Keep It SimPool: Who Said Supervised Transformers Suffer from Attention Deficit?

Bill Psomas\textsuperscript{1,2} Ioannis Kakogeorgiou\textsuperscript{1} Konstantinos Karantzalos\textsuperscript{1} Yannis Avrithis\textsuperscript{2}

\textsuperscript{1}National Technical University of Athens
\textsuperscript{2}Institute of Advanced Research in Artificial Intelligence (IARAI)

Abstract

Convolutional networks and vision transformers have different forms of pairwise interactions, pooling across layers and pooling at the end of the network. Does the latter really need to be different? As a by-product of pooling, vision transformers provide spatial attention for free, but this is most often of low quality unless self-supervised, which is not well studied. Is supervision really the problem?

In this work, we develop a generic pooling framework and then we formulate a number of existing methods as instantiations. By discussing the properties of each group of methods, we derive SimPool, a simple attention-based pooling mechanism as a replacement of the default one for both convolutional and transformer encoders. We find that, whether supervised or self-supervised, this improves performance on pre-training and downstream tasks and provides attention maps delineating object boundaries in all cases. One could thus call SimPool universal. To our knowledge, we are the first to obtain attention maps in supervised transformers of at least as good quality as self-supervised, without explicit losses or modifying the architecture. Code at: https://github.com/billpsomas/simpool.

1. Introduction

Extracting visual representations and spatial pooling have been two interconnected processes since the study of 2D Gabor filters \cite{10} and early convolutional networks \cite{17}. Modern convolutional networks \cite{20, 32} gradually perform local pooling and downsampling throughout the architecture to extract a low-resolution feature tensor, followed by global spatial pooling. Vision transformers \cite{14} only downsample at input tokenization and then preserve resolution, but pooling takes place again throughout the architecture via the interaction of patch tokens with a CLS token, inherited from language models \cite{13}.

The pooling operation has been studied extensively in instance-level tasks on convolutional networks \cite{3, 42}, but less so in category-level tasks or transformers. Pooling in transformers is based on weighted averaging, using as weights the 2D attention map of the CLS token at the last layer. However, this attention map is typically of low quality, unless under self-supervision \cite{7}.

In this work, we argue that vision transformers can be reformulated in two streams, where one is extracting a visual representation on patch tokens and the other is performing spatial pooling on the CLS token; whereas, convolutional networks undergo global spatial pooling at the very last
step, before the classifier. In this sense, one can isolate the pooling process from both kinds of networks and replace it by a new one. This raises the following questions:

1. Can we derive a simple pooling process at the very last step of either convolutional or transformer encoders that improves over their default?
2. Can this process provide high-quality attention maps that delineate object boundaries, for both networks?
3. Do these properties hold under both supervised and self-supervised settings?

To answer these questions, we develop a generic pooling framework, parametrized by: (a) the number of vectors in the pooled representation; (b) whether pooling is iterative or not; (c) mappings at every stage of the process; (d) pairwise similarities, attention function and normalization; and (e) a function determining the pooling operation.

We then formulate a number of existing pooling methods as instantiations of this framework, including (a) simple pooling mechanisms in convolutional networks [20, 48, 42, 40, 47], (b) iterative methods on more than one vectors like k-means [34, 33], (c) feature re-weighting mechanisms originally designed as network components rather than pooling [23, 56], and (d) vision transformers [14, 49].

Finally, by discussing the properties of each group of methods, we derive a new, simple, attention-based pooling mechanism as a replacement of the default one for both convolutional and transformer encoders. SimPool provides high-quality attention maps that delineate object boundaries under both supervised and self-supervised settings, as shown for ViT-S [14] in Figure 1.

In summary, we make the following contributions:

1. We formulate a generic pooling framework that allows easy inspection and qualitative comparison of a wide range of methods.
2. We introduce a simple, attention-based, non-iterative, universal pooling mechanism that provides a single vector representation and answers all the above questions in the affirmative.
3. We conduct an extensive empirical study that validates the superior qualitative properties and quantitative performance of the proposed mechanism on standard benchmarks and downstream tasks.

2. Related Work

We discuss the most related work to pooling in convolutional networks and vision transformers. An extended version with more background is given in the appendix.

Convolutional networks Early convolutional networks [17, 27] are based on learnable convolutional layers interleaved with fixed spatial pooling layers that downsample. The same design remains until today [26, 46, 20, 32]. Apart from mapping to a new space, convolutional layers involve a form of local pooling and pooling layers commonly take average [27] or maximum [44, 26].

Early networks end in a fully-connected layer over a feature tensor of low resolution [27, 26, 46]. This evolved into spatial pooling, e.g. global and average resolution followed by a classifier for category-level tasks like classification [29, 20] or detection [18; 14], or global maximum followed by a pairwise loss [48] for instance-level tasks.

The spatial pooling operation at the end of the network is widely studied in instance level-tasks [3, 48, 22], giving rise to forms of spatial attention [24, 38, 6, 47, 36]. In category-level tasks, it is more common to study feature re-weighting as components of the architecture [23, 56, 22].

The two are closely related because e.g. the weighted average is element-wise weighted following by sum.

Pooling can be spatial [22, 38, 6, 47, 36], over channels [23], or both [24, 56]. CBAM [56] is particularly related to our work in the sense that it includes global average pooling followed by a form of spatial attention, although the latter is not evident in its original formulation and although CBAM is not a pooling mechanism.

Vision transformers Pairwise interactions between features are forms of pooling or self-attention over the spatial [55, 4, 63, 41] or channel dimensions [8, 54]. Originating in language models [51], vision transformers [14] streamlined these approaches and dominated the architecture landscape. Several variants often bring back ideas from convolutional networks [31, 58, 19, 57, 15, 21, 61].

Transformers downsample only at the input, forming spatial patch tokens. Pooling is based on a learnable CLS token, which, beginning at the input space, undergoes the same self-attention operation with patch tokens and provides a global image representation. That is, the network ends in global weighted average pooling, using as weights the attention of CLS over the patch tokens.

Few works that have studied beyond CLS for pooling are mostly limited to global average pooling (GAP) [31, 62, 50, 43]. CLS offers attention maps for free, however of low quality unless in a self-supervised setting [7], which is not well studied. Few works that attempt to rectify this in the supervised setting include a spatial entropy loss [39], shape distillation from convolutional networks [35] and skipping computation of self-attention [52].

We attempt to address these limitations and study pooling in convolutional networks, vision transformers, supervised and self-supervised alike. We derive a simple, attention-based, universal pooling mechanism, improving both performance and attention maps.
3. Method

We develop a generic pooling framework that encompasses many simple or more complex pooling methods, iterative or not, attention-based or not. We then examine a number of methods as instantiations of this framework. Finally, we discuss their properties and make particular choices in designing our solution.

3.1. A generic pooling framework

Preliminaries Let \( X \in \mathbb{R}^{d \times W \times H} \) be the 3-dimensional feature tensor obtained from the last layer of a network for a given input image, where \( d \) is the number of feature channels and \( W, H \) are the width and height. We represent the image by the feature matrix \( X \in \mathbb{R}^{d \times p} \) by flattening the spatial dimensions of \( X \), where \( p := W \times H \) is the number of spatial locations. Let \( x_i \in \mathbb{R}^p \) denote the \( i \)-th row of \( X \), that is, corresponding to the \( 2 \)-dimensional feature map in channel \( i \), and \( x_j \in \mathbb{R}^d \) denote the \( j \)-th column of \( X \), that is, the feature vector of spatial location \( j \).

By \( 1_n \in \mathbb{R}^n \), we denote the all-ones vector. Given an \( m \times n \) matrix \( A \geq 0 \), by \( \eta_1(A) := \text{diag}(A_{1n})^{-1}A \) we denote row-wise \( \ell_1 \)-normalization; similarly, \( \eta_2(A) := A \text{diag}(1_{n1})^{-1} \) for column-wise.

Pooling process The objective of pooling is to represent the image by one or more vectors, obtained by interaction with \( X \), either in a single step or by an iterative process. We denote the pooling process by function \( \pi : \mathbb{R}^{d \times p} \to \mathbb{R}^{d' \times k} \) and the output vectors by matrix \( U = \pi(X) \in \mathbb{R}^{d' \times k} \), where \( d' \) is the number of dimensions, possibly \( d' = d \), and \( k \) is the number of vectors. In the most common case of a single vector, \( k = 1 \), we denote \( U \) by \( u \in \mathbb{R}^{d} \). We discuss here the general iterative process; single-step pooling is the special case where the number of iterations is 1.

Initialization We define \( X^0 := X \) and make a particular choice for \( U^0 \in \mathbb{R}^{d \times k} \), where \( d^0 := d \). The latter may depend on the input \( X \), in which case it is itself a simple form of pooling or not; for example, it may be random or a learnable parameter over the entire training set.

Pairwise interaction Given \( U^t \) and \( X^t \) at iteration \( t \), we define the query and key matrices

\[
Q = \phi_Q(U^t) \in \mathbb{R}^{n^t \times k} \quad (1) \]
\[
K = \phi_K(X^t) \in \mathbb{R}^{n^t \times p}. \quad (2)
\]

Here, functions \( \phi_Q : \mathbb{R}^{d \times k} \to \mathbb{R}^{n^t \times k} \) and \( \phi_K : \mathbb{R}^{d' \times p} \to \mathbb{R}^{n^t \times p} \) may be the identity, linear or non-linear mappings to a space of the same \( (n^t = d^t) \) or different dimensions.

We let \( K, Q \) interact pairwise by defining the \( p \times k \) matrix \( S(K, Q) := ((s(k_{ij}, q_{ij}))_{i=1}^{p}_{j=1})^{k^t} \), where \( s : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R} \) for any \( n \) is a similarity function. For example, \( s \) can be dot product, cosine similarity, or a decreasing function of some distance. In the case of dot product, \( s(x, y) := x^\top y \) for \( x, y \in \mathbb{R}^d \), it follows that \( S(K, Q) = K^\top Q \in \mathbb{R}^{p \times k} \).

Attention We then define the attention matrix

\[
A = h(S(K, Q)) \in \mathbb{R}^{p \times k}. \quad (3)
\]

Here, \( h : \mathbb{R}^{p \times k} \to [0, 1]^{p \times k} \) is a nonlinear function that may be elementwise, for instance relu or exp, normalization over rows or columns of \( S(K, Q) \), or it may yield a form of correspondence or assignment between the columns of \( K \) and \( Q \), possibly optimizing a cost function.

Attention-weighted pooling We define the value matrix

\[
V = \phi_V(X^t) \in \mathbb{R}^{n^t \times p}. \quad (4)
\]

Here, function \( \phi_V : \mathbb{R}^{d \times p} \to \mathbb{R}^{n^t \times p} \) plays a similar role with \( \phi_Q, \phi_K \). Attention-weighted pooling is defined by

\[
Z = f^{-1}(f(V)A) \in \mathbb{R}^{n^t \times k}. \quad (5)
\]

Here, \( f : \mathbb{R} \to \mathbb{R} \) is a nonlinear elementwise function that determines the pooling operation, for instance, average or max-pooling. The product \( f(V)A \) defines \( k \) linear combinations over the columns of \( f(V) \), that is, the features at different spatial locations. If the columns of \( A \) are \( \ell_1 \)-normalized, then those are convex combinations. Thus, matrix \( A \) defines the weights of an averaging operation.

Output Finally, we define the output matrices corresponding to image features and pooling,

\[
X^{t+1} = \phi_X(U^t) \in \mathbb{R}^{d^t+1 \times p} \quad (6)
\]
\[
U^{t+1} = \phi_U(Z) \in \mathbb{R}^{d^t+1 \times k}. \quad (7)
\]

Functions \( \phi_X : \mathbb{R}^{n^t \times p} \to \mathbb{R}^{d^t+1 \times p} \) and \( \phi_U : \mathbb{R}^{n^t \times k} \to \mathbb{R}^{d^t+1 \times k} \) play a similar role with \( \phi_Q, \phi_K, \phi_V \) but also determine the dimensionality \( d^t+1 \) for the next iteration.

At this point, we may iterate by returning to the “pairwise interaction” step, or terminate, yielding \( U^{t+1} \) as \( U \) with \( d^t = d^t+1 \). Non-iterative methods do not use \( \phi_X^t \).

3.2. A pooling landscape

Table 1 examines a number of pooling methods as instantiations of our framework. The objective is to get insight into their basic properties. How this table was obtained is detailed in the appendix.

**Group 1** consists of simple methods with \( k = 1 \) that are not attention-based and have been studied in category-level tasks [20, 40] or mostly in instance-level tasks [48, 42, 47]. Here, the attention is a vector \( a \in \mathbb{R}^p \) and either is uniform or depends directly on \( X \), by pooling over channels [47]. Most important is the choice of pooling operation by function \( f \). Log-sum-exp [40] arises with \( f(x) = e^{rx} \) with
Table 1. A landscape of pooling methods. Cat: used in category-level tasks; Iter: iterative; *: simplified. \( \pi_A \): GAP; \( \sigma \): sigmoid; \( \sigma_2 \): softmax over columns; \( \eta_2 \): column normalization; \( g_m \): partitioning in \( m \) groups (see appendix). Cyan: ours; gray: learnable choices with ours; green: learnable; red: hyperparameter; blue: detailed in the appendix.

As studied by Amari [1], function \( f_\alpha \) is defined for \( x \geq 0 \) (\( \alpha \neq 1 \)) or \( x > 0 \) (\( \alpha = 1 \)). It reduces to the maximum, quadratic mean (RMS), arithmetic mean, geometric mean, harmonic mean, and minimum for \( \alpha = -\infty, -3, -1, 1, 3, +\infty \), respectively. It has been proposed as a transition from average to max-pooling [5] and is known as GeM [42], with \( \gamma = (1 - \alpha)/2 > 1 \) being a learnable parameter.

Group 2 incorporates iterative methods with \( k > 1 \), including standard \( k \)-means, the soft-clustering variant Slot Attention [33] and optimal transport between \( U \) and \( X \) [34]. The latter is not formally iterative according to our framework, but the Sinkhorn algorithm is iterative internally.

Group 3 refers to methods introduced as modules within the architecture rather than pooling mechanisms [23, 56]. An interesting aspect is initialization of \( U^0 \) by global average pooling (GAP) on \( X \):

\[
\pi_A(X) := X 1_p/p = \frac{1}{p} \sum_{j=1}^{p} x_{j} \in \mathbb{R}^d,
\]

where \( 1_p \in \mathbb{R}^p \) is the all-ones vector. Channel attention (\( \phi_Q(U) \)) and spatial attention (\( A \)) in CBAM [56] are based on a few layers followed by sigmoid, playing the role of a binary classifier (e.g. foreground/background); whereas, transformer-based attention uses directly the query and softmax normalization, respectively. Although not evident in the original formulation, we show in the appendix that there is pairwise interaction.

Group 4 refers to vision transformers [14, 49], which we reformulate in two separate streams, one for the CLS token, \( U \), and another for the patch tokens, \( X \). We observe that, what happens to the CLS token throughout the entire encoder, is an iterative pooling process. Moreover, although \( U \) is just one vector, multi-head attention splits it into \( m \) subvectors, where \( m \) is the number of heads. Thus, \( m \) is similar to \( k \) in \( k \)-means. The difference of CaiT [49] from ViT [14] is that this iteration happens only in the last couple of layers, with the patch embeddings \( X \) being fixed.

3.3. SimPool

Group 5 of Table 1 is our method, SimPool. A schematic overview is given in Figure 2.

Ploolling process We are striving for a simple design. While pooling into \( k > 1 \) vectors would yield a more discriminative representation, either these would have to be concatenated, as is the case of multi-head attention, or a particular similarity kernel would be needed beyond dot product, which we consider to be beyond the scope of this
work. We rather argue that it is the task of the encoder to learn a single vector representation of objects, even if those are composed of different parts. This argument is stronger when pre-training is performed on images mostly depicting one object, like ImageNet-1k.

We observe in Table 1 that only methods explicitly pooling into \( k > 1 \) vectors or implicitly using \( m > 1 \) heads are iterative. We explain why in the next paragraph. Following this insight, we perform pooling in a single step.

In summary, our solution is limited to a single vector \( \mathbf{u} \in \mathbb{R}^d \) for pooling, that is, \( k = 1 \), and is non-iterative.

**Initialization** We observe in Table 1 that single-step attention-based methods in Group 3 initialize \( \mathbf{u}^0 \) by GAP. We hypothesize that, since attention is based on pairwise similarities, it is essential that \( \mathbf{u}^0 \) is chosen such that its similarities with \( X \) are maximized on average, which would help to better discriminate between foreground (high similarity) and background (low similarity). Indeed, for \( s(x, y) = -\|x - y\|^2 \), the sum of squared Euclidean distances of each column \( x_i \) of \( X \) to \( u \in \mathbb{R}^d \)

\[
J(u) = \frac{1}{2} \sum_{i=1}^{p} \|x_i - u\|^2 \tag{10}
\]

is a convex distortion measure with unique minimum the average of vectors \( \{x_i\} \)

\[
u^* := \arg \min_{u \in \mathbb{R}^d} J(u) = \frac{1}{p} \sum_{i=1}^{p} x_i = \pi_A(X) \tag{11}
\]

which can be found in closed form. By contrast, for \( k > 1 \) vectors, distortion can only be minimized iteratively, e.g. by \( k \)-means. We therefore choose:

\[
\mathbf{u}^0 = \pi_A(X) = X1_p/p \tag{12}
\]

**Pairwise interaction, attention** We follow the attention mechanism of transformers, in its simplest possible form. In particular, we use a single head, \( m = 1 \), like Slot Attention [33] (which however uses \( k \) vectors). We find that the query and key mappings are essential in learning where to attend as a separate task from learning the representation for the given task at hand. In particular, we use linear mappings \( \phi_Q, \phi_K \) with learnable parameters \( W_Q, W_K \in \mathbb{R}^{d \times d} \) respectively:

\[
q = \phi_Q(u^0) = W_Q u^0 \in \mathbb{R}^d
\]

\[
K = \phi_K(X) = W_K X \in \mathbb{R}^{d \times p} \tag{14}
\]

As in transformers, we define pairwise similarities as dot product, that is, \( S(K, q) = K^\top q \in \mathbb{R}^{p \times k} \), and attention as scaled softmax over columns (spatial locations), that is, \( h(S) := \sigma_2(S/\sqrt{d}) \):

\[
a = \sigma_2 \left(K^\top q/\sqrt{d} \right) \in \mathbb{R}^p, \tag{15}
\]

where \( \sigma_2(S) := \eta_2(\exp(S)) \) and \( \exp \) is taken elementwise.

**Attention-weighted pooling** As shown in Table 1, the average pooling operation \( f = f_{-1} \) is by far the most common. However, the more general function \( f_\alpha \) (8) has shown improved performance in instance-level tasks [42]. For \( \alpha < -1 \) (\( \gamma > 1 \)) in particular, it yields an intermediate operation between average and max-pooling. The latter is clearly beneficial when feature maps are sparse, because it better preserves the non-zero elements.

We adopt \( f = f_\alpha \) for its genericity: the only operation that is not included as a special case in Table 1 is \( \logsumexp \) [40]. This choice assumes \( X \geq 0 \). This is common in networks ending in relu, like ResNet [20], which is also what makes feature maps sparse. However, vision transformers and modern convolutional networks like ConvNeXt [32] do not end in relu; hence \( X \) has negative elements and is not necessarily sparse. We therefore define

\[
V = \phi_V(X) = X - \min X \in \mathbb{R}^{d \times p}, \tag{16}
\]

where the minimum is taken over all elements of \( X \), such that \( f_\alpha \) operates only on non-negative numbers.

We also define \( u = \phi_U(z) = z \) and the output dimension is \( d' = d \). Thus, the mappings \( \phi_V, \phi_U \) are parameter-free. The argument is that, for average pooling for example (\( f = f_{-1} \) in (5)), any linear layers before or after pooling would commute with pooling, thus they would form part of the encoder rather than the pooling process. Moreover, Table 1 shows that \( \phi_U \) is non-identity only for iterative methods.

In summary, we define SimPool (SP) as

\[
u = \pi_{SP}(X) := f_\alpha^{-1}(f_\alpha(V)a) \in \mathbb{R}^d, \tag{17}\]

where \( V \in \mathbb{R}^{d \times p} \) is the value (16) and \( a \in \mathbb{R}^p \) is the attention map (15). Parameter \( \alpha \) is learned in GeM [42], but we find that treating it as a hyperparameter better controls the quality of the attention maps.

### 4. Experiments

#### 4.1. Datasets, networks and evaluation protocols

**Supervised pre-training** We train ResNet-18, ResNet-50 [20], ConvNeXt-S [32], ViT-S and ViT-B [14] for *image classification* on ImageNet-1k. For the analysis subsection 4.2 and ablation subsection 4.4, we train ResNet-18 on the first 20% of training examples per class of ImageNet-1k [11] (called ImageNet-20%) for 100 epochs. For the benchmark of subsection 4.3, we train ResNet-50 for 100 and 200 epochs, ConvNeXt-S and ViT-S for 100 and 300 epochs and ViT-B for 100 epochs, all on the 100% of ImageNet-1k. We evaluate on the full validation set in all cases and measure top-1 classification accuracy. The baseline is the default per network, i.e. GAP for convolutional networks and CLS token for transformers.
Self-supervised pre-training  On the 100% of ImageNet-1k, we train DINO [7] with ResNet-50, ConvNeXt-S and ViT-S for 100 epochs. We evaluate on the validation set by k-NN and linear probing on the training set. For linear probing, we train a linear classifier on top of features as in DINO [7]. For k-NN [59], we freeze the model and extract features, then use a k-nearest neighbor classifier with k = 10.

Downstream tasks  We fine-tune supervised and self-supervised ViT-S on CIFAR-10 [25], CIFAR-100 [25] and Oxford Flowers [37] for image classification, measuring top-1 classification accuracy. We perform object localization without fine-tuning using supervised and self-supervised ViT-S on CUB [53] and ImageNet-1k, measuring MaxBoxAccV2 [9]. We perform object discovery without fine-tuning using self-supervised ViT-S with DINO-SEG [7] and LOST [45] on VOC07 [16], VOC12 [16] and COCO [30], measuring CorLoc [12]. We validate robustness against background changes using ViT-S on ImageNet-9 [60] and its variations. We use the linear head and linear probe for supervised and self-supervised ViT-S, respectively, measuring top-1 classification accuracy.

In the appendix, we provide implementation details, more benchmarks, ablations and visualizations.

4.2. Experimental Analysis

Figure 3 evaluates different methods in groups following Table 1, regardless of their original design for (a) pooling or not, (b) different tasks, e.g. instance-level or category-level, (c) different networks, e.g. convolutional or transformers.

Group 1 consists of simple pooling methods with: (a) no parameters: GAP [29], max [48], GAP+max [28]; and (b) scalar parameter: GeM [42] and LSE [40]. HOW [47] is the only method to use (parameter-free) attention. GeM is performing the best, with LSE following second. These methods are inferior to those in other groups.

Group 2 incorporates methods with k > 1 vectors. We set k = 3 and take the maximum of the 3 logits per class. OTK and Slot use attention. Slot attention [33] works best, outperforming k-means by 1.3%.

Group 3 refers to parametric attention-based methods, weighting features based on their importance for the task: CBAM [56], Squeeze-Excitation [23] and Gather-Excite [22]. While originally designed as components within the architecture, we adapt them to pooling by GAP at the end. Gather-Excite [22] performs best.

Group 4 refers to parametric attention-based methods found in vision transformers. ViT [14] refers to multi-head self-attention learnable CLS and four heads, which we incorporate as a single layer at the end of the model. CaiT [49] is the same but using only cross-attention between CLS and patch embeddings. CaiT performs the best.

SimPool outperforms all other methods. Seeing this experiment as a tournament, we select the best performing method of each group and qualify it for the benchmark of subsection 4.3.

4.3. Benchmark

Image Classification  Table 2 compares SimPool with baseline and tournament winners per group of subsection 4.2 on supervised pre-training for classification. For 100 epochs, SimPool outperforms all methods, consistently improving the baseline by 0.6% using convolutional networks, 1.6% using ViT-S and 1.0% using ViT-B. Gather-Excite [22] improves over the baseline only on convolutional networks, while Slot [33] only on ViT-S. CaiT improves over the baseline only for ConvNeXt-S. By contrast, SimPool improves everywhere. For more than 100 epochs, SimPool improves the baseline by 0.5% using ResNet-50, 0.4% using ConvNeXt-S and 0.8% using ViT-S.

Table 3 evaluates self-supervised pre-training for 100 epochs. SimPool improves over the baseline by 2.0% k-NN and 1.4% linear probing on ResNet-50; 3.7% k-NN and 4.0% linear probing on ConvNeXt-S; and 0.9% k-NN and 1.3% linear probing on ViT-S.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>Ep</th>
<th>RESNET-50</th>
<th>CONVNeXt-S</th>
<th>ViT-S</th>
<th>k-NN</th>
<th>PROB</th>
<th>k-NN</th>
<th>PROB</th>
<th>k-NN</th>
<th>PROB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>100</td>
<td>61.8</td>
<td>63.0</td>
<td>65.1</td>
<td>68.2</td>
<td>68.9</td>
<td>71.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SimPool</td>
<td>100</td>
<td>63.8</td>
<td>64.4</td>
<td>68.8</td>
<td>72.2</td>
<td>69.8</td>
<td>72.8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Image classification accuracy (%) with fine-tuning for classification for 1000 epochs. ViT-S pre-trained on ImageNet-1k for 100 epochs. Self-supervision with DINO [7], CF-10: CIFAR-10 [25], CF-100: CIFAR-100 [25], FL: Flowers [37].

<table>
<thead>
<tr>
<th>METHOD</th>
<th>SUPERVISED</th>
<th>SELF-SUPERVISED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CF-10</td>
<td>CF-100</td>
</tr>
<tr>
<td>Baseline</td>
<td>98.1</td>
<td>86.0</td>
</tr>
<tr>
<td>SimPool</td>
<td>98.4</td>
<td>86.2</td>
</tr>
</tbody>
</table>

Table 5. Localization accuracy MaxBoxAccV2 on CUB test and ImageNet-1k validation set. ViT-S pre-trained on ImageNet-1k for 100 epochs. Self-supervision with DINO [7]. @20: at epoch 20.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>SUPERVISED</th>
<th>SELF-SUPERVISED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CUB</td>
<td>ImageNet</td>
</tr>
<tr>
<td>Baseline</td>
<td>63.1</td>
<td>53.6</td>
</tr>
<tr>
<td>SimPool</td>
<td>77.9</td>
<td>64.4</td>
</tr>
<tr>
<td>Baseline@20</td>
<td>62.4</td>
<td>50.5</td>
</tr>
<tr>
<td>SimPool@20</td>
<td>74.0</td>
<td>62.6</td>
</tr>
</tbody>
</table>

Fine-tuning for classification Table 4 evaluates fine-tuning for classification on different datasets of a supervised and a self-supervised ViT-S. SimPool brings a small improvement over the baseline in all cases.

Object localization Accurate localization can have a significant impact on classification accuracy, particularly under multiple objects, complex scenes and background clutter. Table 5 evaluates localization accuracy under both supervision settings. SimPool significantly improves the baseline by up to 7% MaxBoxAccV2 when self-supervised and up to 14% when supervised. In the latter case, the gain is already up to 12% at epoch 20.

Unsupervised object discovery Table 6 studies LOST [45], which uses the raw features of a vision transformer pre-trained using DINO [7] for unsupervised single-object discovery, as well as the baseline DINO-seg [45, 7], which uses the attention maps instead. SimPool significantly outperforms the baseline on all datasets by up to 25.2% CorLoc for DINO-seg and 5.6% for LOST on VOC12. Again, the gain is significant already at the first 20 epochs.

Background changes We evaluate robustness to the background changes using IN-9 [60] dataset. Table 7 shows that SimPool improves over the baseline under both supervision settings with only 2 out of 8 exceptions under DINO [7] pre-training. The latter is justified, given that none of the foreground objects or masks are present in these settings.

Computation resources Table 8 shows the number of parameters and floating point operations per second for the best competitors of Figure 3. Resources depend on the embedding dimension $d$. SimPool is higher than the baseline but not the highest.
(BASE) when using SimPool and add blocks when using CLS. We find that, to exceed the accuracy of BASE SimPool, BASE CLS needs 5 extra blocks, i.e., 9M more parameters. Equally interestingly, removing 3 blocks from BASE SimPool is only slightly worse than BASE CLS, having 5M fewer parameters.

### 4.4. Ablation study

We ablate the design and components of SimPool. More ablations are found in the appendix. In particular, for function \( f_α \) (8), we set \( γ = 2 \) for convolutional networks and \( γ = 1.25 \) for transformers by default, where \( γ = (1 − α)/2 \) is a hyperparameter.

**Design** In Table 10 (left), we ablate (a) the attention function \( h(3) \); (b) the number of iterations with shared parameters at every iteration (LAYERS) or not (ITER); (c) the initialization \( U^0 \); (d) the pairwise similarity function \( s; \) (e) the number \( k \) of pooled vectors, obtained by \( k \)-means instead of GAP. We also consider queries and keys sharing the same mapping, \( W_Q = W_K \). We observe that multi-head, few iterations and initialization by \( \text{diag}(X^T X) \) perform slightly worse, without adding any extra parameters, while setting \( W_Q = W_K \) performs slightly worse, having 50% less parameters.

**Linear and LayerNorm layers** In Table 10 (right), we systematically ablate linear and LayerNorm (LN) [2] layers on query \( q \), key \( k \) and value \( v \). We strive for performance and quality while at the same time having a small number of components and parameters. In this sense, we choose the setup that includes linear layers on \( q, k \) and LN on \( k, v \), yielding 56.6 accuracy. We observe that having linear and LN layers everywhere performs best under classification accuracy. However, this setup has attention maps of lower quality and more parameters.

### 5. Conclusion

We have introduced SimPool, a simple, attention-based pooling mechanism that acts at the very last step of either convolutional or transformer encoders, delivering highly superior quantitative results on several benchmarks and downstream tasks. In addition, SimPool delivers decent attention maps in both convolutional and transformer networks under both supervision and self-supervision with remarkable improvement in delineating object boundaries for supervised transformers. Despite this progress, we believe that investigating why the standard CLS-based attention fails under supervision deserves further study.

**Acknowledgements** This work was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the BiCUBES project (grant: 03943). It was also partially supported by the RAMONES and iToBos EU Horizon 2020 projects, under grants 101017808 and 965221, respectively. NTUA thanks NVIDIA for the support with the donation of GPU hardware.
References


IEEE conference on computer vision and pattern recognition, pages 7794–7803, 2018. 2


