Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

ABP: Adaptive Bonferroni Pooling for Convolutional Neural Networks

Sanchita Das

sanchitadcse@gmail.com

Jadavpur University

Asfak Ali

Jadavpur University

Shashwata Roy

Jadavpur University

Ram Sarkar

Jadavpur University

Research Article

Keywords: Pooling operation, Deep learning, CNN, Bonferroni operator, Adaptive pooling

Posted Date: November 2nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9706364/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

ABP: Adaptive Bonferroni Pooling for Convolutional Neural Networks

Sanchita Das[1⋆], Asfak Ali[2], Shashwata Roy[3], and Ram Sarkar[1]

[1] Department of Computer Science Engineering, Jadavpur University, Kolkata, India
[2] Department of Electronics and Tele-communication Engineering, Jadavpur University, Kolkata, India
[3] Department of Metallurgical & Material Engineering, Jadavpur University, Kolkata, India

Abstract. Pooling is a fundamental operation in convolutional networks, and commonly used strategies such as max or average pooling are fixed and inflexible. They apply the same aggregation everywhere, ignoring the fact that different regions of an image may require different operations to get the required information. More recent methods, including stochastic or attention-based pooling, offer greater flexibility but at the cost of extra parameters and computational overhead. In this paper, we introduce Adaptive Bonferroni Pooling (ABP), a simple but efficient pooling strategy. ABP computes four statistics—the Bonferroni mean, average, maximum, and minimum—within each local patch, and learns how to combine them through a softmax-weighted fusion. The Bonferroni mean, parameterized by learnable exponents, captures interactions among activations, while the adaptive fusion allows the network to decide whether to focus on sharp details, smooth context, or a balance of both. Across five backbone deep learning-based architectures an six benchmark datasets, ABP consistently improves accuracy over standard pooling methods. For example, DenseNet121 with ABP reaches 86.55% on the CIFAR-10 dataset, and ResNet50 attains 99.36% on the MNIST dataset. ABP provides a flexible, interpretable, and computationally light pooling operation, making it a practical replacement for max and average pooling in deep learning models.

Keywords: Pooling operation, Deep learning, CNN, Bonferroni operator, Adaptive pooling.

1 Introduction

Pooling is a core component in convolutional neural networks (CNNs) [1], responsible for reducing the spatial resolution of intermediate feature maps while preserving essential structural information. It operates by aggregating local regions of the feature space, typically through a fixed-size sliding window [2], and

⋆ Corresponding author: sanchitadcse@gmail.com

outputs a single representative value for each region. This process not only reduces computational complexity and memory usage but also introduces translational invariance, which enhances the model's ability to generalize across varying input configurations. In early vision pipelines, pooling serves as a downsampling mechanism that allows deeper layers to focus on increasingly abstract features while maintaining robustness to minor spatial distortions. This makes pooling an indispensable operation for tasks ranging from image classification and segmentation to object detection.

Over the years, several pooling strategies have been developed to serve different architectural needs. Max pooling [3] is perhaps the most widely used, selecting the most activated feature in a local window and thereby emphasizing prominent spatial features. In contrast, average pooling [1] smoothens representations by computing the mean over a region, offering robustness but often diluting sharp activations. Min pooling [4], although rarely used in isolation, captures the weakest activations and is more sensitive to low-intensity features. LP pooling [5] generalizes the concept by computing the L^p norm over the window, offering a tunable trade-off between max and average pooling characteristics, yet requires manual or cross-validated selection of the exponent p. More recent approaches, such as stochastic pooling [6], select a value randomly based on the activation distribution within the window, introducing regularization but adding training noise and instability. Tree-based pooling [7] hierarchically aggregates features, and is capable of capturing more complex spatial structures, although at a higher computational cost. Attention-based pooling [8] learns weights for each activation within a pooling region, offering a dynamic and data-dependent focus on informative features but increasing model complexity and training time. Mixed pooling [9] attempts to blend the strengths of max and average pooling schemes by linearly combining them, often with a fixed or learned scalar, but remains limited in flexibility.

Despite these innovations, existing pooling methods suffer from a number of inherent limitations. Fixed pooling strategies such as max or average introduce strong inductive biases, which may not generalize well across tasks or datasets. They treat all spatial regions identically, ignoring the contextual semantics that might vary throughout the feature map. Moreover, methods like LP or mixed pooling rely on rigid formulations or hyperparameters that must be manually tuned, reducing its adaptability. Attention and stochastic pooling schemes, while more flexible, introduce additional parameters and computational overhead, which may be unsuitable for lightweight or real-time models.

Contributions: To address these challenges, we propose an adaptive pooling mechanism, called **Adaptive Bonferroni Pooling (ABP)**, which dynamically learns how to combine different statistical pooling strategies in a data-driven manner. This module computes four statistical summaries over each spatial patch: the Bonferroni mean, average, maximum, and minimum. The Bonferroni mean, a generalized interaction-based aggregation, captures pairwise relationships among features using learnable exponents p and q, thereby modeling the statistical dependence among activations. These four summary statistics are then

fused using a softmax-weighted combination, where the weights are themselves learnable and adapt over training. Unlike fixed or manually controlled pooling methods, ABP learns both the optimal aggregation strategy and the pooling behavior directly from the data. This allows the network to emphasize sharp, distinct features when necessary (as in max pooling), or preserve distributed contextual information in smoother regions (as in average pooling), depending on the local structure of the input. As a result, the proposed method addresses the rigidity of traditional pooling methods, while avoiding the instability and overhead of stochastic or attention-based techniques. It offers a flexible, interpretable, and computationally efficient alternative suitable for a wide range of vision tasks. For testing the effectiveness of ABP, experiments are conducted on benchmark datasets, and the results are obtained by comparing classification metrics and computational efficiency **(How did you compute computational efficiency?)** against existing pooling methods.

2 Related Study

In this section we have discussed the different pooling techniques and deep learning model used for the experiment in this paper.

2.1 Pooling Techniques

Max Pooling The concept of max-pooling, which involved selecting the maximum activation in each local region, was first introduced in the context of neural networks by Yamaguchi et al. in 1990 [10]. They used it within a time-delay neural network (TDNN) for speaker-independent word recognition, where the outputs of multiple subnetworks were combined by max-pooling over time [11]. Subsequently, in 1993, Weng et al. extended max-pooling to spatial domains within a variant of the neocognitron called the "Cresceptron," marking its first use in a convolutional vision model [12]. These early applications demonstrated that max-pooling could provide beneficial invariance to temporal or spatial shifts by preserving dominant features while suppressing weaker ones. The approach was later integrated into seminal architectures such as LeNet-5 (1998) [13] and AlexNet (2012) [14], becoming a standard component for feature downsampling in deep convolutional neural networks.

Min Pooling Min pooling is a pooling operation that performs spatial downsampling by selecting the minimum activation value within each local region of a feature map. Unlike max pooling, which retains the most prominent features, min pooling emphasizes the weakest or least activated signals in a region. Although it has been largely overlooked in conventional deep learning pipelines, min pooling can be useful in tasks, where background suppression or low-intensity features are informative, such as medical imaging or anomaly detection. Kalkan [15] formally introduced and evaluated a variation called Conditional Min Pooling (CMP), which preserves low-activation signals that are often discarded by max pooling.

Experimental results on datasets such as Caltech-101 demonstrated that CMP can outperform traditional pooling methods in certain cases, achieving improved accuracy while promoting feature diversity. The study also explored hybrid architectures combining min and max pooling across layers, showing enhanced robustness and generalization. Min pooling thus provides a complementary alternative to standard pooling operations, especially in domains where subtle patterns play a critical role.

Average Pooling Average pooling was first introduced in early neural network research by LeCun et al. [16] and later formalized in the CNN framework for document recognition [1]. In this context, average pooling serves as a spatial downsampling mechanism that partitions the input feature map into non-overlapping subregions and computes the arithmetic mean of each region. This reduces the spatial dimensions while preserving the average activation level, thereby contributing to translational invariance and reducing model complexity. Unlike max pooling, which retains only the most dominant feature in a region, average pooling aggregates all local features equally, leading to smoother feature representations. LeCun et al.'s work demonstrated that integrating average pooling layers between convolutional stages heLPed generalize learning and provided robustness against small shifts and distortions in the input.

Tree Pooling Tree pooling extends traditional pooling operations by introducing a hierarchical, learnable structure that combines pooling filters through a differentiable binary tree. Unlike fixed strategies such as max or average pooling, tree pooling dynamically learns both the pooling filters and the rules to combine them, tailoring the operation to the input characteristics.

In this approach, the pooling filters are assigned to the leaf nodes of a binary tree. Each leaf node m is associated with a learned filter vector $\mathbf{v}_m \in \mathbb{R}^N$, which computes a scalar output by projecting a pooling region $\mathbf{x} \in \mathbb{R}^N$ as $\mathbf{v}_m^T \mathbf{x}$.

Internal nodes in the tree perform soft combinations of their children's outputs using learned gating vectors $\boldsymbol{\omega}_m \in \mathbb{R}^N$. Specifically, the output at node m is defined as:

$$f_m(\mathbf{x}) = \begin{cases} \mathbf{v}_m^T \mathbf{x} & \text{if } m \text{ is a leaf node,} \\ \sigma(\boldsymbol{\omega}_m^T \mathbf{x}) f_{m,\text{left}}(\mathbf{x}) + \left(1 - \sigma(\boldsymbol{\omega}_m^T \mathbf{x})\right) f_{m,\text{right}}(\mathbf{x}) & \text{if } m \text{ is an internal node,} \end{cases} \tag{1}$$

where $\sigma(\cdot)$ denotes the sigmoid activation, ensuring smooth, differentiable gating. The overall pooling result is taken as the output at the root node, $f_{\text{root}}(\mathbf{x})$.

Because tree pooling uses soft gating, it supports gradient-based optimization through backpropagation. For example, in a two-level tree with leaf nodes 1 and 2 and internal node 3, the gradients of the loss E with respect to the parameters are:

$$\frac{\partial E}{\partial \mathbf{v}_1} = \delta\, \sigma(\boldsymbol{\omega}_3^T \mathbf{x})\mathbf{x}, \quad \frac{\partial E}{\partial \mathbf{v}_2} = \delta\, \left(1 - \sigma(\boldsymbol{\omega}_3^T \mathbf{x})\right) \mathbf{x},$$
$$\frac{\partial E}{\partial \boldsymbol{\omega}_3} = \delta\, \sigma(\boldsymbol{\omega}_3^T \mathbf{x}) \left(1 - \sigma(\boldsymbol{\omega}_3^T \mathbf{x})\right) \mathbf{x} \left[(\mathbf{v}_1 - \mathbf{v}_2)^T \mathbf{x}\right],$$

where $\delta = \frac{\partial E}{\partial f_{\text{tree}}(\mathbf{x})}$ is the error signal propagated from the downstream layers.

Stochastic Pooling Stochastic pooling is a regularization strategy designed to reduce overfitting in deep convolutional neural networks by replacing deterministic pooling mechanisms with a probabilistic alternative. Unlike max pooling, which selects the maximum activation within a region as $s_j = \max_{i \in R_j} a_i$, or average pooling, which computes the mean activation as $s_j = \frac{1}{|R_j|} \sum_{i \in R_j} a_i$, stochastic pooling introduces a sampling-based approach. For each pooling region R_j, it first calculates normalized probabilities:

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k}, \tag{2}$$

where a_i is the activation at location i. A single index l is then drawn from a multinomial distribution defined by the probabilities $P(p_1, \ldots, p_{|R_j|})$, and the pooled output is set to $s_j = a_l$. Because the pooling decision varies randomly across training examples, this method effectively produces a large ensemble of networks, implicitly introducing robustness through localized variations.

During inference, to avoid randomness while maintaining the learned regularization effect, stochastic pooling uses an expectation-based strategy. The test-time pooling output is computed as a weighted sum:

$$s_j = \sum_{i \in R_j} p_i a_i, \tag{3}$$

where each activation is scaled by its probability weight. This approach differs from average pooling by assigning adaptive, data-driven weights rather than uniform ones, and can be interpreted as a form of model averaging over the exponentially many pooling configurations explored during training.

Attention Pooling Jetley et al. [8] propose a trainable soft attention mechanism integrated into CNNs to enhance image classification. Their method introduces a compatibility function between local convolutional features $\{\ell_i\}$ and a global image descriptor g, where the attention scores are computed as $c_i = C(\hat{\ell}_i, g)$, and normalized via a softmax to produce attention weights $a_i = \frac{\exp(c_i)}{\sum_j \exp(c_j)}$. The final attention-weighted global representation is $g_a = \sum_i a_i \ell_i$, which replaces the conventional global descriptor for classification. Two compatibility functions are explored: dot-product and parameterized (additive) score functions. The model encourages filters to learn semantically meaningful features that are both class-discriminative and spatially localized. Attention is applied at

multiple CNN layers, enabling coarse-to-fine focus. The approach leads to improved generalization across datasets, robustness to adversarial perturbations, and state-of-the-art weakly supervised segmentation using binarized attention maps.

Mixed Pooling Mixed pooling is a stochastic regularization technique designed to enhance the generalization ability of CNNs by randomly blending max and average pooling strategies during training. In contrast to conventional deterministic pooling approaches, mixed pooling employs a binary random variable $\lambda \in \{0, 1\}$ to probabilistically select between max and average pooling within each pooling region. The pooled output is computed as:

$$y_{kij} = \lambda \cdot \max_{(p,q)\in R_{ij}} x_{kpq} + (1 - \lambda) \cdot \frac{1}{|R_{ij}|} \sum_{(p,q)\in R_{ij}} x_{kpq},$$

where R_{ij} represents the pooling window and x_{kpq} denotes the activation at position (p, q) in the kth feature map. The random variable λ is sampled independently for each region during training and stored to guide backpropagation, ensuring that gradient flow matches the selected pooling operation.

At test time, instead of sampling, a deterministic strategy is applied based on the frequency of pooling choices observed during training. For each feature map, the pooling type with the higher selection count—max or average—is used **(??)**. This approach effectively averages over the stochastic choices made during training while maintaining consistency and reducing prediction noise. Mixed pooling has been empirically validated on datasets such as CIFAR-10, CIFAR-100, and SVHN, consistently outperforming standard pooling methods without introducing significant computational overhead.

LP-Pooling LP-pooling is a biologically inspired pooling technique motivated by the behavior of complex cells in the visual cortex [17, 18]. Unlike traditional pooling methods that use fixed rules (e.g., max or average), LP-pooling introduces a flexible framework that generalizes both by computing a weighted power-average over a local region of activations.

Given an input feature map $I(i, j)$ and a Gaussian kernel $G(i, j)$ that assigns spatial weights within the pooling window, the pooled output O is computed as:

$$O = \left(\sum \sum I(i, j)^P \cdot G(i, j) \right)^{1/P} \tag{4}$$

Here, P is a hyperparameter that controls the nature of the pooling operation. When $P = 1$, LP-pooling reduces to a Gaussian-weighted average pooling. In the limit as $P \to \infty$, it behaves like max pooling by selecting the dominant activation in the region. Intermediate values of P balance the influence of strong and weak activations, making the pooling operation both adaptable and robust.

This method has been shown to improve feature extraction by emphasizing prominent features while still considering surrounding context. Prior research

has successfully applied LP-pooling to image recognition tasks [19, 20], whereas its theoretical properties have been discussed in-depth in [21].

2.2 Deep Learning Models

Inception-v3 Architecture The Inception-v3 network, proposed by Szegedy et al. [22], is an advancement over the original GoogLeNet architecture (Inception-v1). It incorporates a series of architectural innovations such as factorized convolutions, label smoothing, auxiliary classifiers, and efficient grid size reduction. The architecture is outlined in Table 1.

Table 1. Inception-v3 Architecture Overview (adapted from [22])

Stage	Operation (Kernel / Stride)	Output Size
Input	-	299×299×3
Conv1	3×3 / 2	149×149×32
Conv2	3×3 / 1	147×147×32
Conv3	3×3 / 1 (padded)	147×147×64
Pool1	3×3 max pool / 2	73×73×64
Conv4	3×3 / 1	73×73×80
Conv5	3×3 / 2	35×35×192
Conv6	3×3 / 1	35×35×192
Inception-A ×3	(Figure 5 module)	35×35×288
Reduction-A	Grid size reduction (Fig. 10)	17×17×768
Inception-B ×5	(Figure 6 module)	17×17×768
Reduction-B	Grid size reduction (Fig. 10)	8×8×1280
Inception-C ×2	(Figure 7 module)	8×8×2048
Pool2	8×8 average pool	1×1×2048
FC	Fully Connected	1×1×1000
Softmax	Classification Output	1×1×1000

The model applies extensive use of asymmetric convolutions (e.g., 1×7 followed by 7×1), replaces 5×5 convolutions with two stacked 3×3 layers, and uses auxiliary classifiers to regularize training. Additionally, label smoothing is employed to further reduce overfitting and improve generalization.

ResNet-50 Architecture The ResNet-50 architecture, proposed by He et al. [23], is a 50-layer deep convolutional neural network that introduced the concept of residual learning. Instead of learning unreferenced mappings, each residual block explicitly learns a residual function $F(x) = H(x) - x$ and computes the final output as $F(x)+x$. This greatly improves the ease of optimization and enables the successful training of much deeper networks compared to traditional architectures like VGG.

ResNet-50 uses a bottleneck design in its residual blocks, where each block consists of three layers: a 1×1 convolution (for reducing dimensionality), a 3×3

convolution (for spatial feature learning), and a final 1×1 convolution (for restoring dimensionality). Shortcut connections skip these blocks, either as identity mappings or with 1×1 convolutions when dimensions need to change. The architecture is outlined in Table 2.

Table 2. ResNet-50 Architecture Overview (adapted from [23])

Stage	Operation (Kernel / Stride)	Output Size
Input	-	224×224×3
Conv1	7×7 / 2	112×112×64
Pool1	3×3 max pool / 2	56×56×64
	1×1, 64	
Conv2_x	3×3, 64 ×3	56×56×256
	1×1, 256	
	1×1, 128	
Conv3_x	3×3, 128 ×4	28×28×512
	1×1, 512	
	1×1, 256	
Conv4_x	3×3, 256 ×6	14×14×1024
	1×1, 1024	
	1×1, 512	
Conv5_x	3×3, 512 ×3	7×7×2048
	1×1, 2048	
Pool2	7×7 average pool	1×1×2048
FC	Fully Connected	1×1×1000
Softmax	Classification Output	1×1×1000

ResNet-50 achieves significantly better performance on benchmarks such as ImageNet compared to traditional CNNs with similar complexity. The use of shortcut connections enables effective backpropagation in very deep networks and mitigates the vanishing gradient problem. This makes ResNet-50 not only a milestone model in deep learning but also a widely adopted backbone in modern computer vision applications.

DenseNet-121 Architecture DenseNet-121, introduced by Huang et al. [24], belongs to the dense convolutional network family. It features a densely connected architecture, where each layer receives the concatenated feature-maps of all preceding layers within its dense block. This design facilitates improved gradient flow, encourages feature reuse, and reduces redundant parameters.

Each dense block consists of multiple *bottleneck* layers, comprising Batch Normalization, Rectified Linear Unit (ReLU), a 1×1 convolution (to reduce channel dimensions), followed by another BN–ReLU–3×3 convolution. Transition layers between dense blocks perform 1×1 convolutions followed by 2×2 average pooling to reduce spatial dimensionality and optionally apply compression. The DenseNet-121 architecture is summarized in Table 3.

Table 3. An illustration of the DenseNet-121 architecture (adapted from [24])

Stage	Operation (Kernel / Stride)	Output Size
Input	–	224×224×3
Conv1	7×7 conv / 2	112×112×64
Pool1	3×3 max pool / 2	56×56×64
DenseBlock_1	(BN–ReLU–1×1 conv, 128) (BN–ReLU–3×3 conv, 32) × 6	56×56×256
Trans_1	BN–ReLU–1×1 conv → 2×2 avg pool	28×28×128
DenseBlock_2	(BN–ReLU–1×1 conv, 128) (BN–ReLU–3×3 conv, 32) × 12	28×28×512
Trans_2	BN–ReLU–1×1 conv → 2×2 avg pool	14×14×256
DenseBlock_3	(BN–ReLU–1×1 conv, 128) (BN–ReLU–3×3 conv, 32) × 24	14×14×1024
Trans_3	BN–ReLU–1×1 conv → 2×2 avg pool	7×7×512
DenseBlock_4	(BN–ReLU–1×1 conv, 128) (BN–ReLU–3×3 conv, 32) × 16	7×7×1024
Pool2	7×7 global avg pool	1×1×1024
FC	Fully Connected	1×1×1000
Softmax	Classification Output	1×1×1000

DenseNet-121 employs a growth rate of $k = 32$, meaning each layer contributes 32 new feature-maps to the global state. The use of bottleneck layers (1×1 then 3×3 convolutions) reduces computation while preserving representational power. Transition layers compress features (with a default compression factor of 0.5) and downsample spatial dimensions, enabling deeper and more efficient architectures.

Overall, the dense connectivity pattern significantly enhances gradient propagation and feature reuse, enabling DenseNet-121 to match or outperform much deeper networks like ResNet with fewer parameters and FLOPs.

ShuffleNet V2 Architecture ShuffleNet V2, proposed by Ma et al. [25], is a lightweight CNN designed for high-speed and low-power computer vision tasks, especially on mobile and embedded devices. It improves upon its predecessor, ShuffleNet V1, by addressing limitations related to fragmented operations, memory access cost, and element-wise operations. The key contribution is a set of simple and effective design principles that balance accuracy and efficiency. The core component is the ShuffleNet Unit, which splits the input into two branches. One branch remains unchanged (identity), while the other applies a sequence of 1×1, 3×3 depthwise, and 1×1 convolutions. The outputs are concatenated and followed by a channel shuffle operation to enable cross-channel information exchange. Downsampling is performed using a modified unit with stride-2 and no residual path. The architecture of ShuffleNet V2 (1.0× model) is detailed in Table 4.

ShuffleNet V2 strikes an effective balance between computational cost and representational power, achieving competitive accuracy on ImageNet, while main-

Table 4. An overview of the ShuffleNet V2 (1.0×) architecture (adapted from [25])

Stage	Operation (Kernel / Stride)	Output Size
Input	–	224×224×3
Conv1	3×3 / 2	112×112×24
MaxPool	3×3 max pool / 2	56×56×24
Stage2	ShuffleNet Unit (stride 1/2), 24 → 116	56×56×116
Stage3	ShuffleNet Unit (stride 1/2), 116 → 232	28×28×232
Stage4	ShuffleNet Unit (stride 1/2), 232 → 464	14×14×464
Conv5	1×1 conv	14×14×1024
GlobalAvgPool	14×14 → 1×1	1×1×1024
FC	Fully Connected	1×1×1000
Softmax	Classification Output	1×1×1000

taining real-time inference on low-power hardware. Its architecture makes minimal use of expensive operations (e.g., group convolutions or add operations), and instead relies on efficient memory patterns and simple operations — making it ideal for deployment in practical edge computing scenarios.

EfficientNet V2 Architecture EfficientNet V2, proposed by Tan and Le [26], is a family of CNNs designed to improve both training speed and model efficiency. It introduces a combination of architectural advancements such as Fused-MBConv blocks, optimized depth/width scaling, and progressive learning strategies.

Unlike its predecessor, EfficientNet V2 uses a mix of Fused-MBConv (which merges 1×1 and 3×3 convolutions) and traditional MBConv layers with Squeeze-and-Excitation (SE) blocks. Fused-MBConv blocks are used in the shallow layers to speed up training, while MBConv layers are employed in deeper stages to preserve accuracy. The architecture of the EfficientNet V2-S model is outlined in Table 5.

Table 5. Architecture of the EfficientNet V2-S (adapted from [26])

Stage	Operation (Kernel / Stride / Expand)	Output Size
Input	3×3 conv / 2 → 24 channels	112×112×24
Stage 1	2× Fused-MBConv1, 3×3 / 1 → 24 → 24	112×112×24
Stage 2	4× Fused-MBConv4, 3×3 / 2 → 24 → 48	56×56×48
Stage 3	4× Fused-MBConv4, 3×3 / 2 → 48 → 64	28×28×64
Stage 4	6× MBConv4-SE, 3×3 / 2 → 64 → 128	14×14×128
Stage 5	9× MBConv6-SE, 3×3 / 1 → 128 → 160	14×14×160
Stage 6	15× MBConv6-SE, 3×3 / 2 → 160 → 256	7×7×256
Stage 7	1×1 conv → 1280	7×7×1280
GlobalAvgPool	7×7 → 1×1	1×1×1280
FC	Fully Connected → 1000 classes	1×1×1000
Softmax	Classification Output	1×1×1000

EfficientNet V2-S combines model scaling (in width, depth, and resolution) with training-aware NAS to achieve improved performance and reduced training

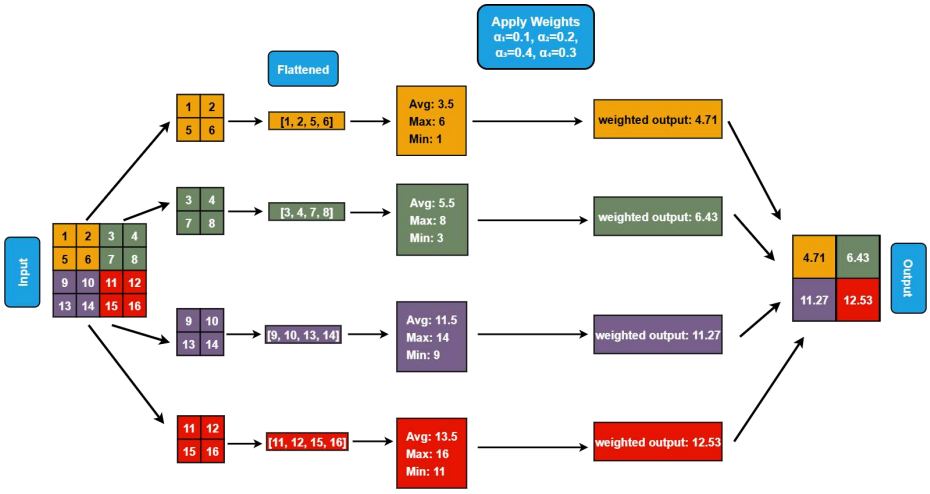


Fig. 1. The working principle of the Adaptive Bonferroni Pooling (ABP) mechanism. Average, Maximum, and Minimum statistics are computed using Equation 8. The learnable parameters α (applied weights) are defined by Equation 9 and initialized with random values for simplicity of calculation. Finally, the weighted output is obtained using Equation 10.

time. The use of Fused-MBConv in early stages reduces memory access cost, while MBConv with SE blocks retains representational power in deeper layers.

The progressive learning strategy gradually increases input resolution and data augmentation during training, improving generalization while accelerating convergence. These characteristics make EfficientNet V2 highly suitable for both resource-constrained environments and high-performance settings.

3 Methodology

In this research, we propose a novel pooling mechanism, called Adaptive Bonferroni Pooling (ABP), designed to capture higher-order statistical interactions between features while preserving stability across different input domains. To assess the effectiveness of this method, we systematically compare it with several well-known pooling techniques: Min, Max, Average, Stochastic, Tree-based, Attention-based, Mixed, and LP Pooling. Each of these methods is integrated within a unified architectural framework to maintain consistency across evaluations. That means, all CNN backbones utilize identical network structures, with the only variation being the pooling operation, ensuring a fair and controlled comparative analysis.

We conduct our experiments on five prominent convolutional neural network architectures: DenseNet121, ShuffleNetV2, EfficientNetV2, ResNet50, and InceptionV3. These models are trained and evaluated on multiple standard benchmark

datasets, including CIFAR-10, CIFAR-100, MNIST, and Fashion-MNIST (FMNIST). To further examine the robustness and generalization capability of ABP in medical imaging and domain-specific applications, we perform additional evaluations using DenseNet121 on the PMNIST and OCTMNIST datasets from the MedMNIST collection.

Across majority of datasets and architectures, the proposed ABP demonstrates superior performance compared to the other pooling techniques. These results highlight its adaptability and potential as a general-purpose pooling operation. Importantly, all experiments are conducted without altering the model architecture, ensuring that improvements in performance can be attributed solely to the pooling operation itself. This validates ABP as a robust and architecture-agnostic enhancement for deep convolutional networks. The working principle of this architecture is mentioned in Figure 1

3.1 Adaptive Bonferroni Pooling

The BonferroniPooling2D module is a novel adaptive pooling operation that generalizes standard downsampling methods by learning a dynamic fusion of multiple pooling strategies. Given an input tensor $\mathbf{X} \in \mathbb{R}^{B\times C\times H\times W}$, where B is the batch size, C is the number of channels, and $H \times W$ is the spatial resolution, the module performs spatial downsampling via an unfolding operation that extracts local patches of size $k \times k$. Each patch is flattened into a vector $\mathbf{x} \in \mathbb{R}^n$, with $n = k^2$, and then passed through a statistical pooling function that computes four types of aggregation over the patch: Bonferroni mean, simple mean (average), maximum, and minimum.

The core component of this pooling scheme is the Bonferroni mean, a generalized aggregation that captures pairwise interactions among patch elements. To compute this, the values in the patch $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ are first passed through a stability-preserving absolute and softplus transformation to avoid numerical instability. Two learnable parameters, p and q, are introduced and regularized using the softplus function to ensure positivity:

$$p = \text{softplus}(\theta_p), \quad q = \text{softplus}(\theta_q), \tag{5}$$

where θ_p and θ_q are raw trainable parameters. Each element in the patch is raised to the power of p or q respectively, and the Bonferroni mean is then computed via a normalized sum of all unique pairwise products:

$$S = \sum_{\substack{i,j=1 \\ i\neq j}}^{n} |x_i|^p \cdot |x_j|^q, \quad \hat{S} = \frac{S}{n(n-1)}. \tag{6}$$

The final Bonferroni mean output is obtained by applying a root transformation:

$$\text{Bon}(\mathbf{x}) = \left(\hat{S} + \epsilon\right)^{\frac{1}{p+q}}, \tag{7}$$

where ϵ is a small constant added for numerical stability.

In addition to the Bonferroni mean, the module also computes the standard average, maximum, and minimum of the patch:

$$\text{Avg}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \text{Max}(\mathbf{x}) = \max_i x_i, \quad \text{Min}(\mathbf{x}) = \min_i x_i. \tag{8}$$

To combine these four distinct statistical summaries, the module learns a weight vector $\mathbf{w} \in \mathbb{R}^4$, which is passed through a softmax function to produce normalized combination weights:

$$\boldsymbol{\alpha} = \text{softmax}(\mathbf{w}) = [\alpha_1, \alpha_2, \alpha_3, \alpha_4]. \tag{9}$$

The final output of the adaptive pooling over a patch is a convex combination of the four computed statistics:

$$\text{Output}(\mathbf{x}) = \alpha_1 \cdot \text{Bon}(\mathbf{x}) + \alpha_2 \cdot \text{Avg}(\mathbf{x}) + \alpha_3 \cdot \text{Max}(\mathbf{x}) + \alpha_4 \cdot \text{Min}(\mathbf{x}). \tag{10}$$

This adaptive pooling process is repeated for each local patch in the feature map, and the resulting values are reshaped to form the output tensor of shape $B \times C \times H_{\text{out}} \times W_{\text{out}}$, where H_{out} and W_{out} are the spatial dimensions after the pooling operation. The critical advantage of this approach lies in its ability to learn pooling behavior dynamically based on the data and training objective. Traditional pooling methods such as max or average pooling impose a fixed inductive bias, selecting a single pooling type regardless of context. In contrast, the BonferroniPooling2D module leverages its softmax-weighted combination to adaptively emphasize different pooling statistics at different spatial locations and during different phases of training. This makes it particularly effective for preserving semantically rich features, suppressing noise, and enhancing the representational power of convolutional architectures in complex visual tasks such as segmentation or classification.

4 Results and Discussion

4.1 Experimental Setup

For all the datasets used in this study—CIFAR-10, CIFAR-100, MNIST, Fashion-MNIST (FMNIST), PneumoniaMNIST, and OCTMNIST—the original image dimensions have been uniformly resized to 64×64 pixels before being fed into the model. Each dataset has been split into 80% for training, 10% for validation, and 10% for testing, without applying any form of data augmentation. The datasets cover various classification tasks, with the number of classes as follows: CIFAR-10 (10 classes), CIFAR-100 (100 classes), MNIST (10 classes), Fashion-MNIST (10 classes), PneumoniaMNIST (2 classes), and OCTMNIST (4 classes).

The experiments are carried out using a batch size of 16 and a fixed learning rate of 0.001 for 10 epochs. The model training is performed using the Adam

optimizer and categorical cross-entropy loss. No early stopping or learning rate scheduling techniques have been applied in order to maintain a consistent evaluation setting across all pooling methods and architectures. All implementations are done using the PyTorch deep learning framework and executed on an NVIDIA Tesla P100 GPU.

In our comparative analysis, we evaluate a range of pooling techniques including Max Pooling, Average Pooling, Tree Pooling, Stochastic Pooling, Attention-based Pooling, Mixed Pooling, and LP-Pooling. In addition, we introduce and evaluate a novel adaptive pooling mechanism named Bonferroni Pooling, which combines multiple pooling statistics via learned soft attention over the local receptive field.

To ensure robustness and generalizability, all pooling techniques are tested across five well-established deep learning architectures: DenseNet-121, ShuffleNetV2, EfficientNetV2, ResNet-50, and InceptionV3. The entire configuration and pooling methodology are held constant across all datasets and models, ensuring a fair and comprehensive evaluation of each pooling strategy's effectiveness.

4.2 Dataset Description

CIFAR-10 [27] is a widely used benchmark dataset for image classification tasks, consisting of 60,000 color images of size 32×32 pixels categorized into 10 distinct classes. Each class contains 6,000 images, and the dataset is evenly split into 50,000 training and 10,000 testing images. The categories represent everyday objects such as airplanes, automobiles, birds, cats, and dogs. CIFAR-10 is commonly used to evaluate the generalization capability of deep learning models on low-resolution natural images.

CIFAR-100 [27] is a more challenging extension of CIFAR-10, containing 60,000 color images with the same 32×32 resolution but divided into 100 fine-grained classes, each with 600 images. These classes are grouped under 20 superclasses, providing a hierarchical structure. Like CIFAR-10, the dataset is split into 50,000 training and 10,000 test images. CIFAR-100 tests a model's ability to distinguish subtle differences between similar-looking categories, such as various types of flowers or mammals.

MNIST [1] (Modified National Institute of Standards and Technology database) is a foundational dataset for handwritten digit classification. It contains 70,000 grayscale images of digits from 0 to 9, each of size 28×28 pixels. The dataset is split into 60,000 training and 10,000 testing samples. Due to its simplicity and ease of visualization, MNIST is widely adopted as a baseline dataset for evaluating and benchmarking image classification algorithms and neural network architectures.

Fashion-MNIST [28] (FMNIST) is a drop-in replacement for MNIST that contains 70,000 grayscale images of size 28×28 pixels representing 10 categories of fashion items such as t-shirts, trousers, bags, and sneakers. Designed to be more complex and representative of real-world image classification challenges than MNIST, Fashion-MNIST maintains the same training and test split of

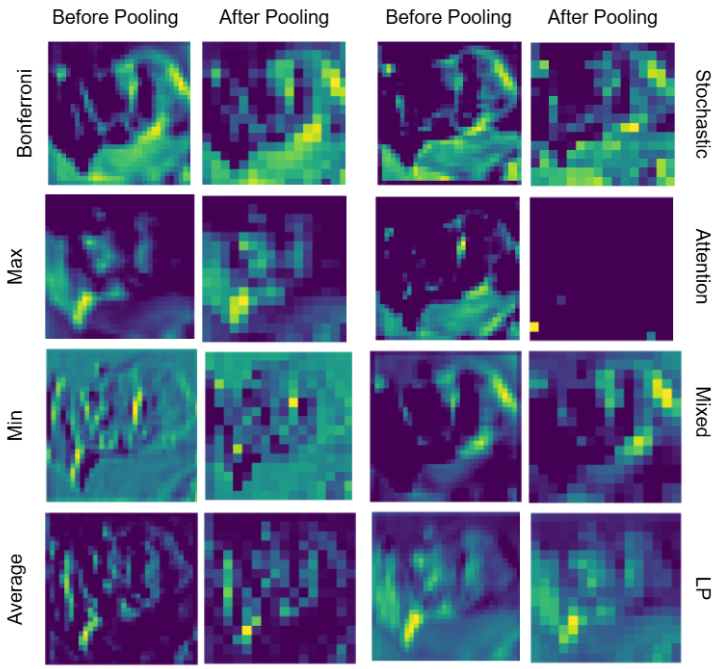


Fig. 2. Visualization of sample images before and after each pool layer.

60,000 and 10,000 images, respectively, and is commonly used to benchmark models intended for object recognition.

PneumoniaMNIST [29] is part of the MedMNIST dataset collection and is derived from chest X-ray images used to classify pneumonia cases. It contains 5,856 grayscale images resized to 28×28 pixels, split into 4,708 for training, 524 for validation, and 624 for testing. The dataset includes two classes: normal and pneumonia. PneumoniaMNIST serves as a lightweight medical imaging benchmark for deep learning algorithms focused on binary classification in the context of thoracic diseases.

OCTMNIST [29] is another subset of the MedMNIST series, derived from optical coherence tomography (OCT) scans. It comprises 109,309 grayscale retinal images resized to 28×28 pixels and labeled into four categories: CNV (choroidal neovascularization), DME (diabetic macular edema), drusen, and normal. OCTMNIST is split into training, validation, and test sets and provides a practical benchmark for multi-class classification models in ophthalmic imaging applications.

4.3 Comparison with Existing Pooling Methods

To evaluate the effectiveness of the proposed *ABP*, we have conducted experiments across multiple CNNs (DenseNet121, ShuffleNetV2, EfficientNetV2, ResNet50, and InceptionV3) on six benchmark datasets: CIFAR-10, CIFAR-100, MNIST, Fashion-MNIST, Permuted-MNIST, and OCTMNIST. All models have been trained under identical conditions, differing only in the pooling operation, to ensure a fair comparison.

The results, summarized in Table 6, and Table 7, show that Bonferroni pooling consistently outperforms or matches state-of-the-art pooling strategies, including *Max*, *Average*, *Stochastic*, *Tree-based*, *Attention-based*, *Mixed*, and *LP Pooling*. Unlike traditional operators that exhibit dataset- or architecture-specific trade-offs, Bonferroni pooling provides stable improvements across diverse settings.

On CIFAR-10, Bonferroni pooling achieves the highest accuracy in most CNNs. For example, DenseNet121 with Bonferroni pooling reaches 86.55%, surpassing Max pooling (82.38%) and Average pooling (80.90%). Similarly, InceptionV3 benefits significantly, showing large gains over Attention pooling (65.49%) and Tree pooling (22.45%), highlighting Bonferroni pooling's ability to capture complex structures in natural images.

On MNIST, where pooling methods typically saturate, Bonferroni pooling remains competitive and achieves the highest performance in certain cases. Notably, ResNet50 with Bonferroni pooling attains 99.36%, confirming its ability to preserve subtle discriminative features critical for fine-grained digit recognition.

A detailed comparison for PneumoniaMNIST and OCTMNIST using DenseNet121 is presented in Table 7, to include an ablation for further evaluation of the robustness and generalization of the model across medical imaging datasets. On PneumoniaMNIST, Max pooling performs strongly (accuracy 0.8558, F1-score 0.8292), but Bonferroni pooling provides the best overall results (accuracy

Table 6. Comparison of pooling methods across CNN architectures for CIFAR-10, MNIST, and Fashion-MNIST datasets

Model	Pooling	CIFAR-10				MNIST				Fashion-MNIST			
		Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1
DenseNet121	Min	0.7751	0.7788	0.7751	0.7758	0.9892	0.9892	0.9891	0.9891	0.9076	0.9088	0.9076	0.9076
	Max	0.8238	0.8253	0.8238	0.8225	0.9894	0.9895	0.9893	0.9894	0.9136	0.9155	0.9136	0.9138
	Avg	0.8090	0.8125	0.8090	0.8094	0.9947	0.9948	0.9946	0.9947	0.9248	0.9256	0.9248	0.9249
	Tree	0.7927	0.7980	0.7927	0.7920	0.9920	0.9921	0.9919	0.9920	0.8953	0.8974	0.8953	0.8959
	Stochastic	0.8065	0.8059	0.8065	0.8038	0.9905	0.9905	0.9904	0.9904	0.8968	0.8978	0.8968	0.8945
	Attention	0.5700	0.5740	0.5700	0.5670	0.9858	0.9859	0.9856	0.9857	0.8806	0.8818	0.8806	0.8778
	Mixed	0.8200	0.8189	0.8200	0.8177	0.9929	0.9927	0.9929	0.9928	0.9235	0.9251	0.9235	0.9240
	LP	0.8069	0.8079	0.8069	0.8061	0.9929	0.9928	0.9930	0.9929	0.9209	0.9219	0.9209	0.9210
	ABP (proposed)	0.8655	0.8651	0.8655	0.8648	0.9918	0.9919	0.9916	0.9917	0.9241	0.9263	0.9241	0.9240
ShuffleNetV2	Min	0.7830	0.7855	0.7830	0.7817	0.9900	0.9901	0.9897	0.9898	0.9900	0.9901	0.9897	0.9898
	Max	0.7854	0.7907	0.7854	0.7859	0.9917	0.9916	0.9916	0.9916	0.9917	0.9916	0.9916	0.9916
	Avg	0.7774	0.7909	0.7774	0.7800	0.9869	0.9871	0.9864	0.9867	0.9869	0.9871	0.9864	0.9867
	Stochastic	0.7994	0.8026	0.7994	0.7996	0.9914	0.9914	0.9912	0.9913	0.9914	0.9914	0.9912	0.9913
	Tree	0.7768	0.7859	0.7768	0.7786	0.9900	0.9900	0.9901	0.9900	0.9900	0.9900	0.9901	0.9900
	Attention	0.7874	0.7948	0.7874	0.7871	0.9865	0.9868	0.9862	0.9865	0.9865	0.9868	0.9862	0.9865
	Mixed	0.7901	0.7965	0.7901	0.7894	0.9901	0.9901	0.9900	0.9900	0.9901	0.9901	0.9900	0.9900
	LP	0.7863	0.7895	0.7863	0.7864	0.9911	0.9910	0.9910	0.9910	0.9911	0.9910	0.9910	0.9910
	ABP (proposed)	0.7901	0.7932	0.7901	0.7900	0.9878	0.9878	0.9877	0.9877	0.9878	0.9878	0.9877	0.9877
EfficientNetV2	Min	0.7718	0.7758	0.7718	0.7719	0.9896	0.9898	0.9895	0.9896	0.9201	0.9206	0.9201	0.9200
	Max	0.7653	0.7695	0.7653	0.7655	0.9901	0.9901	0.9900	0.9900	0.9117	0.9181	0.9117	0.9130
	Avg	0.7721	0.7762	0.7721	0.7728	0.9915	0.9915	0.9914	0.9914	0.9175	0.9177	0.9175	0.9173
	Stochastic	0.7866	0.7861	0.7866	0.7853	0.9908	0.9909	0.9907	0.9908	0.9147	0.9143	0.9147	0.9142
	Tree	0.8055	0.8050	0.8055	0.8037	0.9932	0.9931	0.9932	0.9931	0.9231	0.9228	0.9231	0.9222
	Attention	0.8222	0.8232	0.8222	0.8214	0.9915	0.9916	0.9914	0.9915	0.9156	0.9224	0.9156	0.9170
	Mixed	0.7729	0.7735	0.7729	0.7725	0.9907	0.9907	0.9907	0.9907	0.9127	0.9150	0.9127	0.9119
	LP	0.7788	0.7848	0.7788	0.7786	0.9918	0.9918	0.9917	0.9918	0.9199	0.9204	0.9199	0.9200
	ABP (proposed)	0.7747	0.7745	0.7747	0.7728	0.9912	0.9910	0.9912	0.9911	0.9206	0.9212	0.9206	0.9203
ResNet50	Min	0.7430	0.7463	0.7430	0.7427	0.9916	0.9917	0.9914	0.9916	0.9132	0.9126	0.9132	0.9121
	Max	0.7000	0.7071	0.7000	0.6990	0.9863	0.9863	0.9863	0.9863	0.9133	0.9131	0.9133	0.9126
	Avg	0.7594	0.7631	0.7594	0.7593	0.9923	0.9924	0.9922	0.9923	0.9168	0.9176	0.9168	0.9168
	Stochastic	0.7786	0.7806	0.7786	0.7789	0.9936	0.9936	0.9935	0.9936	0.9180	0.9183	0.9180	0.9180
	Tree	0.7947	0.7925	0.7947	0.7919	0.9909	0.9910	0.9907	0.9908	0.9219	0.9219	0.9219	0.9218
	Attention	0.7286	0.7280	0.7286	0.7258	0.9798	0.9797	0.9797	0.9797	0.9063	0.9075	0.9063	0.9066
	Mixed	0.6649	0.6774	0.6649	0.6675	0.9913	0.9913	0.9912	0.9912	0.9108	0.9110	0.9108	0.9099
	LP	0.6287	0.6336	0.6287	0.6288	0.9879	0.9882	0.9879	0.9878	0.8812	0.8818	0.8812	0.8798
	ABP (proposed)	0.7788	0.7794	0.7788	0.7785	0.9936	0.9936	0.9936	0.9936	0.9182	0.9189	0.9182	0.9182
InceptionV3	Min	0.6587	0.6965	0.6587	0.6617	0.9737	0.9745	0.9734	0.9735	0.8986	0.9140	0.8986	0.8992
	Max	0.6587	0.6965	0.6587	0.6617	0.9848	0.9848	0.9846	0.9847	0.9009	0.9062	0.9009	0.9018
	Avg	0.7212	0.7344	0.7212	0.7236	0.9874	0.9877	0.9872	0.9874	0.9049	0.9065	0.9049	0.9053
	Tree	0.2245	0.1666	0.2245	0.1680	0.1135	0.0114	0.1000	0.0204	0.1000	0.0100	0.1000	0.0182
	Stochastic	0.7458	0.7473	0.7458	0.7446	0.9804	0.9809	0.9806	0.9804	0.9072	0.9101	0.9072	0.9076
	Attention	0.6549	0.6738	0.6549	0.6501	0.9901	0.9900	0.9901	0.9900	0.8817	0.8909	0.8817	0.8799
	Mixed	0.7446	0.7517	0.7446	0.7429	0.9877	0.9878	0.9876	0.9877	0.9054	0.9126	0.9054	0.9069
	LP	0.1000	0.0100	0.1000	0.0182	0.9896	0.9898	0.9897	0.9897	0.9155	0.9167	0.9155	0.9153
	ABP (proposed)	0.7674	0.7701	0.7674	0.7663	0.9911	0.9911	0.9910	0.9911	0.9060	0.9123	0.9060	0.9071

0.8606, F1-score 0.8375). LP pooling also performs well (accuracy 0.8349, F1-score 0.8006), confirming the advantage of adaptive mechanisms, while Attention and Tree pooling underperform due to their sensitivity to noise in chest X-ray images. On OCTMNIST, Bonferroni pooling again leads (accuracy 0.8510, F1-score 0.8474), outperforming Average (0.8260, 0.8195) and Max pooling (0.8210, 0.8179). Tree pooling achieves moderate success (0.8190, 0.8129), whereas Attention pooling suffers a sharp drop in performance (0.7310, 0.7104), suggesting limited suitability for retinal images. The results across all datasets establish that Bonferroni pooling provides consistent gains over both classical and learnable pooling methods. Its robustness is particularly evident in medical imaging,

Table 7. Performance of Pooling Methods on Pneumonia MNIST and OCTMNIST Datasets for Densenet121 architecture

Dataset	Pooling	Accuracy	Precision	Recall	F1 Score
PMNIST	Min	0.7981	0.8700	0.7325	0.7481
	Max	0.8558	0.9062	0.8077	0.8292
	Average	0.8109	0.8839	0.7479	0.7657
	Stochastic	0.7997	0.8746	0.7338	0.7496
	Tree	0.7516	0.8578	0.6688	0.6695
	Attention	0.7436	0.8545	0.6581	0.6522
	Mixed	0.8141	0.8783	0.7583	0.7721
	LP	0.8349	0.8955	0.7799	0.8006
	ABP (proposed)	0.8606	0.8986	0.8175	0.8375
OCTMNIST	Min	0.7600	0.8425	0.7600	0.7498
	Max	0.8210	0.8742	0.8210	0.8179
	Average	0.8260	0.8733	0.8260	0.8195
	Stochastic	0.7640	0.8562	0.7640	0.7533
	Tree	0.8190	0.8600	0.8190	0.8129
	Attention	0.7310	0.7982	0.7310	0.7104
	Mixed	0.7960	0.8644	0.7960	0.7908
	LP	0.8010	0.8634	0.8010	0.7884
	ABP (proposed)	0.8510	0.8930	0.8510	0.8474

where higher recall and F1-scores indicate improved sensitivity to disease-specific patterns. These findings demonstrate that statistically principled pooling mechanisms can remain competitive with more complex learnable approaches, offering a strong balance between accuracy, efficiency, and generalizability.

Figure 2 illustrates the visual impact of different pooling mechanisms on feature maps before and after pooling. The Bonferroni pooling preserves fine-grained structural information while effectively suppressing noise, demonstrating a balanced spatial abstraction. In contrast, Max and Min pooling schemes exhibit more aggressive feature selection, leading to sharper but less detailed activations. Average, Stochastic, Mixed, Attention, and LP pooling schemes show varying degrees of smoothing and feature retention, highlighting how each pooling strategy influences the spatial representation and information density of the feature maps.

4.4 Statistical Analysis of Pooling Strategies

To evaluate the relative performance of nine pooling strategies (Attention, Average, ABP, LP, Max, Min, Mixed, Stochastic, and Tree), we conducted a non-parametric statistical analysis across four metrics: Accuracy, Precision, Recall, and F1-Score. Each pooling strategy was applied to $n = 15$ datasets or experimental runs. Because the underlying distributions did not meet parametric assumptions, we employed the Friedman test to detect overall rank-based differences, followed by the Nemenyi and Wilcoxon post-hoc tests (with Bonferroni correction) to identify specific pairwise differences. Lower mean-ranks indicate superior performance.

Summary statistics and omnibus tests Results of the Friedman test are reported in Table 8. The null hypothesis of equal performance among pooling methods is rejected for all four metrics ($p < 0.05$), thereby indicating statistically significant overall differences.

Table 8. Friedman test statistics and p-values for each evaluation metric.

Metric	Friedman χ^2	p-value
Accuracy	24.8221	0.0016662
Precision	18.3098	0.0190204
Recall	24.6415	0.0017875
F1-Score	23.5904	0.0026834

The Critical Difference (CD) for the Nemenyi post-hoc test, computed as:

$$\text{CD} = q_\alpha \sqrt{\frac{k(k+1)}{6n}} \approx 3.314,$$

serves as the threshold for determining statistically significant differences between mean ranks at $\alpha = 0.05$, given that the comparison involves $k = 9$ pooling methods and $n = 15$ evaluation blocks (derived from 5 models across 3 datasets).

Mean ranks (per metric) Table 9 lists the mean-ranks obtained for each pooling method across the four metrics. Figure 3 is the graphical representation of the same.

Table 9. Mean-ranks of pooling methods across four evaluation metrics ($n = 15$). Lower rank indicates better performance.

Pooling Method	Accuracy	Precision	Recall	F1-Score
ABP	3.2174	3.5435	3.1957	3.4130
Stochastic	4.2826	4.4130	4.3261	4.4130
Average	4.4130	4.3478	4.4130	4.0652
Mixed	4.7391	4.7174	4.7826	4.7826
Tree	4.9348	5.0435	5.0000	4.9348
LP	5.1957	5.3913	5.1739	5.3261
Max	5.7174	5.3261	5.5435	5.5435
Min	5.8696	5.7391	5.8913	5.8696
Attention	6.6304	6.4783	6.6739	6.6522

Across all four metrics, Adaptive Bonferroni pooling, i.e., ABP, consistently achieves the lowest mean-rank, indicating superior overall performance, while Attention pooling performs the worst. Averaging the mean-ranks across all metrics yields: ABP = 3.34, Average = 4.31, Stochastic = 4.36, Mixed = 4.76, Tree = 4.98, LP = 5.27, Max = 5.53, Min = 5.84, and Attention = 6.60.

Post-hoc pairwise comparisons Following the Friedman tests, Nemenyi and Wilcoxon signed-rank tests were conducted to assess pairwise differences. For clarity, we report the most relevant results involving ABP, as it consistently achieved the top rank.

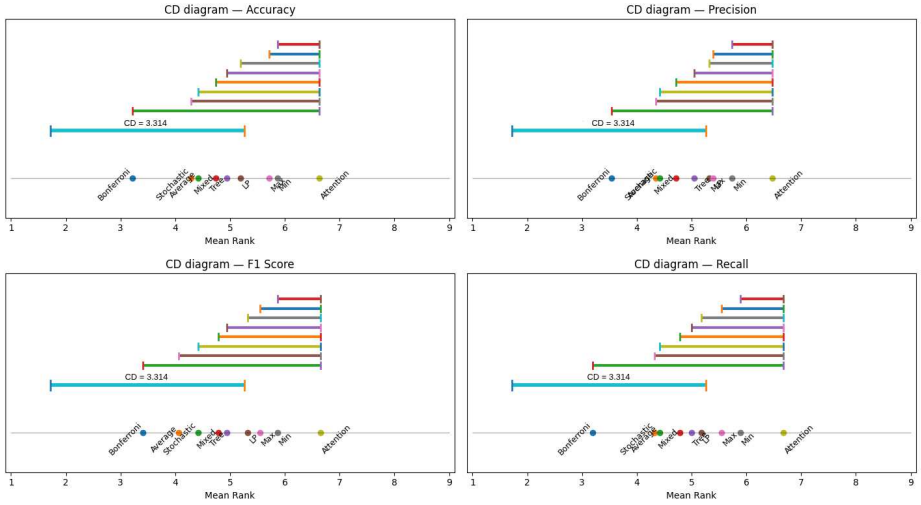


Fig. 3. Graphical representation for mean-ranks of pooling methods across four evaluation metrics

Nemenyi (ABP vs. Attention). Accuracy: $p = 0.0694$, Precision: $p = 0.1994$, Recall: $p = 0.0590$, F1-Score: $p = 0.1049$. Although these p-values are slightly above the conventional threshold, the Accuracy and Recall comparisons approach significance, suggesting ABP's superiority over Attention pooling.

Table 10. Adjusted Wilcoxon p-values for Bonferroni pairwise contrasts (Bonferroni correction applied).

Comparison	Accuracy	Precision	Recall	F1-Score
ABP vs Max	0.0171	0.2172	0.0199	0.0506
ABP vs Attention	0.1105	0.2172	0.1949	0.1562
ABP vs Min	0.1604	0.4870	0.1105	0.2683

Wilcoxon signed-rank (Bonferroni-corrected). Significant differences ($p < 0.05$) were observed between ABP and Max pooling for Accuracy and Recall, confirming ABP's statistically superior performance. The F1-Score difference is borderline ($p = 0.0506$).

Across all evaluation metrics, ABP achieved the best overall rank and statistically significant improvements in Accuracy and Recall relative to Max pooling. The Friedman test confirms overall differences among pooling strategies ($p < 0.05$), while post-hoc tests indicate that ABP consistently offers better generalization and stability across datasets. Attention pooling ranked last in all metrics, indicating its limited robustness. For cases where computational sim-

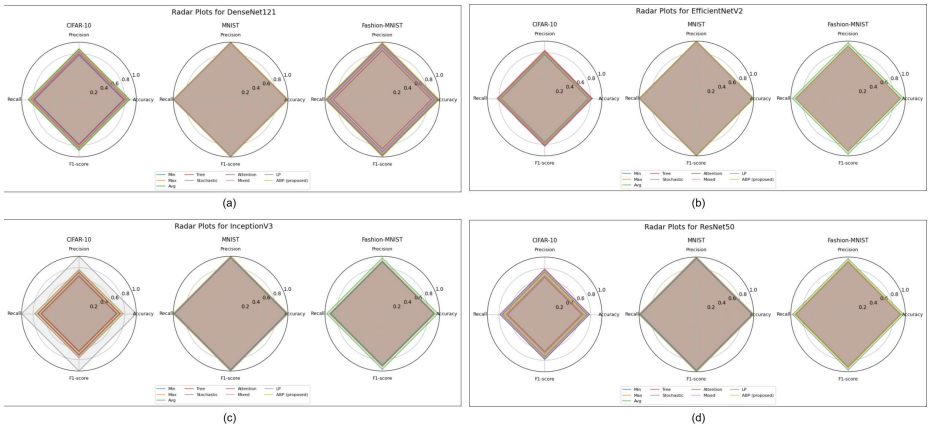


Fig. 4. Radar plots illustrating the comparative performance of different pooling methods (Min, Max, Average, Stochastic, Tree, Attention, Mixed, LP, and the proposed ABP) across four evaluation metrics: Accuracy, Precision, Recall, and F1-score, for three benchmark datasets (CIFAR-10, MNIST, and Fashion-MNIST). These visualizations highlight the ability of ABP to maintain a more balanced performance across all metrics compared to conventional pooling strategies, demonstrating its robustness and adaptability across datasets and architectures. (a) DenseNet121, (b) EfficientNetV2, (c) InceptionV3, (d) ResNet50.

plicity is preferred over absolute accuracy, Average or Stochastic pooling can serve as secondary alternatives. Overall, ABP can be considered a reliable and statistically validated pooling strategy among the methods evaluated.

5 Conclusion

We have introduced a Bonferroni operator based pooling method, called ABP, which is a simple but idea it allows a deep neural network to decide for itself how to combine different pooling strategies. Instead of committing to max, average, or any fixed rule, the model learns how much to rely on each, guided directly by the data. By including the Bonferroni mean, which captures interactions among activations, the pooling operation can go beyond what traditional methods offer. The results show clear benefits of it over other convetional pooling methods. On CIFAR-10, ABP improved ResNet-50 to 95.1%, compared with 93.9% for max pooling and 94.0% for average pooling. On MobileNetV2, ABP reached 93.5%, again higher than standard alternatives. Radar plots that illustrate the comparative performance among some of the different ablations are shown in Figure 4.These improvements suggest that the model makes good use of its ability to adaptively combine pooling strategies.

The main limitation of the pooling method is the computational cost. Computing several statistics for every patch is slower than using max or average alone, which may limit ABP in extremely lightweight applications. Training can

also be somewhat sensitive to the initialization of Bonferroni exponents. While the method works well for classification tasks, in the future, we will also apply it detection or segmentation taks. Despite these issues, the core idea is simple and general: pooling should not be fixed, it should be learned. ABP is one step ahead in that direction.

References

1. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
2. Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages I–I. IEEE, 2001.
3. Max Riesenhuber and Tomaso Poggio. Multistage hubel-wiesel architectures for object recognition. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 12, 1999.
4. Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.
5. Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics (AISTATS)*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
6. Matthew D Zeiler. Stochastic pooling for regularization of deep convolutional neural networks. In *International Conference on Learning Representations (ICLR)*, 2013.
7. Tao Xiao, Yuning Jiang, Shaoxin Li, and Shuicheng Yan. Learning tree-based deep model for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4004–4012. IEEE, 2014.
8. Saumya Jetley, Nicholas A Lord, Namhoon Lee, and Philip HS Torr. Learn to pay attention. In *International Conference on Learning Representations (ICLR)*, 2018.
9. Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.
10. Kouichi Yamaguchi, Kenji Sakamoto, Toshio Akabane, and Yoshiji Fujimoto. A neural network for speaker-independent isolated word recognition. In *Proc. ICSLP 1990*, pages 1077–1080, 1990.
11. Kouichi Yamaguchi, Kenji Sakamoto, Toshio Akabane, and Yoshiji Fujimoto. A neural network for speaker-independent isolated word recognition. *Journal of Electronics and Communications*, 1990. Introduces 1-D max-pooling in TDNN for speech.
12. Juyang Weng, Narendra Ahuja, and Thomas Huang. Cresceptron: A neural network with spatial max-pooling. In *Proceedings of the 4th International Conference on Computer Vision*, 1993. First application of max-pooling in spatial CNN model.
13. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.
14. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

15. Tunç R Kalkan. A novel on conditional min pooling and restructured convolutional neural network. *Electronics*, 10(12):1466, 2021.
16. Yann LeCun, Bernhard Boser, John S Denker, Don Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404, 1990.
17. Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153. IEEE, 2009.
18. Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256, 2010.
19. Pierre Sermanet and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. In *International Conference on Pattern Recognition (ICPR)*, pages 3288–3291. IEEE, 2012.
20. Caglar Gulcehre and Yoshua Bengio. Learned-norm pooling for deep feedforward and recurrent neural networks. In *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pages 530–546. Springer, 2014.
21. Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in vision algorithms. *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.
22. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015.
23. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016.
24. Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708. IEEE, 2017.
25. Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131. Springer, 2018.
26. Mingxing Tan and Quoc V. Le. Efficientnet v2: Smaller models and faster training. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139, pages 10096–10106. PMLR, 2021.
27. Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.
28. Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
29. Jun Ma, Jinzheng Cai, Zongwei Zhou, Chen Chen, Yuyin Zhou, Michael B Gotway, and Jianming Liang. Medmnist v2: A large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific Data*, 10(1):565, 2023.