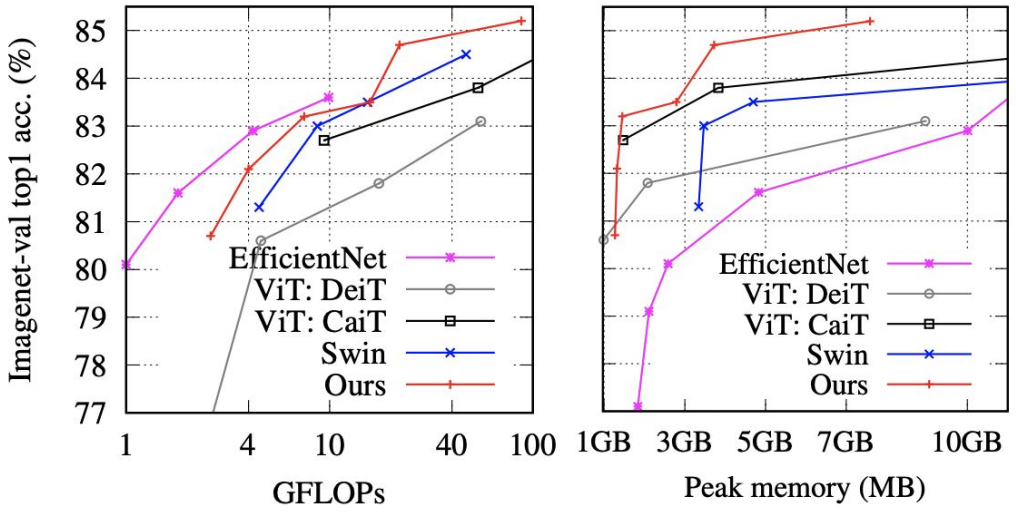


Figure 7. **Trade-offs** for ImageNet-1k top 1 accuracy vs. FLOPs requirement and peak memory requirements (for a batch of 256 images). Patch-based architectures are comparatively inferior w.r.t. the accuracy-FLOP trade-off than hierarchical ones, but offer better operating points in terms of the accuracy-memory compromise at inference time.

Main paper: Results

Table 3. **ADE20k semantic segmentation** performance using UperNet [69] (in comparable settings). All models are pre-trained on ImageNet1k except models with † symbol that are pre-trained on ImageNet21k.

Backbone	#params	FLOPs	UperNet Single scale	UperNet Multi-scale
	($\times 10^6$)	($\times 10^9$)	mIoU	mIoU
ResNet50 [24]	66.5	-	42.0	-
DeiT-S [58]	52.0	1099	-	44.0
XciT-T12/16 [19]	34.2	874	41.5	-
XciT-S12/16 [19]	54.2	966	45.9	-
Swin-T [41]	59.9	945	44.5	46.1
Ours-S60	57.1	952	**46.0**	**46.9**
XciT-M24/16 [19]	112.2	1213	47.6	-
XciT-M24/8 [19]	110.0	2161	48.4	-
Swin-B [41]	121.0	1188	48.1	49.7
Ours-B60	140.6	1258	48.1	48.6
Ours-B120	229.8	1550	**49.4**	**50.3**
Swin-B† (640 × 640)	121.0	1841	50.0	51.6
CSWin-B† [17]	109.2	1941	51.8	52.6
Ours-S60†	57.1	952	48.4	49.3
Ours-B60†	140.6	1258	50.5	51.1
Ours-B120†	229.8	1550	51.9	52.8
Ours-L120†	383.7	2086	**52.2**	**52.9**

Table 4. **COCO object detection and instance segmentation** performance on the mini-val set. All backbones are pre-trained on ImageNet1k, use Mask R-CNN model [23] and are trained with the same 3× schedule.

Backbone	#params	GFLOPs	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
ResNet50 [24]	44.2M	180	41.0	61.7	44.9	37.1	58.4	40.1
ResNet101 [24]	63.2M	260	42.8	63.2	47.1	38.5	60.1	41.3
ResNeXt101-64 [72]	101.9M	424	44.4	64.9	48.8	39.7	61.9	42.6
PVT-Small [63]	44.1M	-	43.0	65.3	46.9	39.9	62.5	42.8
PVT-Medium [63]	63.9M	-	44.2	66.0	48.2	40.5	63.1	43.5
XCiT-S12/16	44.4M	295	45.3	67.0	49.5	40.8	64.0	43.8
XCiT-S24/16 [19]	65.8M	385	46.5	68.0	50.9	41.8	65.2	45.0
ViL-Small [78]	45.0M	218	43.4	64.9	47.0	39.6	62.1	42.4
ViL-Medium [78]	60.1M	294	44.6	66.3	48.5	40.7	63.8	43.7
ViL-Base [78]	76.1M	365	45.7	67.2	49.9	41.3	64.4	44.5
Swin-T [41]	47.8M	267	46.0	68.1	50.3	41.6	65.1	44.9
Ours-S60	44.9M	264	46.4	67.8	50.8	41.3	64.8	44.2
Ours-S120	67.4M	339	47.0	69.0	51.4	41.9	65.6	44.7

Main paper: Results

Table 5. Ablation of our model: we modify each time a single architectural characteristic in our PatchConvNet model S60, and measure how it affects the classification performance on ImageNet1k. Batch-normalization improves the performance a bit. The convolutional stem is key for best performance, and the class-attention brings a slight improvement in addition to enabling attention-based visualisation properties.

↓ Modification to the architecture			Top-1 acc.
none			82.1
class-attention	→	average pooling	81.9
conv-stem	→	linear projection	80.0
layer-normalization	→	batch-normalization	82.4
single-head attention	→	multi-head attention	81.9
a single class-token	→	one class-token per class	81.1

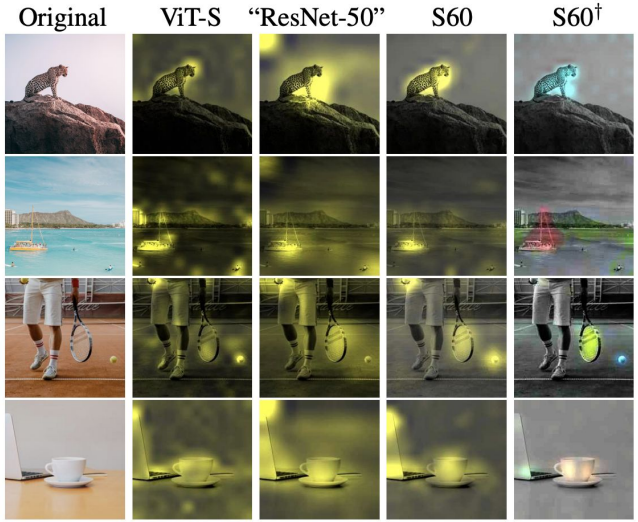


Figure 1. We augment convolutional neural networks with a learned attention-based aggregation layer. We visualize the attention maps for classification for diverse models. We first extract attention maps from a regular ViT-S [18, 58] with Dino-style [8] vizualizations. Then we consider convnets in which we replace the average pooling by our learned attention-based aggregation layer. Unlike ViT, this layer directly provides the contribution of the patches in the weighted pooling. This is shown for a "ResNet-50 [24]", and with our new simple patch-based model (PatchConvNet-S60) that we introduce to increase the attention map resolution. We can specialize this attention per class, as shown with S60†.