AdaPool: Exponential Adaptive Pooling for Information-Retaining Downsampling

Alexandros Stergiou, *Student Member*, IEEE, Ronald Poppe, *Senior Member*, IEEE

***Abstract*—Pooling layers are essential building blocks of Convolutional Neural Networks (CNNs) that reduce computational overhead and increase the receptive fields of proceeding convolutional operations. They aim to produce downsampled volumes that closely resemble the input volume while, ideally, also being computationally and memory efficient. It is a challenge to meet both requirements jointly. To this end, we propose an adaptive and exponentially weighted pooling method named *adaPool*. Our proposed method uses a parameterized fusion of two sets of pooling kernels that are based on the exponent of the Dice-Sørensen coefficient and the exponential maximum, respectively. A key property of adaPool is its bidirectional nature. In contrast to common pooling methods, weights can be used to upsample a downsampled activation map. We term this method *adaUnPool*. We demonstrate how adaPool improves the preservation of detail through a range of tasks including image and video classification and object detection. We then evaluate adaUnPool on image and video frame super-resolution and frame interpolation tasks. For benchmarking, we introduce *Inter4K*, a novel high-quality, high frame-rate video dataset. Our combined experiments demonstrate that adaPool systematically achieves better results across tasks and backbone architectures, while introducing a minor additional computational and memory overhead.**

***Index Terms*—pooling, downsampling, upsampling**

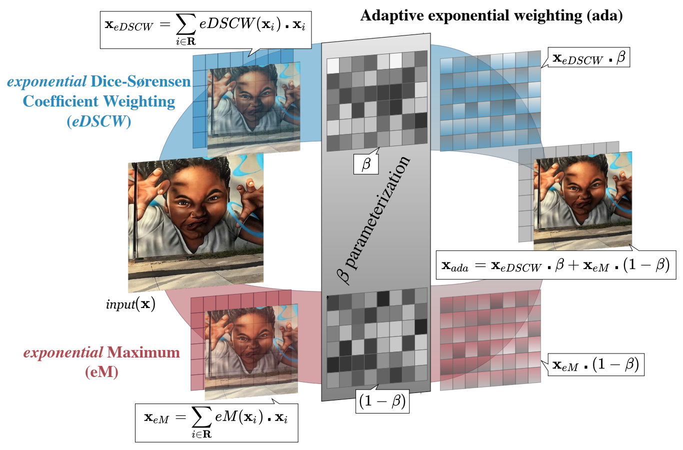

$\mathbf{x}_{eDSCW} = \sum_{i\in\mathbf{R}} eDSCW(\mathbf{x}_i) \cdot \mathbf{x}_i$
Adaptive exponential weighting (ada)
exponential Dice-Sørensen Coefficient Weighting (*eDSCW*)
$\mathbf{x}_{eDSCW} \cdot \beta$
β
β parameterization
input($\mathbf{x}$)
$\mathbf{x}_{ada} = \mathbf{x}_{eDSCW} \cdot \beta + \mathbf{x}_{eM} \cdot (1-\beta)$
exponential Maximum (eM)
$\mathbf{x}_{eM} \cdot (1-\beta)$
$(1-\beta)$
$\mathbf{x}_{eM} = \sum_{i\in\mathbf{R}} eM(\mathbf{x}_i) \cdot \mathbf{x}_i$

Fig. 1. **AdaPool downsampling**. The output is the combination of two processes. The first uses exponential Dice-Sørensen Coefficient Weighting (*e*DSCW) downsampling, based on a region's mean ($\overline{\mathbf{x}}$). The second downsamples using the exponential maximum *e*M. Both outputs ($\mathbf{x}_{eM}$,$\mathbf{x}_{eDSCW}$) are summed with region-based weight masks $\boldsymbol{\beta}$ and (1-$\boldsymbol{\beta}$) to produce the exponentially sub-sampled and adaptively weighted output ($\mathbf{x}_{ada}$).

I. Introduction

Pooling methods downsample spatial input to a lower resolution. Their goal is to minimize the loss of information by capturing the most important information and by retaining structural aspects such as contrast and texture. Pooling operations are essential in image and video processing approaches, including those based on convolutional neural networks (CNNs). In CNNs, pooling operations aid in reducing the computational burden, while increasing the receptive field of subsequent convolutions. Pooling is a key component in virtually all popular CNN architectures and they should have low computation and memory overheads.

A range of pooling methods has been proposed, each with different properties (see Section II). Most architectures use maximum or average pooling, both of which are fast and memory-efficient but leave room for improvement in terms of retaining information. Another approach is the use of trainable sub-networks. Such methods have shown some improvements over average or maximum pooling, but they are typically less efficient and generally applicable because their parameters need to be determined beforehand.

In this work, we study how the shortcomings of pooling methods can be addressed with low-computational approaches based on exponential weighting. We introduce approaches to weight kernel regions, either based on the softmax-weighted sum of activations [9], or based on the exponent of the similarity between each kernel activation and the mean kernel activation obtained by the Dice-Sørensen Coefficient [7], [8]. We then propose *adaPool* as the parameterized fusion of both methods, schematically visualized in Figure 1.

Many tasks, including instance segmentation, image generation and super-resolution, require upsampling of the input, which has the inverse goal of pooling. With the exception of LiftPool [10], pooling operations cannot be reversed as this would lead to sparse upsampling results (e.g., using maximum pooling [11]). Common upsampling approaches such as interpolation, transposed convolutions and de-convolution approximate, rather than reconstruct, the higher-dimensional features. The lack of inclusion of prior knowledge is an obstacle as the encoding of information to a lower dimensionality comes at a loss of local information in the higher dimension. Instead, we argue that the inclusion of prior local knowledge benefits the upsampling process. Based on the same formulation as adaPool, we introduce an upsampling process (*adaUnPool*).

We demonstrate the favorable effects of adaPool in preserv-

This work was done while A. Stergiou was with Utrecht University.
A. Stergiou is with the Department of Computer Science, University of Bristol, Bristol, BS8 1UB, United Kingdom, e-mail: alexandros.stergiou@bristol.ac.uk
R. Poppe is with the Department of Information and Computing Sciences, Utrecht University, Utrecht, 3584 CC, The Netherlands, e-mail: r.w.poppe@uu.nl

Code URL: https://git.io/JcDHN
Dataset URL: https://alexandrosstergiou.github.io/datasets/Inter4K/

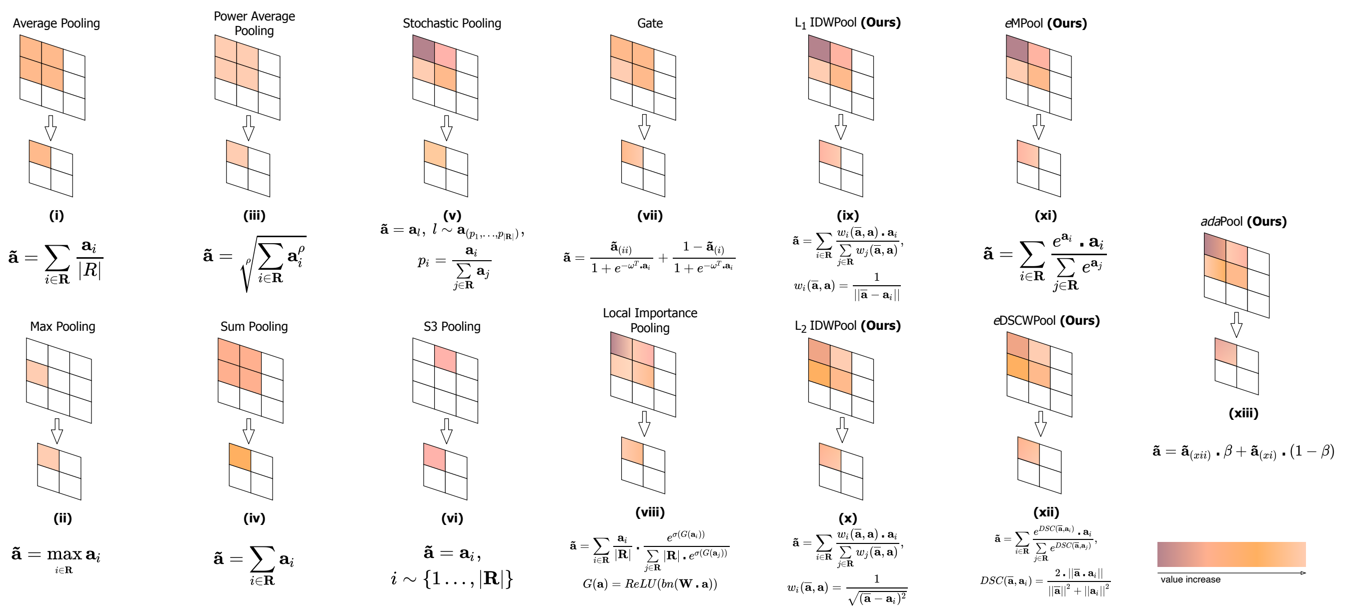

Average Pooling
Power Average Pooling
Stochastic Pooling
Gate
L1 IDWPool (Ours)
eMPool (Ours)
(i)
$\tilde{\mathbf{a}} = \sum_{i \in \mathbf{R}} \frac{\mathbf{a}_i}{|R|}$
(iii)
$\tilde{\mathbf{a}} = \sqrt[\rho]{\sum_{i \in \mathbf{R}} \mathbf{a}_i^{\rho}}$
(v)
$\tilde{\mathbf{a}} = \mathbf{a}_l,\ l \sim \mathbf{a}_{(p_1,\ldots,p_{|\mathbf{R}|})},$
$p_i = \frac{\mathbf{a}_i}{\sum_{j \in \mathbf{R}} \mathbf{a}_j}$
(vii)
$\tilde{\mathbf{a}} = \frac{\tilde{\mathbf{a}}_{(ii)}}{1+e^{-\omega^T \cdot \mathbf{a}_i}} + \frac{1-\tilde{\mathbf{a}}_{(i)}}{1+e^{-\omega^T \cdot \mathbf{a}_i}}$
(ix)
$\tilde{\mathbf{a}} = \sum_{i \in \mathbf{R}} \frac{w_i(\overline{\mathbf{a}}, \mathbf{a}) \cdot \mathbf{a}_i}{\sum_{j \in \mathbf{R}} w_j(\overline{\mathbf{a}}, \mathbf{a})},$
$w_i(\overline{\mathbf{a}}, \mathbf{a}) = \frac{1}{||\overline{\mathbf{a}} - \mathbf{a}_i||}$
(xi)
$\tilde{\mathbf{a}} = \sum_{i \in \mathbf{R}} \frac{e^{\mathbf{a}_i} \cdot \mathbf{a}_i}{\sum_{j \in \mathbf{R}} e^{\mathbf{a}_j}}$
adaPool (Ours)
Max Pooling
Sum Pooling
S3 Pooling
Local Importance Pooling
L2 IDWPool (Ours)
eDSCWPool (Ours)
(ii)
$\tilde{\mathbf{a}} = \max_{i \in \mathbf{R}} \mathbf{a}_i$
(iv)
$\tilde{\mathbf{a}} = \sum_{i \in \mathbf{R}} \mathbf{a}_i$
(vi)
$\tilde{\mathbf{a}} = \mathbf{a}_i,$
$i \sim \{1 \ldots, |\mathbf{R}|\}$
(viii)
$\tilde{\mathbf{a}} = \sum_{i \in \mathbf{R}} \frac{\mathbf{a}_i}{|\mathbf{R}|} \cdot \frac{e^{\sigma(G(\mathbf{a}_i))}}{\sum_{j \in \mathbf{R}} |\mathbf{R}| \cdot e^{\sigma(G(\mathbf{a}_j))}}$
$G(\mathbf{a}) = ReLU(bn(\mathbf{W} \cdot \mathbf{a}))$
(x)
$\tilde{\mathbf{a}} = \sum_{i \in \mathbf{R}} \frac{w_i(\overline{\mathbf{a}}, \mathbf{a}) \cdot \mathbf{a}_i}{\sum_{j \in \mathbf{R}} w_j(\overline{\mathbf{a}}, \mathbf{a})},$
$w_i(\overline{\mathbf{a}}, \mathbf{a}) = \frac{1}{\sqrt{(\overline{\mathbf{a}} - \mathbf{a}_i)^2}}$
(xii)
$\tilde{\mathbf{a}} = \sum_{i \in \mathbf{R}} \frac{e^{DSC(\overline{\mathbf{a}}, \mathbf{a}_i)} \cdot \mathbf{a}_i}{\sum_{j \in \mathbf{R}} e^{DSC(\overline{\mathbf{a}}, \mathbf{a}_j)}},$
$DSC(\overline{\mathbf{a}}, \mathbf{a}_i) = \frac{2 \cdot ||\overline{\mathbf{a}} \cdot \mathbf{a}_i||}{||\overline{\mathbf{a}}||^2 + ||\mathbf{a}_i||^2}$
(xiii)
$\tilde{\mathbf{a}} = \tilde{\mathbf{a}}_{(xii)} \cdot \beta + \tilde{\mathbf{a}}_{(xi)} \cdot (1-\beta)$
value increase

Fig. 2. **Pooling variants**. $\mathbf{R}$ denotes the kernel neighborhood as a set of pixels. (i-ii) **Average** and **maximum pooling** are based on the average or maximum activation value of the kernel region. (iii) **Power-average pooling** [1], [2] is proportional to average pooling raised to the power of ρ. When $\rho \rightarrow \infty$ the output equals MaxPooling, while $\rho = 1$ equals average pooling. (iv) **Sum pooling** is also proportional to average pooling with all kernel activations summed in the output. (v) **Stochastic pooling** [3] samples a random activation from the kernel region. (vi) **Stochastic Spatial Sampling** (S3Pool) [4] samples horizontal and vertical regions given a specified stride. (vi) **Gate pooling** [5] uses max-average pooling based on a gating mask (ω) and a sigmoid function. (viii) **Local Importance Pooling** (LIP) [6] uses a trainable sub-net G to enhance specific features. (ix-x) **$\mathbf{L_1}$ and $\mathbf{L_2}$ Inverse Distance Weighting Pooling** (IDW, ours) weighs kernel regions based on their inverse distance to the mean activation ($\overline{\mathbf{a}}$). (xi) **Exponential maximum pooling** (*e*mPool/SoftPool, ours) exponentially weighs activations using a softmax kernel. (xii) **Exponential Dice-Sørensen Coefficient Weighting Pooling** (*e*DSCWPool, ours) uses the exponent Dice-Sørensen Coefficient [7], [8] of the kernel activations ($\mathbf{a}_i$) and their average ($\overline{\mathbf{a}}$) as weights. (xiii) **Adaptive exponential pooling** (adaPool, ours) combines (xi) and (xii) with a trainable parameter β.

ing descriptive activation features. Consequently, this allows models with adaPool to consistently improve classification and recognition performance. AdaPool maintains a low computational cost and provides an approach for prior-information retainment. We further introduce adaUnPool and address super-resolution and interpolation tasks. Summarized, we make the following contributions:

- We introduce an approximated average pooling approach through Inverse Distance Weighting (IDW) [12]. We extend IDW from vector distance-based to vector similarity-based through the Dice-Sørensen Coefficient (DSC) and by utilizing its exponent *e*DSC to weight kernel elements.
- We propose adaPool, a parameterized learnable fusion of portions from the smooth approximation of the maximum and average. Using the inverse formulation, we develop upsampling process adaUnPool.
- We experiment on multiple global and local-based tasks including image and video classification, object detection and show consistent improvements by replacing original pooling layers with adaPool. We also demonstrate the improved performance of adaUnPool on image and video super-resolution and video frame interpolation.
- We introduce a high-resolution and frame-rate video processing dataset, *Inter4K*, for benchmarking of frame super-resolution and interpolation algorithms.

The remainder of the paper is structured as follows. We first discuss related work. We then detail our downsampling methods IDWPool, *e*mPool, *e*DSCPool and adaPool as well as upsampling method adaUnPool (Section III). We introduce Inter4K in Section IV and evaluate on global and local-based image and video tasks (Section V). We conclude in Section VI.

II. Related Work

Pooling hand-crafted features. Downsampling has been widely used in hand-coded feature extraction. In Bag-of-Words (BoW, [13]), images are represented as a group of local patches that are pooled and then encoded as vectors [14]. Based on this approach, Spatial Pyramid Matching (SPM) [15] aims at preserving spatial information. Later works extend this approach with linear SPM [16] that selects the maximum SIFT features in a spatial region. Most of the early works on feature pooling have focused on max-pooling based on the max-like behavior of biological cortex signals [17]. Maximum and average pooling studies in terms of information preservation by Boureau *et al.* [18] have suggested that max-pooling produces comparatively more representative results in low feature activation settings.

Pooling in CNNs. With the prevalence of learned feature approaches in various computer vision tasks, pooling methods have also been adapted to kernel-based operations, as shown in early CNN models. In CNNs, pooling has been mainly used to create condensed feature representations to reduce the model's computational requirements, and in turn to enable the creation of deeper architectures [19].

More recently, the preservation of relevant features during downsampling has taken a more prominent role. Initial ap-

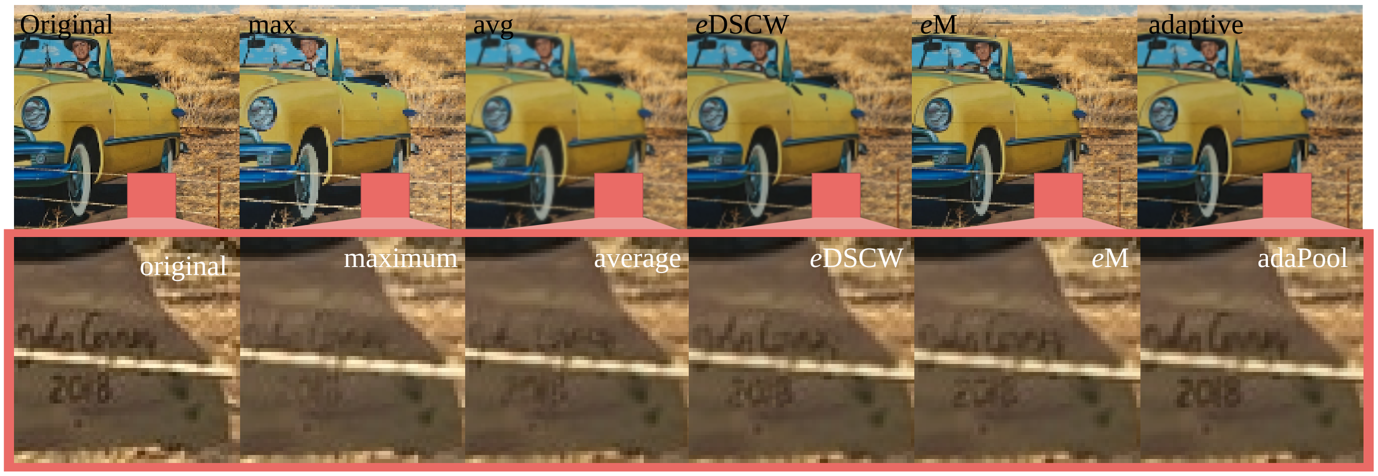


Fig. 3. **Example of detail preservation with different pooling methods**. Common methods such as average and maximum pooling result in a distorted signature with unrecognizable details such as numbers or characters. Exponential weighting through either normalized local maximum (*e*M) or similarity-based measures (*e*DSCW) better capture details. Further improvements in the detail and representation quality are observed when introducing an adaptive fusion between both of these exponential weighting methods (adaPool).

proaches include stochastic pooling [3], which uses a probabilistic weighted sampling of activations within a kernel region. Other pooling methods such as mixed pooling are based on a combination of maximum and average pooling, either probabilistically [20] or through a combination of portions from each method [5]. Power Average (L_p) [2] utilizes a learned parameter p to determine the relative importance of average and maximum pooling. With $p = 1$, sum pooling is used, while $p \rightarrow \infty$ corresponds to max-pooling.

Some approaches are based on grid-sampling. S3Pool [4] uses random sampling for both rows and columns of the original feature map grid to create the downsampled version. Methods can also employ learned weights such as in Detail Preserving Pooling (DPP, [21]) that uses average pooling while enhancing activations with above-average values. Local Importance Pooling (LIP, [6]) utilizes learned weights within a sub-network attention mechanism. A visual and mathematical overview of the operations performed by different pooling methods can be seen in Figure 2.

The majority of the pooling work reported in the literature cannot be inverted for upsampling. Badrinarayanan *et al.* [11] proposed an inversion of the maximum operation by tracking the in-kernel position of the selected maximum pixel while the other positions are populated by zero values in the upsampled output. This leads to the situation that the original values are used, but the output is inherently sparse. Recently, Zhao and Snoek [10] proposed LiftPool based on the use of four learnable sub-bands of the input. The produced output is composed as a mixture of the discovered sub-bands. They also propose an upsampling inversion of their approach (LiftUpPool). Both methods are based on sub-network structures that limit their usability as a computation- and memory-efficient pooling technique.

Most of the aforementioned methods rely on combinations of maximum and average pooling, or the inclusion of sub-networks that prohibit low-compute and efficient downsampling. Instead of combining existing methods, our work is based on an adaptive exponential weighting approach to improve the retention of information and to better preserve details of the original signal. Our proposed method, adaPool, is inspired by Luce's choice of axiom [22]. We thus weight kernel regions based on their relevance without being affected by the neighboring kernel items. This is in contrast to both average and maximum pooling. AdaPool uses two sets of pooling kernels. The first relies on a softmax weighting approach for amplifying feature activations of greater intensity. The second uses the channel-wise similarity of individual kernel items to their mean in order to determine their relevance. The similarities are discovered based on the Dice-Sørensen coefficient. Finally, outputs from both kernel operations are parametrically fused to a single volume. Parameters are specific to each kernel location thus making our approach regionally-adaptive.

A key property of adaPool is that gradients are calculated for each kernel item during backpropagation. This improves the network connectivity. In addition, downsampled regions are less likely to exhibit a vanishing trend of activations, as observed by equal-contribution approaches such as average or sum pooling. We demonstrate how adaPool can adaptively capture details in Figure 3, where the zoomed-in region displays a signature. AdaPool shows improvements in the clarity and recognizability of the letters and numbers.

III. Methodology

We start by introducing the operations that are shared in our pooling methods. We define local kernel region $\mathbf{R}$ in an activation map $\mathbf{a}$ of size $C \times H \times W$, with C channels, height H and width W. For notation simplicity, we omit the channel dimension and assume that $\mathbf{R}$ is the set of relative position indices corresponding to the activations in the 2D spatial region of size $k \times k$ (i.e., $|\mathbf{R}| = k^2$ activations). We denote the pooling output as $\widetilde{\mathbf{a}}$ and the corresponding gradients as $\nabla \widetilde{\mathbf{a}}_i$, where i is the set of coordinates within region $\mathbf{R}$.

A. *Inverse Distance Weighting pooling*

We first introduce Inverse Distance Weighting (IDW) pooling. IDW is widely applicable as a weighted average approach for multivariate interpolation [23], [24]. The assumption is that geometrically close observations exhibit a higher degree of resemblance than geometrically more distant ones. To assign a weight value, IDW relies on the measured observation distances within the region. A visual representation of this weighting process is shown in Figure 4.

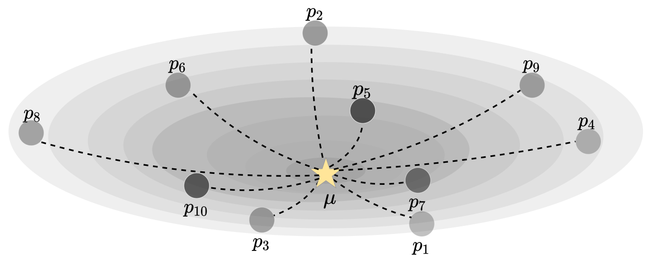


Fig. 4. **Inverse Distance Weighting**. Given multiple points $\{p_1, ..., p_n\}$ in a feature space and their mean (μ), their weights are equal to the inverse of their distance divided by their sum, following Luce's choice of axiom.

We argue that improvements in the calculation of the regional average can limit the effect of outlier input values in both the creation of pooled volumes in the forward pass, as well as gradient calculations in the backward pass. Current average calculations for pooling use equal weights for all input vectors within a kernel region. This implies that all vectors are deemed equally important with respect to their feature activations. The outcome is the composition of output regions that are strongly affected by outliers that are geometrically further from the region's average. To overcome the limitations of uniformly-weighted region averaging, we propose a weighted approach based on IDW, which we term *IDWPool*. We extend the concept of IDW to kernel weighting by using the distance of each activation $\mathbf{a}_i$, of relative pixel coordinate index $i \in \mathbf{R}$, to the mean activation $\overline{\mathbf{a}}$ of $\mathbf{R}$. The resulting pooled region $\underset{IDW}{\widetilde{\mathbf{a}}}$ is formulated as:

$$\underset{IDW}{\widetilde{\mathbf{a}}} = \begin{cases} \sum\limits_{i\in\mathbf{R}} \dfrac{\underset{IDW}{w(\overline{\mathbf{a}},\mathbf{a}_i)} \cdot \mathbf{a}_i}{\sum\limits_{j\in\mathbf{R}} \underset{IDW}{w(\overline{\mathbf{a}},\mathbf{a}_j)}},\; if\; d(\overline{\mathbf{a}},\mathbf{a}_i) \neq 0 \;\forall\; i \in \mathbf{R} \\ \mathbf{a}_i,\; if\; d(\overline{\mathbf{a}},\mathbf{a}_i) = 0 \;\exists\; i \in \mathbf{R} \end{cases} \quad (1)$$

The weights $\underset{IDW}{w(\cdot,\cdot)}$ are based on the inverse of the distance $d(\cdot,\cdot)$ between activations and the mean activation:

$$\underset{IDW}{w(\overline{\mathbf{a}},\mathbf{a}_i)} = \frac{1}{d(\overline{\mathbf{a}},\mathbf{a}_i)} \quad (2)$$

Distance function $d(\cdot,\cdot)$ can be calculated by any geometric distance approach. In our main results in Section V, we use the Euclidean distance (L_2) between the mean and the individual activations. We also provide results for the distance functions of Table I in Section V-F.

In comparison to uniformly-weighted averaging, IDWPool produces normalized results with higher weights for feature activation vectors geometrically closer to the mean. This also applies to the calculation of the gradients, and reduces the effect of outliers, providing a better representative update rate based on feature activation relevance. In that aspect, IDWPool works differently than the common approach of averaging all activations in which the output activation is not regularized.

Manhattan (L_1)	$\underset{L_1}{d} = \sum\limits_{c\in\mathbf{C}} \lvert\lvert \overline{\mathbf{a}}_c - \mathbf{a}_{i,c} \rvert\rvert$ (3)
Euclidean (L_2)	$\underset{L_2}{d} = \sum\limits_{c\in\mathbf{C}} \sqrt{\lvert\lvert \overline{\mathbf{a}}_c - \mathbf{a}_{i,c} \rvert\rvert^2}$ (4)
Huber [25]	$\underset{Hub}{d} = \begin{cases} \dfrac{d_{L_1}^2}{2},\; if\; d_{L_2} \leq \delta \\ \delta \cdot (d_{L_1} - \dfrac{\delta}{2}) \end{cases}$ (5)
Chebyshev [26]	$\underset{L_{Che}}{d} = \underset{c\in C}{max}\; d_{L_1}$ (6)
Gower [27]	$\underset{L_{Gow}}{d} = \dfrac{1}{C} \cdot d_{L_1}$ (7)

TABLE I
DISTANCE FUNCTIONS FOR VECTORS. ALL METHODS CAN BE APPLIED TO MULTI-DIMENSIONAL VECTOR VOLUMES.

B. Smooth approximated average pooling

We extend the use of weighted averaging in two ways. First, we argue that weighted averaging based on distance, over multi-dimensional spaces, is sub-optimal despite its advantages over the uniform approach. The L_1 or L_2 distance between a feature activation vector and the average over the region are calculated based on the mean, sum or maximum per-channel pair. The resulting distance is unbounded since the pair-wise distances are also unbounded. In addition, the calculated distance is sensitive to per-channel distance pair outliers. The effects of this are visible with the inverse distance weighting approaches for pooling in Figure 5. When using distance methods, the distance in certain channels can be significantly larger than in others. This creates the problem of weights that are nearing zero ($\underset{IDW}{w(\overline{\mathbf{a}}_{\mathbf{c}},\mathbf{a}_{j,c})} \rightarrow 0$).

Alternatively, the use of similarity measures could circumvent the bounding problem. But, specifically for the widely-used cosine similarity, we face the issue that the similarity between two vectors can be 1 even if one of the two vectors is infinitely large [28]. Fortunately, other dot-product methods for vector volumes such as the Dice-Sørensen Coefficient (DSC) overcome this limitation by taking into account the vector lengths. The weight calculation is done based on Equation 11.

We additionally consider other similarity-based methods to find the relevance of two volumes of vectors [29]. Apart

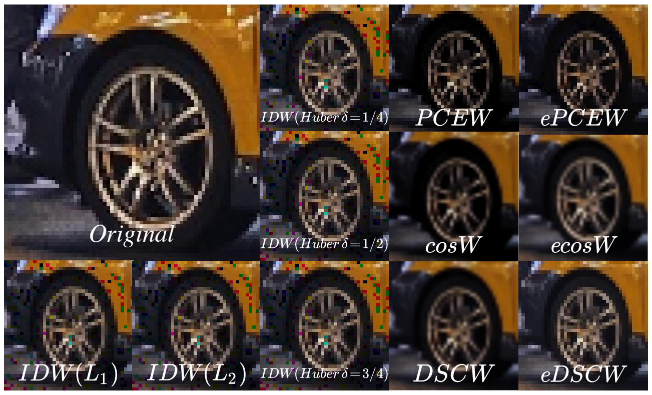


Fig. 5. **Instances of average distance/similarity weighting methods**. Kernel weights are determined based on IDW [12] with various inverse distance functions. Alternatively, weights are found based on similarity measures using the inner product ((*e*)PCEW, (*e*)cosW and (*e*)DSCW).

from the cosine similarity, the Kumar and Hassebrook Peak-to-correlation energy (PCE) [30] can be applied to vector volumes (as shown in Table II). We present the differences in the pooling quality based on different similarity methods in Figure 5. Considering the aforementioned shortfalls of cosine similarity, our use of DSC over PCE is primarily due to PCE's non-monotonic nature and value distribution [30].

Our second extension is the use of the exponent (e) of the similarity between the activations vector and the average activations. Given the IDW approach of Equation 1, zero-valued distances will be assigned a zero weight. This effectively makes the pooling method non-differentiable during backpropagation as not all gradients for every location will be calculated. This also amplifies the possibility for the vanishing gradients problem to arise when weights are close to zero. Through arithmetic underflow, their gradients may become zero. Based on the introduction of the exponent of the similarity coefficient, we re-formulate Equation 1 as:

$$\tilde{\mathbf{a}}_{eDSC} = \sum_{i\in\mathbf{R}} \frac{e^{\underset{DSC}{w}(\bar{\mathbf{a}},\mathbf{a}_i)}}{\sum\limits_{j\in\mathbf{R}} e^{\underset{DSC}{w}(\bar{\mathbf{a}},\mathbf{a}_j)}} \cdot \mathbf{a}_i \tag{8}$$

One of the key goals of downsampling is the preservation of informative features while reducing the spatial resolution of the input. The creation of downsampling volumes that do not fully capture the structural and feature appearances can have a negative impact on the performance. An example of such loss in detail can be seen in Figure 3. Average pooling will produce a decreased version of activations uniformly. IDWpool can improve on the activation preservation with weight values. But the weights are unbounded and can lead to vanishing gradients. Instead, using the exponent of the Dice-Sørensen Coefficient (eDSCWPool) provides a balanced approach of non-zero weight values while maintaining the beneficial properties of IDW.

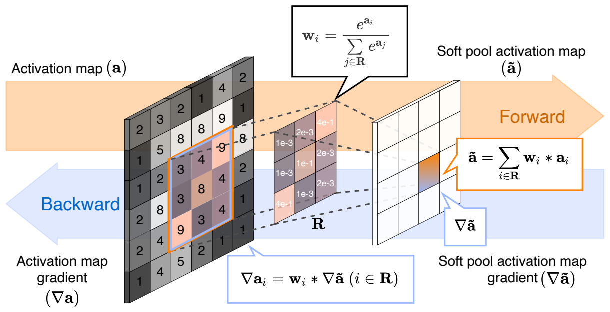
C. Smooth approximated maximum pooling

Similarly to creating a method for discovering the smooth approximated average within a kernel region we also discuss the formulation of downsampling based on the smooth approximated maximum which has been recently introduced as *SoftPool* [9]. For clarity, and in-line with the used terminology, we refer to SoftPool as the exponential maximum (eM).

Cosine	$\underset{cos}{S} = \frac{\sum\limits_{c\in\mathbf{C}} \bar{\mathbf{a}} \cdot \mathbf{a}_{i,c}}{\sqrt{\sum\limits_{c\in C} \bar{\mathbf{a}}_c^2} \cdot \sqrt{\sum\limits_{c\in C} \mathbf{a}_c^2}}$	(9)
PCE	$\underset{PCE}{S} = \frac{\sum\limits_{c\in\mathbf{C}} \bar{\mathbf{a}} \cdot \mathbf{a}_{i,c}}{\sum\limits_{c\in C} \bar{\mathbf{a}}_c^2 + \sum\limits_{c\in C} \mathbf{a}_c^2 - \sum\limits_{c\in C} \bar{\mathbf{a}}_c \cdot \mathbf{a}_c}$	(10)
DSC	$\underset{DSC}{S} = \sum\limits_{c\in\mathbf{C}} \frac{2 \cdot \lvert\lvert \bar{\mathbf{a}}_c \cdot \mathbf{a}_{i,c} \rvert\rvert}{\lvert\lvert \bar{\mathbf{a}}_c \rvert\rvert^2 + \lvert\lvert \mathbf{a}_{i,c} \rvert\rvert^2}$	(11)

TABLE II
SIMILARITY FUNCTIONS FOR VECTORS. BOTH METHODS CAN BE DIRECTLY APPLIED TO MULTI-DIMENSIONAL VECTOR VOLUMES.

The motivation behind the use of the exponential maximum is influenced by the cortex neural simulations of Riesenhuber and Poggio [31] and Boureau's *et al.* [18] downsampling hand-coded features. The method is based on the natural exponent (e), which ensures that larger activations will have a greater effect on the final output. The operation is differentiable, similarly to the exponential average, with kernel region activations assigned proportional gradients during backpropagation.



$\mathbf{w}_i = \frac{e^{\mathbf{a}_i}}{\sum_{j\in\mathbf{R}} e^{\mathbf{a}_j}}$
Activation map $(\mathbf{a})$
Soft pool activation map $(\tilde{\mathbf{a}})$
Forward
$\tilde{\mathbf{a}} = \sum_{i\in\mathbf{R}} \mathbf{w}_i * \mathbf{a}_i$
Backward
$\mathbf{R}$
$\nabla\tilde{\mathbf{a}}$
Activation map gradient $(\nabla\mathbf{a})$
$\nabla\mathbf{a}_i = \mathbf{w}_i * \nabla\tilde{\mathbf{a}}\ (i \in \mathbf{R})$
Soft pool activation map gradient $(\nabla\tilde{\mathbf{a}})$

Fig. 6. **Exponential maximum pooling calculation**. Forward operations, in orange, use as weights the softmax of each activation within the kernel region $\mathbf{R}$. Weights are also used for gradient calculations ($\nabla\tilde{\mathbf{a}}_i$), in blue.

The weights in exponential maximum pooling (eMPool) are used as non-linear transforms based on the value of the corresponding activation. Higher-valued activations will become more dominant than the lower-valued ones. As the majority of pooling operations are performed over high-dimensional feature spaces, highlighting the activations with greater effect is more balanced than the selection of the maximum activation. In the latter case, discarding the majority of the activations

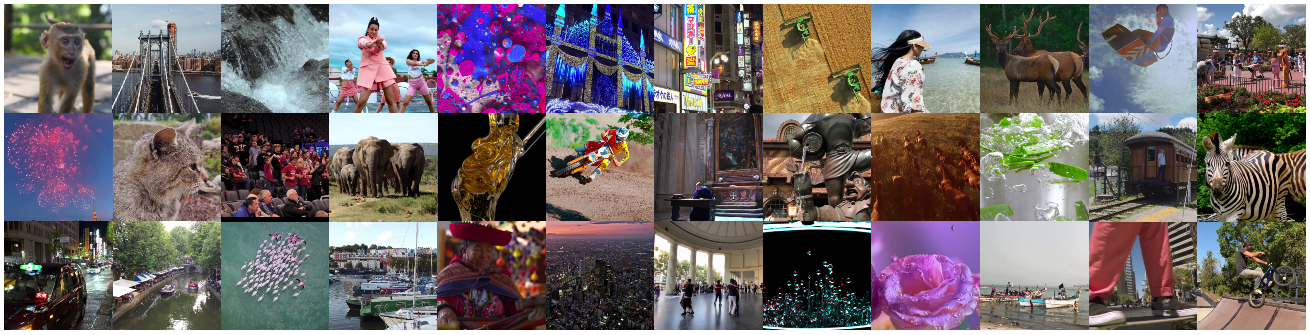


Fig. 7. **Video frame samples from Inter4K.** These samples show the high resolution quality (UHD/4K) and variation in the frames. The videos are challenging for video processing due to rapid motions and movements, complex lighting, textures and object detail.

presents the risk of loosing important information.

The output of *e*MPool is produced through a summation of all weighted activations within the kernel region $\mathbf{R}$:

$$\underset{em}{\widetilde{\mathbf{a}}} = \sum_{i\in\mathbf{R}} \underset{em}{w}(\mathbf{a}_i) \cdot \mathbf{a}_i,\ where\ \underset{em}{w}(\mathbf{a}_i) = \frac{e^{\mathbf{a}_i}}{\sum\limits_{j\in\mathbf{R}} e^{\mathbf{a}_j}} \tag{12}$$

In comparison to other max-based pooling approaches, the regional softmax of activations produces normalized results, similarly to *e*DSCWPool. But in contrast to the smooth exponential average, the normalized results are based on a probability distribution that is proportional to the values of each activation with respect to the neighboring activations within the kernel region. A visualization of a full forward and backward pass of information appears in Figure 6.

D. AdaPool: Adaptive exponential pooling

Finally, we propose an adaptive combination of the previous two smooth approximation-based pooling methods. Based on their properties, either *e*MPool or *e*DSCWPool demonstrates improvements on effectively preserving feature details. From Figure 3, neither of the two methods can be considered as generally superior than the other. Based on this observation, and in line with Lee *et al.*'s introduction of Avg/MaxPooling fusion strategies [5], we use a trainable parameter β to create a combined volume of both smooth approximated average and smooth approximated maximum. Here, β is used to learn the proportion that will be used from each of the two methods. Introducing β as part of the network training process has the advantage of creating a generalized pooling strategy that relies on the combination of the properties of both *e*MPool and *e*DSCWPool. We formulate the method as a weighted combination of the downsampled smooth approximated average ($\underset{eDSC}{\widetilde{\mathbf{a}}}$) and the smooth approximated maximum ($\underset{em}{\widetilde{\mathbf{a}}}$):

$$\underset{ada}{\widetilde{\mathbf{a}}} \overset{(8,9)}{=\!=} \underset{eDSC}{\widetilde{\mathbf{a}}} \cdot \beta + \underset{em}{\widetilde{\mathbf{a}}} \cdot (1-\beta) \tag{13}$$

where $\beta \in \{0, ..., 1\}$ is a weight matrix of the same size as the downsampled volume $\widetilde{\mathbf{a}}$ ($H' \times W'$). The gradients of β for backpropagation are calculated based on the chain rule as:

$$\frac{\partial E}{\partial \beta} = \frac{\partial E}{\partial \underset{ada}{\widetilde{\mathbf{a}}}} \frac{\partial \underset{ada}{\widetilde{\mathbf{a}}}}{\partial \beta} = \frac{\partial E}{\partial \underset{ada}{\widetilde{\mathbf{a}}}} (\max_i \mathbf{a}_i - \frac{1}{|R|} \sum_{i\in\mathbf{R}} \mathbf{a}_i) \tag{14}$$

E. Upsampling using adaUnPool

During pooling, information within a kernel region is condensed to a single output. The majority of the sub-sampling methods do not establish a mapping from the sub-sampled to the original input. Most tasks do not require this link but others, such as semantic segmentation [32]–[34], super-resolution [35]–[38] or frame interpolation [39]–[42] significantly benefit from it. As adaPool is differentiable and uses a minimum weight value assignment, the discovered weights can be used as prior knowledge during upsampling. We refer to the upsampling operation given adaPool weights as *adaUnPool*.

Given the pooled volume ($\widetilde{\mathbf{a}}$), we use the smooth approximated maximum ($\underset{em}{w}(\mathbf{a_i})$) and smooth approximated average weights ($\underset{eDSCW}{w}(\overline{\mathbf{a}}, \mathbf{a_i})$) with a learned value β. The final unpooled output ($\mathbf{a_i}$) for the ith kernel region ($i \in \mathbf{R}$) is computed as:

$$\mathbf{a}_i = \beta \cdot \frac{e^{\underset{DSC}{w}(\overline{\mathbf{a}},\mathbf{a}_i)}}{\sum\limits_{j\in\mathbf{R}} e^{\underset{DSC}{w}(\overline{\mathbf{a}},\mathbf{a}_j)}} \cdot \mathcal{I}_A(\widetilde{\mathbf{a}}) + (1-\beta) \cdot \underset{em}{w}(\mathbf{a}_i) \cdot \mathcal{I}_A(\widetilde{\mathbf{a}}) \tag{15}$$

where $\mathcal{I}_A(\cdot)$ interpolates by assigning the pooled volume ($\widetilde{\mathbf{a}}$) of the original kernel region at each position i within the region. The method is used to inflate the volume $H' \times W' \rightarrow H \times W$. The volumes are then determined with the corresponding smooth approximated average or maximum masks.

IV. The Inter4K video dataset

We introduce a novel video dataset to benchmark upsampling methods. Inter4K contains 1,000 ultra-high resolution videos with 60 frames per second (fps) from online resources. The dataset provides standardized video resolutions at ultra-high definition (UHD/4K), quad-high definition (QHD/2K), full-high definition (FHD/1080p), (standard) high definition (HD/720p), one quarter of full HD (qHD/520p) and one ninth of a full HD (nHD/360p). We use frame rates of 60, 50, 30, 24 and 15 fps for each resolution. Based on this standardization, both super-resolution and frame interpolation tests can be performed for different scaling sizes (×2, ×3 and ×4). In this paper, we use Inter4K to address frame upsampling and interpolation.

	Pooling method	DIV2K [43]						Urban100 [44]						Manga109 [45]					
		$k=2$		$k=3$		$k=5$		$k=2$		$k=3$		$k=5$		$k=2$		$k=3$		$k=5$	
		SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
	Average	0.714	51.247	0.578	44.704	0.417	29.223	0.691	50.380	0.563	41.745	0.372	28.270	0.695	54.326	0.582	43.657	0.396	29.862
	Maximum	0.685	49.826	0.370	41.944	0.358	22.041	0.662	48.266	0.528	40.709	0.330	20.654	0.671	50.085	0.544	41.128	0.324	22.307
	Pow-average	0.419	35.587	0.286	26.329	0.178	16.567	0.312	31.911	0.219	24.698	0.124	15.659	0.381	29.248	0.276	18.874	0.160	9.266
	Sum	0.408	35.153	0.268	26.172	0.193	17.315	0.301	31.657	0.208	24.735	0.123	15.243	0.374	30.169	0.271	20.150	0.168	13.081
Trainable	L_p [2]	0.686	49.912	0.542	43.083	0.347	25.139	0.676	48.508	0.534	39.986	0.326	26.365	0.675	51.721	0.561	41.824	0.367	27.469
	Gate [5]	0.689	50.104	0.560	43.437	0.353	25.672	0.675	49.769	0.537	40.422	0.328	26.731	0.679	51.980	0.569	42.127	0.374	27.754
	Lite-S3DPP [21]	0.702	50.598	0.562	44.076	0.396	27.421	0.684	49.947	0.551	40.813	0.365	27.136	0.691	52.646	0.573	42.794	0.386	28.598
	LIP [6]	0.711	50.831	0.559	44.432	0.401	28.285	0.689	50.266	0.558	41.159	0.370	27.849	0.689	53.537	0.579	43.018	0.391	29.331
Stoch.	Stochastic [3]	0.631	45.362	0.479	39.895	0.295	21.314	0.616	44.342	0.463	37.223	0.286	19.358	0.583	46.274	0.427	39.259	0.255	22.953
	S3 [4]	0.609	44.760	0.454	39.326	0.280	20.773	0.608	44.239	0.459	36.965	0.272	19.645	0.576	46.613	0.426	39.866	0.232	23.242
IDW (Ours)	L_1	0.724	51.415	0.596	44.739	0.418	29.576	0.696	50.723	0.573	41.796	0.366	28.205	0.712	54.614	0.574	43.756	0.395	30.224
	L_2	0.726	51.421	0.601	44.753	0.421	29.581	0.698	50.731	0.575	41.794	0.372	28.211	0.711	54.617	0.579	43.762	0.404	30.256
	Huber [25] $\delta=1/4$	0.728	51.465	0.611	44.813	0.427	29.736	0.702	50.734	0.579	41.862	0.383	28.576	0.714	54.620	0.584	43.897	0.412	30.445
	Huber [25] $\delta=1/2$	0.730	51.487	0.617	44.924	0.429	29.861	0.710	50.742	0.581	41.916	0.389	28.674	0.721	54.637	0.588	43.936	0.421	30.429
	Huber [25] $\delta=3/4$	0.727	51.459	0.606	44.846	0.421	29.728	0.705	50.737	0.576	41.874	0.384	28.612	0.716	54.626	0.581	43.885	0.409	30.432
exponential **(Ours)**	*e*MPool	0.729	51.436	0.594	44.747	0.421	29.583	0.694	50.687	0.578	41.851	0.394	28.326	0.704	54.563	0.586	43.782	0.403	30.114
	*e*DSCWPool	0.732	51.470	0.619	45.324	0.430	30.247	0.706	50.734	0.588	42.173	0.412	29.796	0.715	54.633	0.593	44.185	0.417	30.967
	adaPool ($\beta=1/4$)	0.730	51.523	0.605	44.979	0.429	29.956	0.712	50.733	0.582	41.970	0.397	28.861	0.709	54.625	0.589	43.994	0.412	30.843
	adaPool ($\beta=1/2$)	0.736	52.186	0.614	45.857	0.432	30.881	0.754	51.376	0.594	42.462	0.410	29.874	0.716	54.646	0.598	44.576	0.421	31.392
	adaPool ($\beta=3/4$)	0.742	53.341	0.618	46.173	0.438	31.764	0.741	51.132	0.605	42.996	0.418	30.163	0.731	55.284	0.602	45.344	0.427	31.638
	adaPool ($\nabla\beta$)	**0.778**	**53.769**	**0.624**	**46.892**	**0.443**	**32.216**	**0.769**	**52.438**	**0.614**	**43.637**	**0.425**	**30.845**	**0.747**	**55.890**	**0.609**	**46.253**	**0.436**	**32.794**

TABLE III
QUANTITATIVE RESULTS ON BENCHMARK HIGH-RES DATASETS. BEST RESULTS FOR EACH SETTING ARE DENOTED IN **BOLD**.

In contrast to other datasets used for video super-resolution and interpolation [46]–[52], Inter4K provides both standardized UHD resolution and 60 fps for all of videos. In addition, the dataset contains a diverse set of 1,000 5-second videos (examples are shown in Figure 7). Differences between scenes originate from the equipment (e.g., professional 4K cameras or phones), lighting conditions, variations in movements, actions or objects. The dataset is divided into 800 videos for training, 100 videos for validation and 100 videos for testing.

V. EXPERIMENTAL RESULTS

We initially evaluate the information loss by considering various pooling operators. We compare the downsampled outputs to the original inputs using standard similarity measures (Section V-B). In addition, we examine the computational overhead of each pooling operator (Section V-C).

We proceed by testing the performance on various classification tasks when we substitute the network's original pooling layers by *e*MPool, *e*DSCWPool and adaPool (Section V-D). This pooling layer substitution is performed on widely-used CNN architectures. We also provide results for different β parameters in a trainable or constant-value manner (Section V-E). Additionally, we evaluate several distance- and coefficient-based methods for the approximated average (Section V-F). Our final classification tests focus on comparisons to other pooling methods (Section V-G) and pooling layer replacements on the InceptionV3 [53] (Section V-H).

We present our results for object detection (Section V-I) on MS COCO [54] with RetinaNet [55] and Mask R-CNN [56] using several backbones. We experiment on spatio-temporal data by focusing on action recognition in video (Section V-J).

Lastly, we present our results on image super-resolution, frame interpolation and their combination (Section V-K).

Pooling	CPU (ms)	CUDA (ms)	Flicker2K [43]					
			$k=2$		$k=3$		$k=5$	
	($\downarrow$ F / $\uparrow$ B)	($\downarrow$ F / $\uparrow$ B)	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Avg	9 / 49	14 / 76	0.709	51.786	0.572	44.246	0.408	28.957
Max	91 / 152	195 / 267	0.674	47.613	0.385	40.735	0.329	21.368
Pow-avg	74 / 329	120 / 433	0.392	34.319	0.271	26.820	0.163	15.453
Sum	26 / 163	79 / 323	0.386	34.173	0.265	26.259	0.161	15.218
L_p [2]	116 / 338	214 / 422	0.683	48.617	0.437	42.079	0.341	24.432
Gate [5]	245 / 339	327 / 540	0.687	49.314	0.449	42.722	0.358	25.687
DPP [21]	427 / 860	634 / 1228	0.691	50.586	0.534	43.608	0.385	27.430
LIP [6]	134 / 257	258 / 362	0.696	50.947	0.548	43.882	0.390	28.134
Stoch. [3]	162 / 341	219 / 485	0.625	46.714	0.474	38.365	0.264	21.428
S3 [4]	233 / 410	345 / 486	0.611	46.547	0.476	37.706	0.252	21.363
L_1	N / A	48 / 227	0.712	51.985	0.574	44.814	0.411	29.246
L_2	N / A	49 / 231	0.714	51.592	0.576	44.832	0.416	29.251
Huber [25] $\delta=1/4$	N / A	51 / 234	0.726	51.879	0.583	44.916	0.421	29.458
Huber [25] $\delta=1/2$	N / A	51 / 234	0.727	52.053	0.593	45.231	0.424	29.635
Huber [25] $\delta=3/4$	N / A	51 / 234	0.723	51.912	0.584	45.105	0.418	29.476
*e*MPool	31 / 156	56 / 234	0.721	52.356	0.587	44.893	0.416	29.341
*e*DSCWPool	N / A	52 / 249	0.743	53.293	0.615	45.274	0.421	29.957
adaPool	N / A	119 / 490	**0.766**	**54.608**	**0.629**	**47.139**	**0.434**	**31.883**

TABLE IV
LATENCY AND PIXEL SIMILARITY. LATENCY FOR THE FORWARD AND BACKWARD PASS ON BOTH CPU AND GPU IS AVERAGED OVER ALL IMAGES IN IMAGENET1K. PIXEL SIMILARITY REPORTED ON FLICKER2K.

A. Experimental settings

Datasets. For our image-based experiments, we use seven different datasets for either image quality quantitative evaluation, image classification, object detection or image super-resolution. For the assessment of image quality and similarity, we use the high-resolution DIV2K [43], Urban100 [44], Manga109 [45], and Flicker2K [43] datasets. For image classification we use ImageNet1K [57] and MS COCO [54] for image object detection. For image super-resolution we employ the Urban100, Manga109 and B100 [58] datasets. For our video-based experiments, we employ six datasets. For action recognition, we use the large-scale HACS [59] and Kinetics-700 [60] datasets, as well as the smaller UCF-101 [49] dataset. For frame interpolation, we use Vimeo90K [52]

Model	Params (M)	GFLOP	Original (Baseline) top-1	top-5	*e*MPool top-1	top-5	*e*DSCWPool top-1	top-5	adaPool	
ResNet18	11.7	1.83	69.76	89.08	71.27 (+1.51)	90.16 (+1.08)	70.79 (+1.03)	89.96 (+0.88)	**71.78** (+2.02)	**90.65** (+1.57)
ResNet34	21.8	3.68	73.30	91.42	74.67 (+1.37)	92.30 (+0.88)	74.36 (+1.06)	92.15 (+0.73)	**75.43** (+2.13)	**92.87** (+1.45)
ResNet50	25.6	4.14	76.15	92.87	77.35 (+1.17)	93.63 (+0.76)	77.38 (+1.23)	93.90 (+1.03)	**78.42** (+2.27)	**94.16** (+1.29)
ResNet101	44.5	7.87	77.37	93.56	78.32 (+0.95)	94.21 (+0.65)	78.58 (+1.21)	94.42 (+0.86)	**79.59** (+2.22)	**94.88** (+1.32)
ResNet152	60.2	11.61	78.31	94.06	79.24 (+0.92)	94.72 (+0.66)	79.54 (+1.23)	94.98 (+0.92)	**80.74** (+2.43)	**95.08** (+1.02)
DenseNet121	8.0	2.90	74.65	92.17	75.88 (+1.23)	92.92 (+0.75)	76.06 (+1.41)	93.16 (+0.99)	**77.29** (+2.64)	**93.21** (+1.04)
DenseNet161	28.7	7.85	77.65	93.80	78.72 (+0.93)	94.41 (+0.61)	78.77 (+1.12)	94.53 (+0.73)	**80.10** (+2.35)	**94.87** (+1.07)
DenseNet169	14.1	3.44	76.00	93.00	76.95 (+0.95)	93.76 (+0.76)	77.19 (+1.19)	93.86 (+0.86)	**78.56** (+2.56)	**94.23** (+1.23)
ResNeXt50 32x4d	25.0	4.29	77.62	93.70	78.48 (+0.86)	93.37 (+0.67)	78.76 (+1.14)	94.48 (+0.78)	**79.98** (+2.36)	**94.82** (+1.12)
ResNeXt101 32x8d	88.8	7.89	79.31	94.28	80.12 (+0.81)	94.88 (+0.60)	80.57 (+1.26)	95.02 (+0.74)	**81.69** (+2.38)	**95.51** (+1.23)
Wide-ResNet50	68.9	11.46	78.51	94.09	79.52 (+1.01)	94.85 (+0.76)	79.61 (+1.10)	94.92 (+0.83)	**80.24** (+1.73)	**95.26** (+1.17)

TABLE V
Pairwise comparisons of top-1 and top-5 accuracies on ImageNet1K [57] between original networks and their counterparts with pooling replaced by *e*MPool, *e*DSCWPool and adaPool. All networks have been trained from scratch.

Pooling	Original (Baseline)				*e*MPool				*e*DSCWPool				adaPool			
Model \ Run	1	2	3	(best)	1	2	3	(best)	1	2	3	(best)	1	2	3	(best)
ResNet18	69.61	69.73	69.69	69.76	71.18	71.04	71.25	71.27	70.65	70.78	70.73	70.79	71.70	71.74	71.62	**71.78**
ResNet34	73.26	73.11	73.24	73.30	74.66	74.52	74.31	74.67	74.25	74.30	74.28	74.36	75.35	75.42	75.37	**75.43**
ResNet50	76.01	75.97	76.04	76.15	77.26	77.24	77.19	77.35	77.35	77.26	77.23	77.38	78.36	78.38	78.41	**78.42**

TABLE VI
Top-1 accuracy over runs on ImageNet1K [57] for original networks and those with *e*MPool, *e*DSCWPool and adaPool. We performed four runs for each combination of network and pooling type. The best run is denoted with (best).

and Middlebury [46] video processing datasets, as well as our newly introduced Inter4K dataset, which is also used for the combined task of frame interpolation and super-resolution.

Classification training scheme. For image classification, we use a random spatial region crop of size 294×294, which is then resized to 224×224. The initial learning rate across our experiments is set to 0.1 with an SGD optimizer. We train for a total of 100 epochs with a step-wise learning rate reduction every 40 epochs. For higher numbers of epochs, no further improvements were observed. The batch size is set to 256.

For our video action recognition tests, we use a multigrid training scheme [61], with frame sizes between 4–16 and frame crops of 90–256 depending on the cycle. The average video inputs are of size $8 \times 160 \times 160$, while the batch sizes are between 64 and 2048. The size for each of the batches is counter-equal to the input size in every step in order to match the memory use. We use the same learning rate, optimizer, learning rate schedule and maximum number of epochs as in the image-based experiments.

Object detection details. We first rescale the images to ensure that the smallest side has a minimum size of at least 800 pixels [56], [62]. If after rescaling the largest side is larger than 1024 pixels, we resize the entire image so that the largest side now is of 1024 pixels. Our rescaling and resizing preserves the aspect ratio of the images. We use the pre-trained networks from the image classification task as backbones. The learning rate is set to $1e{-}5$ and we use an SGD optimizer with momentum.

B. Downsampling similarity

In the first set of tests, we evaluate the information loss of our pooling operations. The comparisons focus on the similarity of the original inputs and downsampled outputs. Three widely used kernel sizes are employed ($k = \{2, 3, 5\}$). We use two standardized evaluation metrics [63]:

Structural Similarity Index Measure (SSIM) is calculated as the difference of two images in terms of their luminance, contrast, and a structural term. Larger SSIM values correspond to larger structural similarities.

Peak Signal-to-Noise Ratio (PSNR) is a quantification of the produced image's compression quality. PSNR takes into account the inverse of the Mean Squared Error (MSE) of two images' channels. Higher PSNR values translate to smaller channel-wise distances between the two images.

In Tables III and IV, we present the SSIM and PSNR values averaged over all images in DIV2K [43], Urban100 [44], Manga109 [45], and Flicker2K [43] datasets, for different kernel sizes. IDW-based distance methods outperform non-trainable and stochastic methods. The randomized policy of stochastic methods does not seem to allow to fully capture details. Additionally, the use of exponential weighting to our IDW-based methods yields clear improvements. Both *e*MPool and *e*DSCWPool are top-performing across kernel sizes and datasets. This demonstrates that the use of exponential approximation of the maximum or average benefits the downsampling of images. Finally, the combination of the two exponential methods into adaPool achieves the best overall performance when the fusion parameter (β) is learned.

C. Latency and memory use

Costs in terms of the memory and latency required by pooling operations are largely overlooked in literature as single operations have minor latency times and memory consumption. However, given potentially limited available resources, and the fact that operations are executed thousands of times per epoch, we advocate an evaluation of the running times and memory use. Slow or memory-intensive operations can have a detrimental effect on the performance and may become potential computational bottleneck points.

Computation overheads are reported in Table IV based on the inference over CPU and GPU (CUDA) for forward ($\downarrow F$) and backward ($\uparrow B$) passes over each operation. We observe that our implementations achieve low inference times on CUDA, while remaining memory-efficient. This is because of their simplicity and ease of parallelization which enable for running times similar to average pooling.

D. Image classification performance on ImageNet1K

We now test the assumption that a better preservation of information during downsampling of the exponential weighting method leads to an increase in image classification accuracy. Specifically, we replace the original pooling layers in ResNet [64], DenseNet [65], ResNeXt [66] and wide-ResNet [67] networks by our exponential pooling methods. Results appear in Table V for the classification of ImageNet1K images. In Table VI, we summarize the results of four runs over different training seeds for three models to ensure fair comparisons. The highest accuracies achieved are denoted by (best).

Throughout our experiments, we notice that networks with their pooling layers replaced by any of the exponential-weighting methods yield improved accuracy rates. Overall, the best results are obtained using adaPool pooling layers. We provide a discussion per CNN architecture.

ResNet [64]. We notice an average of 2.19 % top-1 and 1.33% top-5 improvement on ResNet models when replacing their pooling layers with adaPool. Improvements in accuracy are also observed with replacements based on both *e*MPool and *e*DSCWPool with an average +1.17% and +1.15% top-1 accuracy, respectively. ResNet architectures include only a single pooling operation after the first convolution layer. The improvements from replacing only a single layer demonstrate the benefits of adaPool for image classification. In Table VI, we do not notice a significant divergence in accuracy over multiple runs on ResNet18, ResNet34 and ResNet50 networks. On average, a replacement with adaPool can improve by +2.01% the original ResNet18 across runs, by +2.24% ResNet34 and +2.38% ResNet50.

DenseNet [65]. DenseNets include five pooling layers. Our replacements concern the MaxPooling layer after the first convolution and the four average pooling layers between dense blocks. The average top-1 accuracy gains based on layer replacements with adaPool are between 2.35–2.64%. More modest increases are found for *e*MPool and *e*DSCWPool with +(0.93–1.23)% and +(1.12–1.41)%, respectively.

ResNeXt [66]. We note an average of 2.37% top-1 and 1.17% top-5 accuracy improvement with adaPool. An average gain of 1.20% and 0.76% for the top-1 and top-5 accuracies are observed with pooling layer replacement with *e*DSCWPool. For *e*MPool, these improvements are 0.83% and 0.64% for the top-1 and top-5 accuracies.

Wide-Resnet-50 [67]. On Wide-ResNet50, we observe the best top-1 accuracy of 80.24% with a 1.73% improvement when we replace the original pooling layers with adaPool. Gains in performance are also observed for *e*MPool with +1.01% and *e*DSCWPool with 1.10%.

Mode	β value	ResNet18		ResNet34		InceptionV3	
		top-1	top-5	top-1	top-5	top-1	top-5
Constant	β=1/8	71.31	90.21	74.83	92.42	78.98	93.77
	β=1/4	71.34	90.26	74.76	92.38	79.23	93.84
	β=3/8	71.31	90.19	74.63	92.34	79.35	93.92
	β=1/2	71.28	90.07	74.56	92.31	79.54	93.89
	β=5/8	71.16	90.02	74.48	92.28	79.68	94.01
	β=3/4	71.19	89.95	74.38	92.16	79.97	94.05
	β=7/8	71.04	89.96	74.41	92.20	80.16	94.19
trainable		**71.78**	**90.65**	**75.43**	**92.87**	**81.34**	**94.57**

TABLE VII
EFFECT OF β ON IMAGENET1K IMAGE CLASSIFICATION. LARGER VALUES OF β CORRESPOND TO STRONGER RELIANCE ON *e*DSCWPOOL WHILE SMALLER β VALUES PRIORITIZE *e*MPOOL. BEST RESULTS ARE IN **BOLD** WHILE SECOND BEST ARE UNDERLINED.

E. Effect of βarameter value

In order to study how different combinations of the approximated maximum and average effect our proposed adaPool method, we present results in Table VII on ImageNet1K with several constant β values and study the performance gains when β is converted to a trainable parameter.

Method			top-1	top-5
		Original (Baseline)	69.76	89.08
IDW		L_1	69.94 (+0.18)	89.24 (+0.16)
		L_2	70.02 (+0.23)	89.28 (+0.20)
	Huber [25]	$\delta = 1/4$	70.11 (+0.35)	89.33 (+0.25)
		$\delta = 1/2$	70.09 (+0.33)	89.27 (+0.19)
		$\delta = 3/4$	70.13 (+0.37)	89.32 (+0.24)
		Chedyshev	69.96 (+0.20)	89.20 (+0.12)
		Gower	69.58 (-0.18)	88.94 (-0.14)
Sim.		Cosine	70.45 (+0.69)	89.44 (+0.36)
		PCE	70.54 (+0.78)	89.51 (+0.43)
		DSC	**70.66** (+0.90)	**89.77** (+0.69)

TABLE VIII
IMAGENET1K CLASSIFICATION WITH DISTANCE/SIMILARITY-BASED POOLING ALTERNATIVES ON RESNET18. DISTANCE-BASED METHODS ARE DENOTED BY IDW, WHILE SIMILARITY-BASED METHODS ARE DENOTED WITH SIM.

F. Comparison with alternative soft average methods

To evaluate the effect of different distance and similarity measures for average-approximating pooling, in image classification performance, we use a ResNet18 as backbone. We set as baseline the original ResNet18 with MaxPooling.

The results in Table VIII show negligible differences between distances in IDW pooling. Huber-based pooling shows small top-1 accuracy improvements, in the range of +(0.10–0.19)% over L_1, L_2 and Chedyshev distance-weighting. A slight performance reduction is observed with the Gower method. This could be because of the production of small weight values as Gower uses the L_1 distance divided by the number of channels (Equation 7).

In contrast to distance approaches, similarity measures show a larger increase over the baseline model. This can be attributed to the sparsity of the per-pixel volumes. Considering the relatively small number of kernel pixels ($k \times k$) and the high-dimensional spaces that they are represented in, distances between points and the mean of thereof are large [68]. The Dice-Sørensen coefficient is the most effective method with 70.66% and 89.77% for the top-1 and top-5 accuracies. Increases are observed by the exponent of DSC in *e*DSCWPool shown in Table V at the row for ResNet18, with 70.79% top-1 and 90.16% top-5 accuracies.

Pooling	Networks					
	ResNet18	ResNet34	ResNet50	ResNeXt50	DenseNet121	InceptionV1
Original (Baseline)	(Max) 69.76	(Max) 73.30	(Max) 76.15	(Max) 77.62	(Avg+Max) 74.65	(Max) 69.78
Stochastic [3]	70.13	73.34	76.11	77.71	74.84	70.14
S3 [4]	70.15	73.56	76.24	77.82	74.85	70.17
L_p [5]	70.45	73.74	76.56	77.86	74.93	70.32
Gate [2]	70.74	73.68	76.75	77.98	74.88	70.52
DPP [21]	70.86	74.25	77.09	78.20	75.37	70.95
LIP [6]	70.83	73.95	77.13	78.14	75.31	70.77
*e*MPool **(ours)**	71.27	74.67	77.35	78.48	75.88	71.43
*e*DSCWPool **(ours)**	70.79	74.36	77.38	78.76	76.06	71.85
adaPool **(ours)**	**71.78**	**75.43**	**78.42**	**79.98**	**77.29**	**72.56**

TABLE IX
POOLING LAYER SUBSTITUTION TOP-1 ACCURACY FOR A VARIETY OF POOLING METHODS. EXPERIMENTS WERE PERFORMED ON IMAGENET1K.

G. Comparison with alternative pooling methods

We provide quantitative comparisons between different pooling methods over six different models in Table IX. We systematically replaced the original pooling layers of the original model (baseline). Non-adaptive *e*MPool and *e*DSCWPool still outperform stochastic methods while the obtained accuracies are similar to those of learnable methods. Across the tested architectures, adaPool outperforms other learnable and stochastic pooling methods. The largest overall margins are observed for InceptionV1 with improvements over other methods in the range of 1.61–2.78% and on DenseNet121 (1.92–2.64%).

Layer	Pooling layer substitution with adaPool							
	N	I	II	III	IV	V	VI	VII
$pool_1$		✓	✓	✓	✓	✓	✓	✓
$pool_2$			✓	✓	✓	✓	✓	✓
$mixed\ 5_{b-d}$				✓	✓	✓	✓	✓
$mixed\ 6_a$					✓	✓	✓	✓
$mixed\ 6_{b-e}$						✓	✓	✓
$mixed\ 7_a$							✓	✓
$mixed\ 7_{b-d}$								✓
Top-1 (%)	77.45	78.34	78.89	79.32	79.78	80.21	80.54	**81.34**
Top-5 (%)	93.56	93.77	93.92	94.05	94.17	94.26	94.32	**94.57**

TABLE X
PROGRESSIVE LAYER SUBSTITUTION FOR INCEPTIONV3 ON IMAGENET1K. COLUMN NUMBERS REFER TO THE NUMBER OF REPLACED POOLING LAYERS, MARKED WITH ✓.

H. Layer-wise ablation on InceptionV3

To understand the effect of adaPool at different network depths, we hierarchically ablate over pooling layers of the InceptionV3 architecture. This choice is primarily based on the Inception block's structure that includes pooling operations. This allows for a per-block evaluation of the change in the pooling operator.

From results summarized in Table X, we notice that with each additional replacement of an original pooling operation by adaPool, we can expect an average increase of +0.56% in top-1 accuracy. While the performance gains are systematic, the largest improvements are observed on replacements over the first pooling operation after the initial convolutional layer ($pool_1$) with a +0.89% jump in accuracy and at the final Inception block ($mixed7_{b-d}$) with a +0.80% increase. We thus demonstrate that adaPool yields accuracy improvement through its adaptive weighting, regardless of the network depth and number of channels.

I. Object detection performance on MS COCO

To investigate the merits of our proposed exponentially-weighted pooling on encapsulating relevant local information, we present results for object detection on MS COCO [54] in Table XI. We use RetinaNet [55] and Mask-RCNN [56] with several different backbone networks. We chose these two models based on their wide popularity. Overall, we observe that both *e*MPool and *e*DSCWPool come with average precision (AP) improvements of 1.0% and 0.86%, respectively. A 2.4% increase over the original models is observed for adaPool. Similar trends in AP are also visible for AP_{50} and AP_{75}, demonstrating that adaPool does not only benefit tasks that rely primarily on general features such as classification, but also provides a performance boost for local-based feature tasks such as object detection.

J. Video classification performance

We now turn to the evaluation of our pooling operators on spatio-temporal data by focusing on the task of action recognition in videos. The accurate classification and representation of space-time features stands as a major challenge in the field of video understanding [69].

The majority of space-time networks are based on the extension of 2D convolutions to 3D to include the temporal dimension. Stacks of frames are used as inputs. Similarly, in our tested, the only modification in our method is the inclusion of the temporal dimension over in kernel region $\mathbf{R}$.

For our tests, we first train models from scratch on HACS [59] using author implementations. These models are then used to initialize the weights for the Kinetics-700 and UCF-101 tests. SlowFast [75] and ir-CSN-101 [73] are the only two models that use different initialization weights, with ir-CSN-101 pre-trained on IG65M and SlowFast on ImageNet.

We report in Table XII the performance of three spatio-temporal CNNs with pooling layers replaced by adaPool. We observe state-of-the-art performance using MTNet$_L$ with adaPool on HACS and Kinetics-700, with 87.83% and 64.67% top-1 accuracies, respectively. This corresponds to an increase of 1.21% and 1.36% over the same networks with the original pooling layers. This also comes with negligible additional GFLOPs (+0.2). On UCF-101, we show that both MTNet$_L$ and SRTG r3d-101 with adaPool outperform the original and the

Model	Backbone	Original (Baseline)						*e*MPool						*e*DSCWPool						adaPool					
		AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
RetinaNet [55]	ResNet18	28.3	48.7	31.6	12.6	33.6	40.9	29.7	50.2	33.3	14.1	35.2	42.6	28.9	49.6	32.8	13.8	34.7	41.5	**31.2**	**51.4**	**34.7**	**15.4**	**36.5**	**43.4**
	ResNet34	31.6	50.8	33.9	15.1	36.0	43.6	32.8	52.1	35.5	16.2	37.3	45.0	32.4	51.4	34.8	15.9	36.8	44.7	**33.6**	**53.4**	**36.4**	**16.9**	**38.2**	**44.7**
	ResNet50	34.0	52.5	36.5	17.0	37.4	45.1	34.9	53.4	37.6	18.0	38.5	46.4	34.6	53.1	37.2	17.7	38.2	46.1	**35.6**	**53.9**	**38.0**	**18.4**	**39.1**	**47.2**
	ResNet101	39.1	59.1	42.3	21.8	42.7	50.2	39.8	59.9	43.3	22.4	43.5	51.1	40.1	60.3	43.7	22.6	43.9	51.4	**40.8**	**61.6**	**44.8**	**23.7**	**44.8**	**52.5**
Mask R-CNN [56]	ResNet34	32.9	53.6	32.7	14.5	35.1	43.2	34.0	54.8	34.1	15.7	36.6	44.6	33.8	54.1	33.6	15.3	36.2	44.0	**35.7**	**56.9**	**36.4**	**16.8**	**38.6**	**46.5**
	ResNet50	33.6	55.2	35.3	15.4	36.8	45.3	34.5	56.2	36.4	16.2	37.7	46.3	34.4	56.2	36.3	16.3	37.5	46.2	**36.3**	**57.5**	**36.9**	**17.1**	**39.0**	**47.3**
	ResNet101	38.2	60.3	41.7	20.1	41.1	50.2	39.0	61.1	42.6	20.9	42.0	51.3	39.5	61.7	43.1	21.5	42.8	51.9	**42.4**	**62.8**	**45.1**	**24.5**	**45.6**	**52.8**

TABLE XI
OBJECT DETECTION BOUNDING BOX AP RESULTS ON MS COCO `TEST-DEV` FOR MODELS WITH ORIGINAL BACKBONE NETWORKS AND THE SAME NETWORKS WITH POOLING LAYERS REPLACED BY SOFTPOOL. ALL MODELS ARE PRE-TRAINED ON IMAGENET1K [57].

Model	GFLOPs	HACS		K-700		UCF-101	
		top-1(%)	top-5(%)	top-1(%)	top-5(%)	top-1(%)	top-5(%)
r3d-101 [70]**	78.5	80.49	95.18	52.58	74.63	95.76	98.42
r(2+1)d-50 [71]**	50.0	81.34	94.51	49.93	73.40	93.92	97.84
I3D [72]‡*	55.3	79.95	94.48	53.01	69.19	92.45	97.62
ir-CSN-101 [73]‡†	17.3	N/A	N/A	54.66	73.78	95.13	97.85
SRTG r3d-101 [74]††	78.7	81.66	96.37	56.46	76.82	97.32	99.56
SlowFast r3d-50 [75]‡†	36.7	N/A	N/A	56.17	75.57	94.62	98.75
MTNet_M [76]††	5.8	83.39	95.86	58.43	77.62	95.42	98.07
MTNet_L [76]††	17.6	86.62	96.68	63.31	84.14	97.38	99.23
r3d-50 with adaPool **(Ours)**	53.2	81.13	94.96	50.87	74.06	94.21	97.76
SRTG r(2+1)d-50 with adaPool **(Ours)**	50.1	85.24	98.13	56.43	76.21	97.17	99.04
SRTG r3d-101 with adaPool **(Ours)**	78.7	84.37	97.84	58.62	78.56	98.53	**99.86**
MTNet_L with adaPool **(Ours)**	17.8	**87.83**	**98.21**	**64.67**	**84.78**	**98.60**	99.74

** re-implemented models trained from scratch. †† models and weights from official repositories. ‡* unofficial models trained from scratch. ‡† models from unofficial repositories with official weights.

TABLE XII
ACTION RECOGNITION TOP-1 AND TOP-5 ACCURACY FOR HACS, K-700 AND UCF-101. MODELS ARE TRAINED ON HACS AND FINE-TUNED ON K-700 AND UCF-101, EXCEPT FOR IR-CSN-101 AND SLOWFAST R3D-50 (SEE TEXT). N/A MEANS NO TRAINED MODEL WAS PROVIDED.

other top-performing models. Increases in top-1 performance are also observed for SRTG r3d-101 with +2.71% on HACS and +1.47% on Kinetics-700.

These results demonstrate, once again, that the simple replacement of a pooling operator by adaPool consistently results in a modest performance gain. Even for the almost saturated performance on UCF-101, using adaPool results in a performance increase of 1.22% on MTNet_L.

Scale	Model	Urban100 [52]		Manga109 [45]		B100 [58]	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
2x	SRCNN [77]	29.50	0.8946	35.60	0.9663	31.36	0.8879
	RCAN [78]	33.34	0.9384	39.44	0.9786	32.41	0.9027
	SAN [79]	33.10	0.9370	39.32	0.9792	32.42	0.9028
	HAN+ [80]	33.53	0.9398	39.62	0.9787	32.41	0.9027
	RCAN w/ adaP/U	33.58	0.9456	39.67	0.9834	32.63	0.9103
	HAN+ w/ adaP/U	**33.72**	**0.9469**	**39.82**	**0.9841**	**32.79**	**0.9187**
4x	SRCNN [77]	24.52	0.7221	27.58	0.8555	26.90	0.7101
	RCAN [78]	26.82	0.8087	31.22	0.9173	27.77	0.7436
	SAN [79]	26.79	0.8068	31.18	0.9169	27.78	0.7436
	HAN+ [80]	27.02	0.8131	31.73	0.9207	27.85	0.7454
	RCAN w/ adaP/U	27.24	0.8195	31.78	0.9243	**28.11**	0.7482
	HAN+ w/ adaP/U	**27.96**	**0.8246**	**32.30**	**0.9286**	28.06	**0.7513**

TABLE XIII
IMAGE SUPER-RESOLUTION WITH 2X AND 4X UPSAMPLING. BEST AND SECOND BEST RESULTS ARE SHOWN IN **BOLD** AND UNDERLINED.

K. Image super-resolution and frame interpolation results

In order to assess if the adaptive pooling weights are beneficial for upsampling with adaUnPool, we experiment on image super-resolution, video frame interpolation and their join task. For each task we replace pooling layers with adaPool and the respective upsampling layers with adaUnPool.

Model	Vimeo90K [52]			Middlebury [46]			Inter4K (4K, 30$\rightarrow$60fps)		
	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
DAIN [39]	34.70	0.964	0.022	36.70	0.965	0.017	35.48	0.959	0.021
CAIN [81]	34.65	0.959	0.020	35.11	0.951	0.019	34.92	0.953	0.019
BMBC [82]	35.06	0.964	0.015	36.79	0.965	0.015	35.76	0.966	0.015
XVFI [83]	34.27	0.971	N/A	N/A	N/A	N/A	35.28	0.969	0.018
CDFI [84]	35.17	0.964	0.010	37.14	0.966	0.007	36.31	0.967	0.010
DAIN w/ adaP/U	34.96	0.968	0.017	36.82	0.968	0.015	35.73	0.964	0.012
CDFI w/ adaP/U	**35.23**	**0.972**	**0.008**	**37.22**	**0.970**	**0.006**	**36.57**	**0.972**	**0.007**

TABLE XIV
QUALITATIVE FRAME INTERPOLATION RESULTS ON VIMEO90K, MIDDLEBURY AND INTER4K. N/A INDICATES THAT THE AUTHORS OF THE ORIGINAL WORKS DO NOT PROVIDE THE SPECIFIC RESULTS.

Scale	Resolution and fps conversions	Measures		
		PSNR	SSIM	LPIPS
2x	nHD15fps $\rightarrow$ HD30fps	33.95	0.936	0.018
	qHD24fps $\rightarrow$ FHD50fps	33.91	0.928	0.020
	HD30fps $\rightarrow$ QHD60fps	33.87	0.925	0.021
	FHD30fps $\rightarrow$ UHD60fps	33.81	0.918	0.021
4x	nHD15fps $\rightarrow$ QHD60fps	25.32	0.822	0.028
	qHD15fps $\rightarrow$ UHD60fps	25.38	0.819	0.031

TABLE XV
FRAME INTERPOLATION AND SUPER-RESOLUTION WITH CDFI ON INTER4K. THE RESOLUTION AND FPS OF THE ORIGINAL AND PROCESSED VIDEOS ARE INDICATED IN THE SECOND COLUMN.

Our comparisons on image super-resolution are shown in Table XIII. Both RCAN [78] and HAN+ [80] perform favorably with down and up-sampling layers substituted by ada(Un)Pool. We observe that, in both cases of $2\times$ and $4\times$ image upsampling, our converted networks not only outperform their original implementations but also all other methods.

We demonstrate the merits of replacing all down-sampling layers with ada(Un)Pool for frame interpolation in Table XIV. The two converted networks, DAIN [39] and CDFI [84], produce improved results across the tested datasets. CDFI with adaPool and adaUnPool yields state-of-the-art results on both Vimeo90K and Middlebury as well as on our Inter4K for 4K video interpolation from 30 to 60 fps.

We also preform tests over the combined task of frame super-resolution and interpolation and report our findings in Table XV on the Inter4K dataset. We use CDFI with adaPool and adaUnPool. From Table XV, in general, results are within range between conversions in both scales (2×, 4×). Some slight degradation in performance is observed on high-resolution, high-frame-rate conversions.

VI. Conclusion

In this paper, we have proposed a pooling method for the preservation of informative features based on adaptive exponential weighting. The parameterized fusion of the exponential maximum eMPool and exponential average eDSCWPool creates a regionally-adaptive method that we name adaPool. The weights of adaPool can be used to invert the pooling operation (adaUnPool), to achieve upsampling. We have tested our approach on image and video classification, image similarity, object detection, image and frame super-resolution tasks, as well as frame interpolation to demonstrate the merits of our proposed approach when faced with various challenges such as capturing global and local information, or to consider 2D image data and 3D video data. Over all tasks, and using a variety of network backbones and experiment settings, adaPool consistently outperforms any other method while computational latencies and memory use remain modest. Based on these extensive experiments, we believe adaPool is a good alternative for currently popular pooling operators.

Acknowledgment

The authors thank the Netherlands Organization for Scientific Research (NWO) for the support through TOP-C2 grant "Automatic recognition of bodily interactions" (ARBITER).

References

[1] J. B. Estrach, A. Szlam, and Y. LeCun, "Signal recovery from pooling representations," in *ICML*, 2014, pp. 307–315.

[2] C. Gulcehre, K. Cho, R. Pascanu, and Y. Bengio, "Learned-norm pooling for deep feedforward and recurrent neural networks," in *ECML PKDD*, 2014, pp. 530–546.

[3] M. D. Zeiler and R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," in *ICLR*, 2013.

[4] S. Zhai, H. Wu, A. Kumar, Y. Cheng, Y. Lu, Z. Zhang, and R. Feris, "S3pool: Pooling with stochastic spatial sampling," in *CVPR*, 2017, pp. 4970–4978.

[5] C.-Y. Lee, P. W. Gallagher, and Z. Tu, "Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree," in *AISTATS*, 2016, pp. 464–472.

[6] Z. Gao, L. Wang, and G. Wu, "LIP: Local importance-based pooling," in *ICCV*, 2019.

[7] L. R. Dice, "Measures of the amount of ecologic association between species," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[8] T. J. Sørensen, *A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons*. I kommission hos E. Munksgaard, 1948.

[9] A. Stergiou, R. Poppe, and K. Grigorios, "Refining activation downsampling with softpool," in *ICCV*, 2021, pp. 10 357–10 366.

[10] J. Zhao and C. G. Snoek, "LiftPool: Bidirectional convnet pooling," in *ICLR*, 2021.

[11] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE TPAMI*, vol. 39, no. 12, pp. 2481–2495, 2017.

[12] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *ACM*, 1968, pp. 517–524.

[13] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *ECCVW*, 2004, pp. 1–22.

[14] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong, "Locality-constrained linear coding for image classification," in *CVPR*, 2010, pp. 3360–3367.

[15] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *CVPR*, 2006, pp. 2169–2178.

[16] J. Yang, K. Yu, Y. Gong, and T. Huang, "Linear spatial pyramid matching using sparse coding for image classification," in *CVPR*, 2009, pp. 1794–1801.

[17] T. Serre, L. Wolf, and T. Poggio, "Object recognition with features inspired by visual cortex," in *CVPR*, 2005, pp. 994–1000.

[18] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *ICML*, 2010, pp. 111–118.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015, pp. 1–9.

[20] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *RSKT*, 2014, pp. 364–375.

[21] F. Saeedan, N. Weber, M. Goesele, and S. Roth, "Detail-preserving pooling in deep networks," in *CVPR*, 2018, pp. 9108–9116.

[22] R. D. Luce, "The choice axiom after twenty years," *Elsevier J. Math Psychol.*, vol. 15, no. 3, pp. 215–233, 1977.

[23] H. Akima, "A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points," *ACM TOMS*, vol. 4, no. 2, pp. 148–159, 1978.

[24] R. Franke, "Scattered data interpolation: Tests of some methods," *J. Math. of comp.*, vol. 38, no. 157, pp. 181–200, 1982.

[25] P. J. Huber, "Robust estimation of a location parameter," in *Breakthroughs in statistics*. Springer, 1992, pp. 492–518.

[26] F. Van Der Heijden, R. P. Duin, D. De Ridder, and D. M. Tax, *Classification, parameter estimation and state estimation: an engineering approach using MATLAB*. John Wiley & Sons, 2005.

[27] J. C. Gower, "A general coefficient of similarity and some of its properties," *Biometrics*, pp. 857–871, 1971.

[28] B. Fernando and S. Herath, "Anticipating human actions by correlating past with the future with Jaccard similarity measures," in *CVPR*, 2021, pp. 13 224–13 233.

[29] S.-H. Cha, "Comprehensive survey on distance/similarity measures between probability density functions," *IJMMAS*, vol. 1, no. 2, p. 1, 2007.

[30] B. V. Kumar and L. Hassebrook, "Performance measures for correlation filters," *Applied optics*, vol. 29, no. 20, pp. 2997–3006, 1990.

[31] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature neuroscience*, vol. 2, no. 11, pp. 1019–1025, 1999.

[32] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," in *CVPRW*, 2017, pp. 11–19.

[33] G. Lin, A. Milan, C. Shen, and I. Reid, "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation," in *CVPR*, 2017, pp. 1925–1934.

[34] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, "Understanding convolution for semantic segmentation," in *WACV*, 2018, pp. 1451–1460.

[35] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE TPAMI*, vol. 40, no. 4, pp. 834–848, 2017.

[36] W. Li, X. Tao, T. Guo, L. Qi, J. Lu, and J. Jia, "Mucan: Multi-correspondence aggregation network for video super-resolution," in *ECCV*. Springer, 2020, pp. 335–351.

[37] L. Lu, W. Li, X. Tao, J. Lu, and J. Jia, "MASA-SR: Matching acceleration and spatial adaptation for reference-based image super-resolution," in *CVPR*, 2021, pp. 6368–6377.

[38] Z. Wang, J. Chen, and S. C. Hoi, "Deep learning for image super-resolution: A survey," *IEEE TPAMI*, vol. 43, no. 10, pp. 3365–3387, 2020.

[39] W. Bao, W.-S. Lai, C. Ma, X. Zhang, Z. Gao, and M.-H. Yang, "Depth-aware video frame interpolation," in *CVPR*, 2019, pp. 3703–3712.

[40] Y.-L. Liu, Y.-T. Liao, Y.-Y. Lin, and Y.-Y. Chuang, “Deep video frame interpolation using cyclic frame generation,” in *AAAI*, vol. 33, no. 01, 2019, pp. 8794–8802.

[41] S. Niklaus and F. Liu, “Context-aware synthesis for video frame interpolation,” in *CVPR*, 2018, pp. 1701–1710.

[42] ——, “Softmax splatting for video frame interpolation,” in *CVPR*, 2020, pp. 5437–5446.

[43] E. Agustsson and R. Timofte, “NTIRE 2017 challenge on single image super-resolution: Dataset and study,” in *CVPRW*, 2017, pp. 126–135.

[44] J.-B. Huang, A. Singh, and N. Ahuja, “Single image super-resolution from transformed self-exemplars,” in *CVPR*, 2015, pp. 5197–5206.

[45] Y. Matsui, K. Ito, Y. Aramaki, A. Fujimoto, T. Ogawa, T. Yamasaki, and K. Aizawa, “Sketch-based manga retrieval using Manga109 dataset,” *Springer MTAP*, vol. 76, no. 20, pp. 21 811–21 838, 2017.

[46] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, “A database and evaluation methodology for optical flow,” *IJCV*, vol. 92, no. 1, pp. 1–31, 2011.

[47] C. Liu and D. Sun, “On bayesian adaptive video super resolution,” *IEEE TPAMI*, vol. 36, no. 2, pp. 346–360, 2013.

[48] A. Mercat, M. Viitanen, and J. Vanne, “UVG dataset: 50/120fps 4K sequences for video codec analysis and development,” in *ACMMSys*, 2020, pp. 297–302.

[49] K. Soomro, A. R. Zamir, and M. Shah, “UCF101: A dataset of 101 human actions classes from videos in the wild,” *arXiv preprint arXiv:1212.0402*, 2012.

[50] H. Takeda, P. Milanfar, M. Protter, and M. Elad, “Super-resolution without explicit subpixel motion estimation,” *IEEE TIP*, vol. 18, no. 9, pp. 1958–1975, 2009.

[51] P. Yi, Z. Wang, K. Jiang, J. Jiang, and J. Ma, “Progressive fusion video super-resolution network via exploiting non-local spatio-temporal correlations,” in *ICCV*, 2019, pp. 3106–3115.

[52] T. Xue, B. Chen, J. Wu, D. Wei, and W. T. Freeman, “Video enhancement with task-oriented flow,” *IJCV*, vol. 127, no. 8, pp. 1106–1125, 2019.

[53] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *CVPR*, 2016, pp. 2818–2826.

[54] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: Common objects in context,” in *ECCV*, 2014, pp. 740–755.

[55] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” in *ICCV*, 2017, pp. 2980–2988.

[56] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” in *ICCV*, 2017, pp. 2961–2969.

[57] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.

[58] D. Martin, C. Fowlkes, D. Tal, and J. Malik, “A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics,” in *ICCV*, vol. 2, 2001, pp. 416–423.

[59] H. Zhao, A. Torralba, L. Torresani, and Z. Yan, “HACS: Human action clips and segments dataset for recognition and temporal localization,” in *ICCV*, 2019, pp. 8668–8678.

[60] J. Carreira, E. Noland, C. Hillier, and A. Zisserman, “A short note on the Kinetics-700 human action dataset,” *arXiv preprint arXiv:1907.06987*, 2019.

[61] C.-Y. Wu, R. Girshick, K. He, C. Feichtenhofer, and P. Krähenbühl, “A multigrid method for efficiently training video models,” in *CVPR*, 2020, pp. 153–162.

[62] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” *CVPR*, 2017.

[63] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE TIP*, vol. 13, no. 4, pp. 600–612, 2004.

[64] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016, pp. 770–778.

[65] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks.” in *CVPR*, 2017, pp. 2261–2269.

[66] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *CVPR*, 2017, pp. 5987–5995.

[67] S. Zagoruyko and N. Komodakis, “Wide residual networks,” in *BMVC*, 2016, pp. 87.1–87.12.

[68] P. Domingos, “A few useful things to know about machine learning,” *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[69] A. Stergiou and R. Poppe, “Analyzing human-human interactions: A survey,” *CVIU*, vol. 188, p. 102799, 2019.

[70] H. Kataoka, T. Wakamiya, K. Hara, and Y. Satoh, “Would mega-scale datasets further enhance spatiotemporal 3D CNNs?” *arXiv preprint arXiv:2004.04968*, 2020.

[71] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, “A closer look at spatiotemporal convolutions for action recognition,” in *CVPR*, 2018, pp. 6450–6459.

[72] J. Carreira and A. Zisserman, “Quo vadis, action recognition? A new model and the Kinetics dataset,” in *CVPR*, 2017, pp. 4724–4733.

[73] D. Tran, H. Wang, L. Torresani, and M. Feiszli, “Video classification with channel-separated convolutional networks,” in *ICCV*, 2019, pp. 5552–5561.

[74] A. Stergiou and R. Poppe, “Learn to cycle: Time-consistent feature discovery for action recognition,” *PRL*, vol. 141, pp. 1–7, 2021.

[75] C. Feichtenhofer, H. Fan, J. Malik, and K. He, “SlowFast networks for video recognition,” in *ICCV*, 2019, pp. 6202–6211.

[76] A. Stergiou and R. Poppe, “Multi-temporal convolutions for human action recognition in videos,” in *IJCNN*, 2021.

[77] C. Dong, C. C. Loy, K. He, and X. Tang, “Learning a deep convolutional network for image super-resolution,” in *ECCV*, 2014, pp. 184–199.

[78] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, “Image super-resolution using very deep residual channel attention networks,” in *ECCV*, 2018, pp. 286–301.

[79] T. Dai, J. Cai, Y. Zhang, S.-T. Xia, and L. Zhang, “Second-order attention network for single image super-resolution,” in *CVPR*, 2019, pp. 11 065–11 074.

[80] B. Niu, W. Wen, W. Ren, X. Zhang, L. Yang, S. Wang, K. Zhang, X. Cao, and H. Shen, “Single image super-resolution via a holistic attention network,” in *ECCV*, 2020, pp. 191–207.

[81] M. Choi, H. Kim, B. Han, N. Xu, and K. M. Lee, “Channel attention is all you need for video frame interpolation,” in *AAAI*, vol. 34, no. 07, 2020, pp. 10 663–10 671.

[82] J. Park, K. Ko, C. Lee, and C.-S. Kim, “BMBC: Bilateral motion estimation with bilateral cost volume for video interpolation,” in *ECCV*, 2020, pp. 109–125.

[83] H. Sim, J. Oh, and M. Kim, “XVFI: extreme video frame interpolation,” in *ICCV*, 2021.

[84] T. Ding, L. Liang, Z. Zhu, and I. Zharkov, “CDFI: Compression-driven network design for frame interpolation,” in *CVPR*, 2021, pp. 8001–8011.

Alexandros Stergiou (Student Member, *IEEE*) received his Ph.D. degree in Computer Science from Utrecht University’s Department of Information and Computing Sciences (2021). He obtained his B.Sc. and M.Sc degrees in Computer Science from the University of Essex. He is currently a Research Associate at University of Bristol’s Department of Computer Science. His research interests include recognition and prediction of human actions from videos and deep learning model explainability.

Ranald Poppe (Senior Member, *IEEE*) received his Ph.D. in Computer Science from the University of Twente, the Netherlands (2009). He was a visiting researcher at the Delft University of Technology, Stanford University and University of Lancaster. He is currently an associate professor at the Department of Information and Computing Sciences of Utrecht University. His research interests include modeling of visual attention and the analysis of human (interactive) behavior from videos and other sensors.