An Adaptive Orthogonal Convolution Scheme for Efficient and Flexible CNN Architectures

Thibaut Boissin, Franck Mamalet, Thomas Fel, Agustin Martin Picard, Thomas Massena, Mathieu Serrurier

An Adaptive Orthogonal Convolution Scheme for Efficient and Flexible CNN Architectures

Thibaut Boissin[1,2,3] Franck Mamalet[1] Thomas Fel[5] Agustin Martin Picard[1] Thomas Massena[3,4]
Mathieu Serrurier[2,3]

[1] Institut de Recherche Technologique Saint-Exupery, France
[2]Artificial and Natural Intelligence Toulouse Institute
[3]IRIT [4] Innovation & Research Division, SNCF
[5] Kempner Institute, Harvard University

Abstract

Orthogonal convolutional layers are the workhorse of multiple areas in machine learning, such as adversarial robustness, normalizing flows, GANs, and Lipschitz-constrained models. Their ability to preserve norms and ensure stable gradient propagation makes them valuable for a large range of problems. Despite their promise, the deployment of orthogonal convolution in large-scale applications is a significant challenge due to computational overhead and limited support for modern features like strides, dilations, group convolutions, and transposed convolutions.

*In this paper, we introduce **AOC** (Adaptative Orthogonal Convolution), a scalable method for constructing orthogonal convolutions, effectively overcoming these limitations. This advancement unlocks the construction of architectures that were previously considered impractical. We demonstrate through our experiments that our method produces expressive models that become increasingly efficient as they scale. To foster further advancement, we provide an open-source library implementing this method, available at* https://github.com/thib-s/orthogonium.

1. Introduction and Related Works

Orthogonal layers have become fundamental components in various deep learning architectures due to their unique mathematical properties, which offer benefits across multiple applications. For instance, robustness against adversarial attacks can be achieved by managing a model's Lipschitz constant [54] – with 1-Lipschitz networks being a prime candidate [2] – an approach that requires the use of orthogonal layers. Initially, researchers experimented with regularization techniques [12]; however, constrained networks, especially those employing orthogonal layers, soon became central, as they provided the advantage of tighter certification bounds. Beyond robustness, orthogonal layers also play a key role in enhancing performance in normalizing flows. Normalizing flows are generative models that transform simple distributions into complex ones via invertible mappings [19, 42]. Orthogonal convolutions enable these transformations with a computable Jacobian determinant, thus improving training efficiency [29] and forming the basis for invertible residual networks [6] Additionally, orthogonal layers stabilize training deep and recurrent neural networks (RNNs) by preserving gradient norms through time, essential in capturing long-term dependencies in time-series, such as language and speech tasks [5, 27, 40]. Lastly, in Wasserstein GANs (WGANs) [4] orthogonality in both the discriminator and generator [35, 36] supports stability and expressivity without requiring weight clipping or gradient penalties [23], making it essential for large-scale GAN training [11].

However, despite these benefits, extending orthogonality to convolutional layers remains challenging. The orthogonalization of large Toeplitz matrices—structures central to convolution—presents difficulties without compromising convolutional properties. Efficient orthogonalization of these structured matrices has theoretical importance, affecting generalization [7] and indicating when orthogonal convolution is feasible [1]. Early approaches [40, 60] explored regularization, yet practical constraints led to the following solutions:

Explicit Construction Methods. Building on [64], approaches like BCOP [31], SC-Fac [53], and ECO [68] construct orthogonal convolutions directly in the spatial domain. These methods maintain orthogonality but often lack flexibility in kernel size control and do not support operations like striding and transposed convolutions.

Frequency Domain Approaches. Methods such as Cayley Convolution [55], LOT [66], and ProjUNN-T [27] enforce orthogonality by parameterizing kernels in Fourier space,

Method	Orthogonal	Equivalent Kernel Size	Code available	Change Channels	Stride	Conv Transpose	Groups	Dilation
BCOP [51]	✓	k	✓	✓	≈	✗	✗	✗
SC-FAC [53]	✓	k separable	✗	✓	✓	≈	✓	✓
ECO [68]	✓	$I_w \times I_h$	✗	≈	≈	✗	✗	≈
Cayley [55]	✓	$I_w \times I_h$	✓	≈	≈	≈	✗	✗
LOT [66]	✓	$I_w \times I_h$	✓	✓	≈	✗	✗	✗
ProjUNN-T [27]	✓	$I_w \times I_h$	✓	✗	✗	✗	✗	✗
SLL [3]	✗	composed	✓	✗	✗	✗	✗	✗
Sandwich [61]	✗	composed	✓	≈	≈	✗	✗	✗
AOL [38]	✗	k	✓	✓	✓	✗	✗	✗
SOC [51]	✓	$k + (n\frac{k}{2})$	✓	≈	≈	✗	✗	✗
AOC (Ours)	✓	k	✓	✓	✓	✓	✓	✓

Table 1. **Comparison of orthogonal convolution methods.** A check mark (✓) indicates full support for the feature, a cross mark (✗) indicates lack of support (in the implementation), and an approximate symbol (≈) indicates partial support (emulation). Here, k denotes the kernel size, and $I_w \times I_h$ represents the input dimensions. AOC is the first method to support strides, dilations, group convolution and transposed convolution.

albeit at the cost of increased computational complexity and constraints on spatial or grouped convolutions.

Composite Layer Techniques. Skew Orthogonal Convolutions [51] and Dynamical Isometry methods [3, 34] combine multiple convolutional layers to approximate orthogonality, increasing flexibility but adding layers and parameters, which can heighten model complexity.

Relaxed orthogonality approaches. In some cases, strict orthogonality is relaxed to mitigate vanishing gradients, avoiding the computational demands of full orthogonalization [3, 34, 38].

Finally, some studies question the overall impact of orthogonality on performance; for example, [39] noted inconsistencies when standardizing conditions, and [26] showed that architecture and dataset size could outweigh the benefits of orthogonal layers.

Our Contributions. In response to the limitations of existing methods, we introduce **A**daptive **O**rthogonal **C**onvolution (**AOC**), a novel approach for constructing convolution layers that adress key constraints in orthogonalization while remaining efficient:

- **Orthogonal**: AOC enforces strict orthogonality, allowing convolutional layers to retain essential properties across applications.
- **Explicit**: In constrast to frequency domain methods, AOC generates explicit convolution kernels in the spatial domain, allowing straightforward implementation in standard deep learning frameworks without specialized operations or significant computational overhead.
- **Flexible**: Supporting a range of essential operations – including striding, transposed convolutions for upsampling, grouped convolutions, and dilation – AOC adapt effectively to modern neural network architecture.
- **Scalable**: Designed for large-scale applications, our implementation maintains efficiency, incurring only a 10% slowdown compared to unconstrained models in realistic IN1K [17] training setup, as tested on ResNet [24] architecture.

To underscore the advantages of our method, we include a comparative summary in Table 1, highlighting support for key features across different methods. By combining orthogonality, explicit construction, and flexibility, our approach seamlessly bridges theoretical rigor with practical efficiency in deep learning models.

The paper is organized as follows: Section 2 outlines the three main aspects of AOC – its core tools, kernel construction, and scalable implementation. Section 3 presents an evaluation of the method's performance in terms of speed and expressive power. Finally, Section 4 discusses how our methodology can enhance existing methods in the literature.

2. An Adaptive scheme to build Orthogonal Convolution (AOC)

We will first recall what is the Block Convolution in 2.1. This tool allows the explicit construction of orthogonal kernels, which also support modern features depicted in 2.2. Finally, 2.3 provides implementation details that allow the method to scale.

2.1. Core tool: Block Convolution

Our approach builds upon three foundational papers: [64], which generalized orthogonal initialization to convolution to enable training networks with 10 000 layers, though without addressing constrained training; [53], which tackled this for separable convolutions; and [31], which extended it to general 2D convolutions. These works rely heavily on a

tool known as block convolution. In this section, we review, clarify, and extend this mathematical framework.

Notations: We consider convolutional layers characterized by c_o, the number of output channels; c_i, the number of input channels; $k_1 \times k_2$, the kernel size; s, the stride parameter; and g, the number of groups. For simplicity in the notation, we fix the group parameter to $g = 1$ by default (all proofs hold for other values of g, see Section 2.2.4). Also, we assume circular padding in all proofs. The kernel tensor of the convolution is denoted $\mathbf{K} \in \mathbb{R}^{c_o \times c_i \times k_1 \times k_2}$, while $x \in \mathbb{R}^{c_i \times h \times w}$ denotes the input tensor. We describe the convolution operation with three different notations:

$$y = \mathbf{K} \star_s x \quad \text{(Kernel notation)} \tag{1}$$
$$\bar{y} = \mathcal{S}_s \mathcal{K} \bar{x} \quad \text{(Toeplitz notation)} \tag{2}$$
$$y = \texttt{conv}_K(x, \texttt{stride} = s) \quad \text{(Code notation)} \tag{3}$$

Equation 1 defines the convolution operation with kernel $\mathbf{K}$ and stride s applied on x. Equation 2 highlights that this convolution is equivalent to a linear operation defined by a matrix product between a Toeplitz matrix $\mathcal{K} \in \mathbb{R}^{c_o hw \times c_i hw}$ and a vector $\bar{x} \in \mathbb{R}^{c_i hw}$, which is obtained by flattening x. The striding operation is represented by a masking diagonal matrix $\mathcal{S}_s \in \mathbb{R}^{c_o \frac{h}{s} \frac{w}{s} \times c_o hw}$ with ones on the selected entries and zeros elsewhere. When $s = 1$, we have $\mathcal{S}_1 = \mathcal{I}$, the identity matrix (with kernel $\mathbf{I}$). Equation 3 shows these notations in pseudo-code form.

Definition 2.1 (Block-convolution $\circledast$[1])**.** *The block convolution, denoted as $\mathbf{B} \circledast \mathbf{A}$, computes the equivalent kernel of the composition of two convolutional kernels, $\mathbf{A}$ and $\mathbf{B}$, enabling their combined effect without performing each convolution separately.*

$$(\mathbf{B} \circledast \mathbf{A}) \star_s x = \mathbf{B} \star_s (\mathbf{A} \star_1 x) \tag{4}$$
$$\mathcal{S}_s(\mathcal{B}\mathcal{A})\bar{x} = (\mathcal{S}_s \mathcal{B})\mathcal{A}\bar{x} \tag{5}$$
$$\texttt{conv}_{B \circledast A}(x, s) = \texttt{conv}_B(\texttt{conv}_A(x, 1), s) \tag{6}$$

This operator assumes that the number of input channels of the second convolution $\mathbf{B}$ matches the number of output channels of the first convolution $\mathbf{A}$ (condition denoted as $A \bowtie B$). Given $\mathbf{A} \in \mathbb{R}^{c_{int} \times c_i \times k_1^A \times k_2^A}$ and $\mathbf{B} \in \mathbb{R}^{c_o \times c_{int} \times k_1^B \times k_2^B}$, then $\mathbf{B} \circledast \mathbf{A} \in \mathbb{R}^{c_o \times c_i \times (k_1^A + k_1^B - 1) \times (k_2^A + k_2^B - 1)}$. The computation of Block-convolution kernel weights is given by:

$$(\mathbf{B} \circledast \mathbf{A})_{m,n,i,j} = \sum_{c=0}^{c_{int}-1} \sum_{i'=0}^{k_1^B - 1} \sum_{j'=0}^{k_2^B - 1} \mathbf{B}_{m,c,i',j'} \cdot \mathbf{A}_{c,n,i-i',j-j'}$$

where $\mathbf{A}$ is zero-padded, i.e., $\mathbf{A}_{c,n,i,j} = 0$ if $i \notin [0, k_1^A[$ or $j \notin [0, k_2^A[$. While a classic result, for completeness, we provide the proof for 1D convolution kernels in Appendix E.2. This operator $\circledast$, using matrix products between order 4 tensors, should not be confused with standard convolution, which takes a 4-dimensional tensor and a 3-dimensional input tensor. The complexity of these weight computations is $\mathcal{O}(c_o c_i c_{int} \prod_{p=1}^{2} (k_p^A + k_p^B - 1) k_p^B)$, necessitating an efficient implementation, detailed in Section 2.3. Among others, this operator enjoys several properties:

Proposition 2.2 (Associativity)**.** *The $\circledast$ operation is associative (given compatible kernels $A \bowtie B$ and $B \bowtie C$):*

$$A \circledast (B \circledast C) = (A \circledast B) \circledast C$$

Proposition 2.3 (Bi-linearity)**.** *: Given two convolutions A and B with the same channel sizes, a third convolution C compatible with A and B ($A \bowtie C$, $B \bowtie C$), and two scalars $\lambda_1, \lambda_2 \in \mathbb{R}$:*

$$(\lambda_1 A + \lambda_2 B) \circledast C = \lambda_1 A \circledast C + \lambda_2 B \circledast C$$

Proposition 2.4 (Non-Commutativity)**.** *: Even when $A \bowtie B$ and $B \bowtie A$ hold, Block-convolution is not commutative, as convolution composition is generally not commutative:*

$$\mathcal{A}\mathcal{B} \neq \mathcal{B}\mathcal{A} \implies \mathbf{A} \circledast \mathbf{B} \neq \mathbf{B} \circledast \mathbf{A}$$

2.2. Construction of Strided, Transposed, Grouped, Dilated, Orthogonal Convolutions with AOC

We first recall the concept of orthogonality for convolutions:

Definition 2.5 (Orthogonal Convolution)**.** *A convolution defined by a kernel $\mathbf{K}$ is row or column orthogonal if:*

$$(\mathcal{S}_s \mathcal{K})(\mathcal{S}_s \mathcal{K})^T = \mathcal{I} \quad \textit{(row orthogonal)}$$
$$(\mathcal{S}_s \mathcal{K})^T(\mathcal{S}_s \mathcal{K}) = \mathcal{I} \quad \textit{(column orthogonal)}$$

The type of orthogonality (row or column) is given by the matrix in the toeplitz notation (Eq. 2). When $c_i s^2 > c_o$, $\mathcal{S}_s \mathcal{K}$ is column orthogonal [1]. When $c_i s^2 = c_o$, $\mathcal{S}_s \mathcal{K}$ is a square matrix, and the two conditions are equivalent. Finally, when $s = 1$, the condition on the Toeplitz matrices is equivalent to $\mathbf{K} \circledast \mathbf{K}^T = \mathbf{I}$ (resp. $\mathbf{K}^T \circledast \mathbf{K} = \mathbf{I}$). A formal definition of $\mathbf{K}^T$ can be found in Definition 2.9.

On one side, methods like BCOP and SC-Fac support any kernel size, but striding is emulated in a way that makes the convolution more costly than without striding. This limitation is even more pronounced in methods such as SOC, LOT, ECO, and Cayley, which handle channel changes (when $c_i \neq c_o$) through padding or channel dropping. On the other side, RKO [47, 48] offers advantages like effective channel handling and efficient striding, but it is not orthogonal when stride $\neq$ kernel size (see 2.2.2). By leveraging the ability to fuse kernels, we explore whether combining methods can offset the drawbacks of each component.

[1] Initially denoted by [31] as $\square$

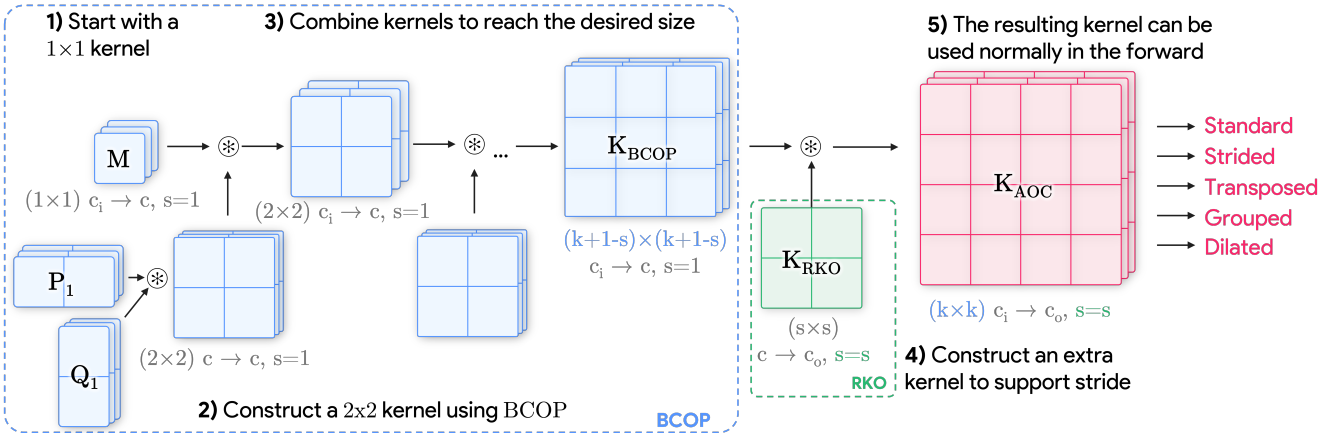


Figure 1. **AOC enables the construction of orthogonal kernels with customizable sizes and strides.** By leveraging the $\circledast$ operator, we can fuse kernels obtained from two existing methods, namely BCOP and RKO. With this approach, we can build orthogonal kernels that support native striding, effectively canceling the drawbacks of the two base methods.

2.2.1 Standard Orthogonal Convolution

We unify the works of [64], [31], and [53] within a consistent notation framework, highlighting similarities and differences among their approaches to constructing standard orthogonal convolutions (i.e., without stride, transposition, grouping, or dilation). These methods primarily rely on constructing elementary blocks (1×1, 1×2, and 2×1 orthogonal convolutions) and assembling these blocks to create orthogonal convolutions of the desired size and shape.

From Matrices to Orthogonal 1×1 Convolutions. A substantial body of research exists on building orthogonal matrices $M \in \mathbb{R}^{c_o \times c_i}$. One common approach involves applying a differentiable projection operator to an unconstrained weight matrix, yielding an orthogonal matrix such that $MM^T = I$ or $M^TM = I$. Various methods exist, including the Björck and Bowie orthogonalization scheme [8], the exponential method [51], the Cayley method [55], and QR factorization [57]. An orthogonal matrix can easily be reshaped into a convolution kernel with a 1×1 kernel $\mathbf{M} \in \mathbb{R}^{c_o \times c_i \times 1 \times 1}$, and such a convolution is orthogonal if M is orthogonal. These convolutions are mainly used to change the number of channels (Fig. 1-1).

From Matrices to 1×2 Orthogonal Convolutions. Stacking two orthogonal 1×1 convolution kernels along their last dimensions[2] leads to a 1×2 convolution, though it is generally not orthogonal. Authors of [53, 64] noted that additional constraints are needed, proposing a half-rank symmetric projector to construct a 1×2 orthogonal convolution: from a column-orthogonal matrix $M \in \mathbb{R}^{c \times \frac{c}{2}}$[3], the matrix $N = MM^T \in \mathbb{R}^{c \times c}$ is a symmetric projector that satisfies:

$$N = N^2 = N^T \quad \text{and} \quad (I - N) = (I - N)^2 = (I - N)^T$$

These two matrices can be reshaped into $c \times c \times 1 \times 1$ convolution kernels, and stacking them along the last dimension results in an orthogonal $c \times c \times 1 \times 2$ convolution kernel:

$$\mathbf{P} = \texttt{stack}([\mathbf{N}, \mathbf{I} - \mathbf{N}], \texttt{axis} = -1) \Rightarrow \mathbf{P} \circledast \mathbf{P}^T = \mathbf{I}$$

Similarly, stacking along the penultimate dimension, $\mathbf{Q} = \texttt{stack}([\texttt{N}, \texttt{I} - \texttt{N}], \texttt{axis} = -2)$, results in an orthogonal $c \times c \times 2 \times 1$ kernel. Although already proven by previous work, proof of this can be found in Appendix E.

From 1×2 to $k_1 \times k_2$ Orthogonal Convolutions. The three papers propose to build standard orthogonal convolutions by composing smaller kernels based on the following properties:

Proposition 2.6 (Composition of Orthogonal Convolutions)**.** *The composition of two row orthogonal convolutions is a row orthogonal convolution:*

$$\mathcal{A}\mathcal{A}^T = \mathcal{I} \quad \textit{and} \quad \mathcal{B}\mathcal{B}^T = \mathcal{I} \implies \mathcal{A}\mathcal{B}(\mathcal{A}\mathcal{B})^T = \mathcal{I}$$

The same applies to two column orthogonal convolutions. However, the composition of a row orthogonal with a column orthogonal convolution is, in general, not orthogonal.

Using block convolutions, we can represent the composition of 1×2 and 2×1 kernels[4] to obtain a kernel with any desired shape. The differences among the three approaches

[2] Done in practice with `torch.stack([K_1, K_2], axis=-1)`

[3] This implies that $c \geq 2$. When c is even, $\lfloor \frac{c}{2} \rfloor$ is used in practice

[4] As indicated in Definition 2.5, k_1 and k_2 parameters do not affect row or column orthogonality

lie in the composition order: Authors of [53] chose to compose $(k_1 - 1)$ 2×1 kernels $\mathbf{P_i}$, followed by a 1×1 kernel $\mathbf{M}$, and $(k_2 - 1)$ 1×2 kernels $\mathbf{Q_i}$ to form a $k_1 \times k_2$ kernel:

$$\mathbf{K}_{\text{SC-Fac}} = \underbrace{\mathbf{P_{k_1-1}} \circledast \ldots \circledast \mathbf{P_1}}_{\text{all 1x2 kernels}} \circledast \underbrace{\mathbf{M}}_{\text{1x1}} \circledast \underbrace{\mathbf{Q_1} \circledast \ldots \circledast \mathbf{Q_{k_2-1}}}_{\text{all 2x1 kernels}}$$

On the other hand, authors of [31][64] alternated 2×1 and 1×2 kernels, ending with a 1×1 convolution:

$$\mathbf{K}_{\text{BCOP}} = \underbrace{(\mathbf{P_{k-1}} \circledast \mathbf{Q_{k-1}})}_{\text{pairs of 1x2 and 2x1 kernels}} \circledast \ldots \circledast (\mathbf{P_1} \circledast \mathbf{Q_1}) \circledast \underbrace{\mathbf{M}}_{\text{1x1}}$$

Both approaches have incomplete parametrizations: the first is limited to separable convolutions, while the second shows counterexamples in the general 2D convolution case. However, both methods use the same number of parameters for a given kernel size. Building a complete parametrization of 2D convolutions remains an open question, discussed in Appendix D. We thus base our work on the BCOP parametrization [31] for two main reasons: (1) any 2×2 convolution not parametrizable by BCOP can be represented with a 3×3 kernel – a feasible solution given the trend toward larger kernels [18, 56]; (2) BCOP enables a faster and less memory-intensive implementation (see Section 2.3), unlocking larger networks that compensate for any potential expressiveness loss.

2.2.2 Native Striding for Orthogonal Convolutions

Classical convolutional networks use strided convolutions. However, most existing work on orthogonal convolutions does not support stride directly, instead emulating striding via a reshaping operation [49]: transforming the $c_i \times I_h \times I_w$ tensor into a $c_i s^2 \times \frac{I_h}{s} \times \frac{I_w}{s}$ tensor, followed by a non-strided convolution. This emulation requires more parameters ($c_i s^2 k_1 k_2$) than its non-strided counterpart, unlike native striding. Beyond this limitation, emulated striding also prevents native implementation of transposed convolutions (see Section 2.2.3). In this section, we propose a method to construct orthogonal kernels that support native striding.

To our knowledge, only two works [47, 48] claim to use native stride. These rely on a method referred to by [31] as Reshaped Kernel Orthogonalization (RKO). This method involves reshaping the kernel $\mathbf{K}_{\text{RKO}} \in \mathbb{R}^{c_o \times c_i \times k_1 \times k_2}$ into a matrix $K' \in \mathbb{R}^{c_o \times c_i k_1 k_2}$ and orthogonalizing it. In general, with the adequate multiplicative factor, the resulting convolution is 1-Lipschitz but not orthogonal. In this work, we prove that no additional factor is required when $k = s$ to obtain an orthogonal convolution:

Proposition 2.7 (RKO gives an orthogonal kernel when $k_1 = k_2 = s$)**.** *When $K' \in \mathbb{R}^{c_o \times c_i k k}$ is orthogonal, the convolution with the reshaped kernel $\mathbf{K}_{RKO} \in \mathbb{R}^{c_o \times c_i \times k \times k}$ and a stride $s = k$ is orthogonal. The formal proof can be found in Appendix E.*

The proposed method, called AOC, combines the BCOP and RKO methods to construct a Strided Convolution with Arbitrary Kernel Size:

$$\mathbf{K}_{AOC} = \mathbf{K}_{\text{RKO}} \circledast \mathbf{K}_{\text{BCOP}} \tag{7}$$

As shown in Fig. 1, $\mathbf{K}_{\text{BCOP}} \in \mathbb{R}^{c \times c_i \times (k_1+1-s) \times (k_2+1-s)}$ is fused with $\mathbf{K}_{\text{RKO}} \in \mathbb{R}^{c_o \times c \times s \times s}$, resulting in a kernel $k_1 \times k_2$ that can be used with stride s. This formulation uses an internal channel size c, which is set to $\max(c_i, \lfloor \frac{c_o}{s^2} \rfloor)$ to preserve orthogonality. According to [1], no orthogonal kernel exists when $s > k$. Our proposed approach thus covers all valid configurations of orthogonal convolutions since $k + 1 - s \geq 0$.

Proposition 2.8 (Orthogonality of strided AOC (Informal))**.** *Setting $c = max(c_i, \lfloor \frac{c_o}{s^2} \rfloor)$ yields an orthogonal convolution. The complete proof can be found in Appendix E.*

The proof relies on the fact that Proposition 2.6 also holds for strided convolutions applied on $\mathcal{S}_s \mathcal{K}_{\text{RKO}}$. We prove that when $\min(c_i, \frac{c_o}{s^2}) \leq c \leq \max(c_i, \frac{c_o}{s^2})$, the two matrices $\mathcal{K}_{\text{BCOP}}$ and $\mathcal{S}_s \mathcal{K}_{\text{RKO}}$ are either both row-orthogonal or both column-orthogonal. The choice of $c = \max(c_i, \lfloor \frac{c_o}{s^2} \rfloor)$ maximizes the expressiveness of the parametrization while ensuring that the resulting convolution is orthogonal.

2.2.3 Native Transposed Orthogonal Convolutions:

In addition to the practical verification of definition 2.5, transposed convolutions are mostly used as learnable upscaling layers in architectures such as U-Net [43] or VAEs [28, 58]. For a given convolution with stride defined in 2, it corresponds to the application of the transposed matrix $(\mathcal{S}_s \mathcal{K})^T$, inverting the role of c_i and c_o. The resulting operation can be defined by the three notations:

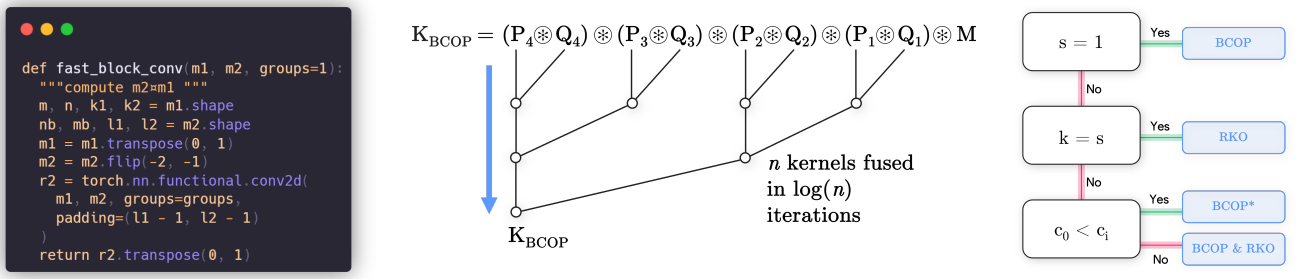
Definition 2.9 (Transposed Convolution)**.** *A transposed convolution is defined as follows:*

$$y = \mathbf{K}^T \star_{\frac{1}{s}} x \tag{8}$$

$$\bar{y} = (\mathcal{S}_s \mathcal{K})^T \bar{x} = \mathcal{K}^T \mathcal{S}_s^T \bar{x} \tag{9}$$

$$y = \texttt{ConvTranspose}_K(x, \texttt{stride} = s) \tag{10}$$

The code notation (Equation 10) corresponds to the implementation in PyTorch parametrized by the original kernel $\mathbf{K}$. The equation 9 corresponds to the transposition of the underlying Toeplitz matrix. The kernel notation (Equation 8) can be viewed as a standard convolution with a transposed kernel and fractional striding. The kernel $\mathbf{K}^T \in \mathbb{R}^{c_i \times c_o \times k_1 \times k_2}$ is obtained by transposing the channel dimensions (the first ones) and reversing the kernel ones (the last two).



(a) **Fast block convolution.** We optimized the 2D convolution in order to compute the $\circledast$ operator with maximum parallelism.

(b) **Parallelize BCOP iterations.** By leveraging associativity of $\circledast$, we can compute the n iteration in $\mathcal{O}\log(n)$ steps using parallel associative scan.

(c) **Parsimonious parametrization.** The method can sometimes simplify to quicker equivalent parametrization.

Figure 2. **Beyond BCOP and RKO.** We achieve a highly scalable parametrization thanks to optimizations at every level of our method: starting from the $\circledast$ operator 2a, to BCOP 2b, to our complete method 2c. It results in a method with a lower overhead as scale increases.

This construction results in an orthogonal transposed convolution. Although the proof is straightforward, its practical application requires the explicit construction of a strided orthogonal convolution kernel (as detailed in 7).

Proposition 2.10 (Transposed Convolution)**.** *The transposition of a row orthogonal convolution is a column orthogonal convolution, and vice versa. The proof follows from combining Definitions 2.5 and 9.*

2.2.4 Native Grouped Orthogonal Convolutions:

Most modern CNNs use grouped convolutions [25, 33, 65]. Beyond its advantages in terms of parameters and computational efficiency, it makes AOC more efficient as its parametrization can be parallelized, similarly as [21, 32]. Given a group number g, the kernel of a grouped convolution $\mathbf{K} \in \mathbb{R}^{c_o \times \frac{c_i}{g} \times k \times k}$ can be viewed as a stack of g kernels $\mathbf{K_i} \in \mathbb{R}^{\frac{c_o}{g} \times \frac{c_i}{g} \times k \times k}$, each constructed independently. Note that c_o and c_i must be multiple of g.

Proposition 2.11 (Grouped Orthogonal Convolution)**.** *A grouped convolution composed of g kernels $(\mathbf{K_i})_g$ is orthogonal if and only if each individual convolution of kernel $\mathbf{K_i}$ is orthogonal.*

The proof uses the fact that the Toeplitz matrix of a grouped convolution is block diagonal. This matrix is orthogonal if and only if each block is orthogonal. When $g = c_i = c_o$, each kernel $\mathbf{K}_i$ has a single channel $c_i = c_o = 1$ that cannot be built with BCOP which requires $c \geq 2$.

2.2.5 Native Orthogonal Convolutions with Dilation:

Introduced by [67], Dilation is an effective means to increase the receptive field of a convolution without increasing its number of parameters or its computational cost. In [53], the authors stated in an appendix that "any filter bank that is orthogonal for a standard convolution is also orthogonal for a dilated convolution and vice versa." While this is mathematically accurate, it is essential to note that circular padding must be adjusted accordingly to remain within the scope of their theorem. Our method thus also supports orthogonal convolution with dilation.

2.3. Efficient implementation of AOC

Beyond a mathematical framework that unlocks a more flexible use of orthogonal convolutions, we propose several design choices for an implementation that scales well to larger kernels, larger images and larger batch sizes. Although AOC includes the construction of BCOP and RKO kernels, our implementation improves the original ones at many stages. It results in an **8x** reduction of the original overhead in realistic settings. Our implementation will be integrated at https://github.com/thib-s/orthogonium.

Fast implementation of the block convolution: To the best of our knowledge, the only differentiable implementation of the BCOP method is available in the reference code [31]. However, since there is no cuda kernel available for block convolution, the authors relied on nested loops to perform all matrix multiplication to compute the resulting kernel. Unfortunately, this approach prevents PyTorch from parallelizing the nested loops.

However, our analysis have shown that this can be efficiently paralleled in a single operation. Inspired by [60] and [15], which aimed to compute $A \circledast A^T$ to prove orthogonality, we propose to replace the computation $B \circledast A$ by a convolution with zero padding between B and A^T. This approach can also be seen as a specific case of convolutional einsum [41]. This operation can be rewritten by re-ordering the summation to use a 2D convolution at its core. The strategy is to use the 2D convolution to compute one output

Name	Batch Size	Train Time (ms)	Train Memory (GB)	Test Time (ms)	Test Memory (GB)
Conv2D (ref)	128	137 (1.00x)	4.7 (1.00x)	50 (1.00x)	1.4 (1.00x)
AOC (ours)	128	**239 (1.75x)**	**5.3 (1.15x)**	**53 (1.06x)**	**1.4 (1.02x)**
BCOP	128	389 (2.85x)	8.6 (1.84x)	62 (1.25x)	1.5 (1.06x)
SOC	128	664 (4.86x)	12.8 (2.73x)	429 (8.55x)	1.8 (1.30x)
Cayley	128	584 (4.27x)	19.0 (4.07x)	247 (4.94x)	2.0 (1.45x)
Conv2D (ref)	256	284 (1.00x)	9.1 (1.00x)	91 (1.00x)	2.7 (1.00x)
AOC (ours)	256	**354 (1.25x)**	**9.8 (1.07x)**	**97 (1.06x)**	**2.7 (1.01x)**
BCOP	256	624 (2.20x)	13.9 (1.53x)	135 (1.48x)	2.8 (1.03x)
Conv2D (ref)	512	550 (1.00x)	17.9 (1.00x)	172 (1.00x)	5.3 (1.00x)
AOC (ours)	512	**622 (1.13x)**	**18.6 (1.04x)**	**176 (1.02x)**	**5.4 (1.01x)**
BCOP	512	1116 (2.03x)	24.6 (1.38x)	256 (1.48x)	5.4 (1.02x)

Table 2. **AOC benefits from scale.** As demonstrated on a ResNet-34, previous methods impose significant overhead when input images are large (224×224). In contrast, since our method's computational cost is independent of layer input size, its overhead decreases as batch size increases. Furthermore, the low memory overhead enables larger batches at scale.

filter. Then, the batch dimension can be used to compute all output filters in parallel. The code is detailed in the Fig. 2a.

Reducing time complexity of BCOP: Beyond the efficient parallelism of the $\circledast$ operation, we propose to parallelize the whole kernel computation: the parametrization can be seen as the composition of many small kernels. Not only can those be created in parallel, but they can also be combined efficiently. As the $\circledast$ is an associative operation (prop 2.2), we can leverage the parallel associative scan [13, 69] to parallelize the iterations of the original algorithm. The original $2*(k-s)$ sequential $\circledast$ operations can then be done in $\mathcal{O}(\log(k-s))$ iterations (Fig. 2b). This is unlocked in practice if $\circledast$ implementation supports batching. Unfortunately, the batch dimension of our efficient implementation already uses the dimension originally dedicated to batching to compute the c_o channels, so it cannot be used directly. We propose to circumvent this by using grouped conv2d implementation. By concatenating g kernels and setting groups $= g$, we can compute the batched $\circledast$ in parallel.

Efficient implementation. By examining Definition 7, one observes that, depending on the values of s and k, the parametrization can be simplified to either $\mathbf{K}_{AOC} = \mathbf{K}_{\text{BCOP}}$ or $\mathbf{K}_{AOC} = \mathbf{K}_{\text{RKO}}$. While BCOP is generally not suited for handling stride directly, we have identified specific cases – namely when $c_i < c_o$ – where stride can indeed be applied directly to a BCOP kernel (see Proof E.5) without requiring the full parametrization. Although not proposed in [31], this observation refines our overall characterization of BCOP's limitations with stride, showing that exceptions exist under certain conditions. The complete decision tree used in our implementation is shown in Figure 2c, with each branch's orthogonality rigorously validated throughout the paper.

3. Evaluation

Scalability. As observed by [39], a method's implementation is a key factor for its success: a slow implementation leads to increased training time and, consequently, lower performances in practical contexts like robust training. In this section, we demonstrate that AOC offers a key advantage: its computational cost does not depend on the input size or shape, making it well-suited for large-scale datasets like ImageNet [17], where handling large images is crucial. Although other methods may perform better on smaller datasets such as CIFAR [30] or Tiny ImageNet [63], they struggle to scale to widely used architectures like ResNet-34 [24], as shown in Table 2. On the other end, our method's low memory cost enables larger batch sizes, and since our parameterization is batch-size independent, the overhead decreases as batch size increases. Ultimately, this results in a training time only 13% slower than its unconstrained counterpart.

Experiments were conducted on a minimally modified ResNet-34 architecture, chosen for its compatibility with various orthogonal layers and its status as a standard benchmark for ImageNet training. The transition blocks were replaced by a single, strided convolution to maintain simplicity and ensure compatibility with existing orthogonal layers. For each method, we measured the average training and testing times over 100 batches and recorded peak memory consumption. Starting with a batch size of 128, we doubled the batch size incrementally until encountering an out-of-memory error. Each method's performance was compared to a standard convolution baseline, with results reported as overhead percentages. All experiments were performed on a consumer-grade computer equipped with two NVIDIA RTX 4090 GPUs.

To ensure a fair comparison, the hyperparameters of SOC and Cayley were set to their default values. We adjusted the number of Björck iterations to the same values

for BCOP and AOC. While originally set to 20 iterations, our unit testing scheme (see appendix B) shows that 12 iterations are sufficient to ensure a stable rank [44] of 99.9% of the full rank. Finally, standard `Conv2D` is used with circular padding to evaluate the overhead induced by our parametrization rather than the overhead induced by the padding.

	Models	Accuracy	Provable Accuracy $\epsilon = \frac{36}{255}$	Trainable Parameters
CIFAR10	BCOP	72.2	58.26	2.6M
	GloRo	77.0	58.40	8.0M
	Local-Lip-B	77.4	60.70	2.3M
	Cayley Large	74.6	61.40	21.0M
	SOC 20	78.0	62.70	27.0M
	SOC+ 20	76.3	62.60	27.0M
	CPL XL	78.5	64.40	236.0M
	AOL Large	71.6	64.00	136.0M
	SLL Small	71.2	62.60	41.0M
	SLL Medium	72.2	64.30	78.0M
	SLL Large	72.7	65.00	118.0M
	SLL X-Large	73.3	65.80	236.0M
	AOC $m = \frac{72}{255}$	80.0	60.12	41.3M
	AOC $m = \frac{3\sqrt{2}}{2}$	74.0	64.33	41.3M
IN1K	**AOC** Cosine	68.2	00.00	53.0M
	AOC $m = \frac{3\sqrt{2}}{2}$	42.1	26.31	53.1M

Table 3. **AOC is competitive on small-scale datasets and enables affordable training on large-scale datasets.** For both CIFAR10 (top) and Imagenet-1K (bottom), we evaluate our model under two settings: one emphasizing clean accuracy and another emphasizing robustness.

Expressive power. The most straightforward way to evaluate the expressiveness of our layers is to apply them to classification problems. It is well known that 1-Lipschitz constrained networks face an inherent trade-off between accuracy and robustness [7]. Fortunately, this trade-off can be controlled using the loss's parameters: we can then evaluate our model at two different points of the trade-off to show that it can achieve both a decent accuracy and a decent certifiable robustness. Our approach is not expected to improve the expressiveness of the original building blocks like BCOP and RKO. However, authors of [39] observed that improvement of the state of the art in certifiable adversarial robustness comes along with larger architectures and longer training times. Therefore, given the same computational budget, we can expect that our efficient implementation unlocks larger networks and more training steps, which in turn should improve final performance.

In our evaluation on CIFAR10, we scaled the network from [31] to be wider and deeper: our model consists of 4 blocks of 3 convolutions interleaved with strided convolutions where the number of channels was increased by a factor of 2x. The head of the network consists of 5 dense layers. The number of channels of the first block was set to 128, and the size of dense layers was set to 1024. A detailed description of our training setting can be found in Appendix F. We changed the loss to the CrossEntropy loss as in [38], which allows us to control the accuracy/robustness trade-off. We did not use techniques such as last layer normalization, certificate regularization [51], or DDPM augmentation [26] to obtain our results, which are shown in the first part of Table 3. We controlled three elements during the hyper-parameter tuning process:

- **Loss parameters**: Increasing the margin in the loss function helps improve training robustness, but reduces training accuracy.
- **Model size**: Increasing the model size generally improves training accuracy.
- **Data augmentation**: Increasing data augmentation reduces training accuracy but can improve validation accuracy, especially when training accuracy is greater than validation accuracy.

The tuning process begins with a given model. We first tune the learning rate and increase the margin in the loss function to improve robustness until training accuracy drops below 100%. Then, we increase the model size and apply more data augmentation until training accuracy falls below validation accuracy.

In our evaluation on ImageNet1K, we leveraged the flexibility of our layers to design more complex blocks. A comprehensive description of the whole architecture is given in Appendix F. Each block consists of a depthwise convolution that doubles the number of channels, followed by a MaxMin activation function [2], and a pointwise convolution that reduces the number of channels. These layers are encapsulated within a skip connection featuring a learnable factor to ensure a Lipschitz constant of 1. The network was constructed by repeating these blocks, using strided convolutions to progressively reduce the spatial dimensions while increasing the channel dimensions. The architecture ends with L2 Norm Pooling [10], followed by a single dense layer for classification. We evaluated the network under two distinct settings. In the first setting, we used Cosine similarity to maximize accuracy, focusing on achieving optimal performance for clean data. In the second setting, we applied categorical cross-entropy with a margin of $m = 1.5\sqrt{2}$ to emphasize robustness against adversarial examples. The results for both settings are presented in the second part of Table 3.

4. Conclusion and Broader Impact

We introduced AOC, a method for constructing orthogonal convolutions that supports essential features such as stride, transposition, groups, and dilation. Our results demonstrate that this layer is both expressive and scalable. Beyond its standalone benefits, our framework enhances existing layers: in Appendix C , we integrate our method with SLL [3] to build an efficient downsampling residual block, propose optimizations to reduce the memory footprint of SOC [50], and present a strategy to make Sandwich layers [61] scalable for convolutions. Also, although our experiments used AOC in isolation, we think this layer is intended to be seamlessly combined with other methods, such as SLL, where each approach's strengths can amplify the other's capabilities. We believe this work opens pathways for advancing modern convolutional architectures. In support of further research and development, we have made our implementation available at https://github.com/thib-s/orthogonium.

5. Acknowledgement

This work has benefited from the support of the DEEL project, [5] with funding from the Agence Nationale de la Recherche, part of the ANITI AI cluster.

References

[1] El Mehdi Achour, François Malgouyres, and Franck Mamalet. Existence, stability and scalability of orthogonal convolutional neural networks. Journal of Machine Learning Research, 23(347):1–56, 2022. 1, 3, 5

[2] Cem Anil, James Lucas, and Roger Grosse. Sorting out lipschitz function approximation. In International Conference on Machine Learning, pages 291–301. PMLR, 2019. 1, 8, 12, 18

[3] Alexandre Araujo, Aaron J Havens, Blaise Delattre, Alexandre Allauzen, and Bin Hu. A unified algebraic perspective on lipschitz neural networks. In The Eleventh International Conference on Learning Representations, 2023. 2, 9, 14

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In International conference on machine learning, pages 214–223. PMLR, 2017. 1, 13

[5] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep networks? Advances in Neural Information Processing Systems, 31, 2018. 1, 13

[6] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In International conference on machine learning, pages 573–582. PMLR, 2019. 1, 12

[7] Louis Béthune, Thibaut Boissin, Mathieu Serrurier, Franck Mamalet, Corentin Friedrich, and Alberto Gonzalez Sanz. Pay attention to your loss: understanding misconceptions about lipschitz neural networks. Advances in Neural Information Processing Systems, 35:20077–20091, 2022. 1, 8

[8] Åke Björck and Clazett Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. SIAM Journal on Numerical Analysis, 8(2):358–364, 1971. 4

[9] Ali Ebrahimpour Boroojeny, Matus Telgarsky, and Hari Sundaram. Spectrum extraction and clipping for implicitly linear layers. In International Conference on Artificial Intelligence and Statistics, pages 2971–2979. PMLR, 2024. 13

[10] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In Proceedings of the 27th international conference on machine learning (ICML-10), pages 111–118, 2010. 8

[11] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In International Conference on Learning Representations, 2018. 1, 13

[12] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In International conference on machine learning, pages 854–863. PMLR, 2017. 1, 12

[13] Hillis W Daniel. Data parallel algorithms. Commun. AGM, 29(12):1170–1183, 1986. 7

[14] Aaron Defazio, Xingyu Alice Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. The road less scheduled, 2024. 18

[15] Blaise Delattre, Quentin Barthélemy, Alexandre Araujo, and Alexandre Allauzen. Efficient bound of lipschitz constant for convolutional layers by gram iteration. In International Conference on Machine Learning, pages 7513–7532. PMLR, 2023. 6, 13

[16] Blaise Delattre, Quentin Barthélemy, and Alexandre Allauzen. Spectral norm of convolutional layers with circular and zero paddings. arXiv preprint arXiv:2402.00240, 2024. 13

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009. 2, 7

[18] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In Proceedings

[5] https://www.deel.ai/

of the IEEE/CVF conference on computer vision and pattern recognition, pages 11963–11975, 2022. 5, 14

[19] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. arXiv preprint arXiv:1605.08803, 2016. 1, 12

[20] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. Advances in neural information processing systems, 32, 2019. 13

[21] Mikhail Gorbunov, Nikolay Yudin, Vera Soboleva, Aibek Alanov, Alexey Naumov, and Maxim Rakhuba. Group and shuffle: Efficient structured orthogonal parametrization. arXiv preprint arXiv:2406.10019, 2024. 6

[22] Ekaterina Grishina, Mikhail Gorbunov, and Maxim Rakhuba. Tight and efficient upper bound on spectral norm of convolutional layers, 2024. 13

[23] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. Advances in neural information processing systems, 30, 2017. 1, 13

[24] 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, pages 770–778, 2016. 2, 7

[25] Andrew G Howard. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017. 6

[26] Kai Hu, Klas Leino, Zifan Wang, and Matt Fredrikson. A recipe for improved certifiable robustness. In The Twelfth International Conference on Learning Representations, 2023. 2, 8

[27] Bobak Kiani, Randall Balestriero, Yann LeCun, and Seth Lloyd. projunn: efficient method for training deep networks with unitary matrices. Advances in Neural Information Processing Systems, 35:14448–14463, 2022. 1, 2, 12

[28] Diederik P Kingma. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013. 5

[29] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. Advances in neural information processing systems, 31, 2018. 1, 12

[30] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 7

[31] Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger B Grosse, and Jörn-Henrik Jacobsen. Preventing gradient attenuation in lipschitz constrained convolutional networks. Advances in neural information processing systems, 32, 2019. 1, 2, 3, 4, 5, 6, 7, 8, 16, 17, 18

[32] Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, et al. Parameter-efficient orthogonal finetuning via butterfly factorization. In The Twelfth International Conference on Learning Representations. 6

[33] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 11976–11986, 2022. 6

[34] Laurent Meunier, Blaise J Delattre, Alexandre Araujo, and Alexandre Allauzen. A dynamical system perspective for lipschitz neural networks. In International Conference on Machine Learning, pages 15484–15500. PMLR, 2022. 2

[35] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In International Conference on Learning Representations, 2018. 1, 13

[36] Jan Müller, Reinhard Klein, and Michael Weinmann. Orthogonal wasserstein gans. arXiv preprint arXiv:1911.13060, 2019. 1, 13

[37] Patricia Pauli, Dennis Gramlich, and Frank Allgöwer. Lipschitz constant estimation for general neural network architectures using control tools, 2024. 13

[38] Bernd Prach and Christoph H Lampert. Almost-orthogonal layers for efficient general-purpose lipschitz networks. In European Conference on Computer Vision, pages 350–365. Springer, 2022. 2, 8

[39] Bernd Prach, Fabio Brau, Giorgio Buttazzo, and Christoph H Lampert. 1-lipschitz layers compared: Memory speed and certifiable robustness. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 24574–24583, 2024. 2, 7, 8

[40] Haozhi Qi, Chong You, Xiaolong Wang, Yi Ma, and Jitendra Malik. Deep isometric learning for visual recognition. In International conference on machine learning, pages 7824–7835. PMLR, 2020. 1, 13

[41] Tahseen Rabbani, Jiahao Su, Xiaoyu Liu, David Chan, Geoffrey Sangston, and Furong Huang. conv_einsum: A framework for representation and fast evaluation of multilinear operations in convolutional tensorial neural networks. arXiv preprint arXiv:2401.03384, 2024. 6

[42] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In International conference on machine learning, pages 1530–1538. PMLR, 2015. 1, 12

[43] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18, pages 234–241. Springer, 2015. 5

[44] Amartya Sanyal, Philip H Torr, and Puneet K Dokania. Stable rank normalization for improved generalization in neural networks and gans. In International Conference on Learning Representations. 8

[45] Hanie Sedghi, Vineet Gupta, and Philip M Long. The singular values of convolutional layers. In International Conference on Learning Representations, 2018. 13

[46] Alexandra Senderovich, Ekaterina Bulatova, Anton Obukhov, and Maxim Rakhuba. Towards Practical Control of Singular Values of Convolutional Layers. Advances in Neural Information Processing Systems, 35:10918–10930, 2022. 13

[47] Mathieu Serrurier, Franck Mamalet, Alberto González-Sanz, Thibaut Boissin, Jean-Michel Loubes, and Eustasio Del Barrio. Achieving robustness in classification using optimal transport with hinge regularization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 505–514, 2021. 3, 5

[48] Mathieu Serrurier, Franck Mamalet, Thomas Fel, Louis Béthune, and Thibaut Boissin. On the explainable properties of 1-lipschitz neural networks: An optimal transport perspective. Advances in Neural Information Processing Systems, 36, 2024. 3, 5

[49] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1874–1883, 2016. 5

[50] Sahil Singla and Soheil Feizi. Fantastic four: Differentiable and efficient bounds on singular values of convolution layers. In International Conference on Learning Representations. 9

[51] Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In International Conference on Machine Learning, pages 9756–9766. PMLR, 2021. 2, 4, 8

[52] Sahil Singla and Soheil Feizi. Improved techniques for deterministic l2 robustness. Advances in Neural Information Processing Systems, 35:16110–16124, 2022. 14

[53] Jiahao Su, Wonmin Byeon, and Furong Huang. Scaling-up diverse orthogonal convolutional networks by a paraunitary framework. In International Conference on Machine Learning, pages 20546–20579. PMLR, 2022. 1, 2, 4, 5, 6

[54] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In International Conference on Learning Representations, 2014. 1, 12

[55] Asher Trockman and J Zico Kolter. Orthogonalizing convolutional layers with the cayley transform. In International Conference on Learning Representations, 2021. 1, 2, 4, 15

[56] Asher Trockman and J. Zico Kolter. Patches Are All You Need? Technical Report arXiv:2201.09792, arXiv, 2022. Issue: arXiv:2201.09792 arXiv:2201.09792 [cs] type: article. 5

[57] Rianne Van Den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. In 34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018, pages 393–402. Association For Uncertainty in Artificial Intelligence (AUAI), 2018. 4

[58] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. Advances in neural information processing systems, 30, 2017. 5

[59] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. Advances in Neural Information Processing Systems, 31, 2018. 13

[60] Jiayun Wang, Yubei Chen, Rudrasis Chakraborty, and Stella X Yu. Orthogonal convolutional neural networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 11505–11515, 2020. 1, 6

[61] Ruigang Wang and Ian Manchester. Direct parameterization of lipschitz-bounded deep networks. In International Conference on Machine Learning, pages 36093–36110. PMLR, 2023. 2, 9, 15

[62] Zi Wang, Bin Hu, Aaron J Havens, Alexandre Araujo, Yang Zheng, Yudong Chen, and Somesh Jha. On the scalability and memory efficiency of semidefinite programs for lipschitz constant estimation of neural networks. In The Twelfth International Conference on Learning Representations, 2024. 13

[63] Jiayu Wu, Qixiang Zhang, and Guoxi Xu. Tiny imagenet challenge. Technical report, 2017. 7

[64] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In International Conference on Machine Learning, pages 5393–5402. PMLR, 2018. 1, 2, 4, 5, 16

[65] S Xie, RB Girshick, Piotr Dollár, Z Tu, and K He. Aggregated residual transformations for deep neural networks. corr. arXiv preprint arxiv:1611.05431, 2016. 6

[66] Xiaojun Xu, Linyi Li, and Bo Li. Lot: Layer-wise orthogonal training on improving l2 certified robustness. Advances in Neural Information Processing Systems, 35:18904–18915, 2022. 1, 2

[67] F Yu. Multi-scale context aggregation by dilated convolutions. arXiv preprint arXiv:1511.07122, 2015. 6

[68] Tan Yu, Jun Li, Yunfeng Cai, and Ping Li. Constructing orthogonal convolutions in an explicit manner. In International Conference on Learning Representations, 2022. 1, 2

[69] Anastasios Zouzias and William F McColl. A parallel scan algorithm in the tensor core unit model. In European Conference on Parallel Processing, pages 489–502. Springer, 2023. 7

A. Applications of orthogonal convolutions

Orthogonal layers have become fundamental components in various deep learning architectures due to their unique mathematical properties, which benefit multiple applications.

Provable Robustness with 1-Lipschitz Networks. Ensuring robustness against adversarial attacks is a critical challenge. Early work in this field [54] identified a link between a network's adversarial robustness and its Lipschitz constant, which led to the development of networks with a Lipschitz constant of one (1-Lipschitz networks). Initially, regularization techniques were used [12], but interest in constrained networks quickly grew. Orthogonal layers, in particular, have an inherent Lipschitz constant of one, as they preserve the input norm through each transformation. This property is instrumental in achieving provable robustness by allowing for certified bounds on the network's output perturbations in response to adversarial inputs [2]. By controlling the network's sensitivity to input changes, orthogonal layers play a crucial role in building models resilient to adversarial manipulations.

Enhancing Performance in Normalizing Flows. Normalizing flows are a class of generative models that transform simple probability distributions into complex ones through a series of invertible and differentiable mappings. Although they have different objectives, this domain intersects with the field of provable adversarial robustness. For instance, [19, 42] employs a channel masking scheme, which was later used to emulate striding in Lipschitz layers designed for adversarial robustness. Separately, Lipschitz layers can be applied to build invertible residual networks [6]. In both fields, orthogonal convolutions are essential, as they facilitate the construction of invertible transformations with tractable Jacobian determinants. The use of orthogonal layers ensures that the Jacobian determinant is constant or easily computable, simplifying likelihood estimation during training [29]. This property enables efficient and stable training of normalizing flow models, leading to improved performance in density estimation and generative tasks.

Stabilizing Training in Deep and Recurrent Neural Networks. Training recurrent neural networks (RNNs) involves propagating gradients through time, which can lead to vanishing or exploding gradients due to the multiplicative nature of sequential weight applications. Orthogonal weight matrices in RNNs help preserve the gradient norm across time steps, thus preventing degradation of the learning signal [27]. By constraining recurrent weights to be orthogonal, the network maintains a consistent flow of information, enabling it to capture long-term dependencies more

effectively. This stabilization is essential for tasks requiring understanding long-range dependencies in time series, such as language modeling and speech recognition. These approaches also facilitate the training of very deep networks [40] with improved generalization properties [5].

Improving Stability in Wasserstein Generative Adversarial Networks. Generative Adversarial Networks (GANs) are powerful models for generating realistic data, but they often suffer from training instability. Wasserstein GANs (WGANs) [4] address this issue by optimizing the Wasserstein distance between the real and generated data distributions. A key requirement for WGANs is that the discriminator (or critic) function must be Lipschitz continuous. Orthogonal layers naturally satisfy this Lipschitz condition, eliminating the need for techniques like weight clipping or gradient penalties [23], which can adversely affect training dynamics. By incorporating orthogonal convolutions into the discriminator, WGANs achieve more stable training [35, 36] and produce higher-quality generative results. Orthogonality has been integral to the successful scaling of GAN training [11].

B. Empirical evaluation of the Lipschitz constant of our method

Evaluating the Lipschitz constant of a network Beyond the creation of a constrained layer, the evaluation of the Lipschitz constant of a layer is by itself an active field: early work used fast Fourier transform to evaluate a lower bound of the Lipschitz constant of a convolutional layer with circular padding [45]. This work was later improved with a method that is quicker [46], supports other types of padding [22], or allows the extraction of a larger part of the spectrum [9]. The work of [15] [16] allows us to compute a certifiable upper bound efficiently under different types of padding. It is worth recalling that inferring the global Lipschitz constant of a network given the Lipschitz constant of each layer is an NP-Hard problem[59]. Then, [20, 37, 62] aim to tackle using SDP (Semi-definite programming) tools. Our work can also contribute to this issue as the orthogonal layer allows a tighter product bound (ie. bound using the product of the Lipschitz constant of each layer to evaluate the constant of the whole network).

The need for an empirical evaluation of the Lipschitz bound of AOC. Despite the theoretical guarantees ensuring orthogonality in our construction, empirical checks are necessary to confirm implementation correctness. Such verification prevents two types of issues:

1. **Checking of numerical Instabilities:** Issues arising from floating-point precision, such as those introduced by small epsilon values added to avoid division by zero.
2. **Checking for implementation Discrepancies:** Differences between mathematical formalism and its translation to popular frameworks (e.g., SOC proofs assume circular padding, while its implementation uses zero padding).

Checking the orthogonality of a layer under stride, group, transposition, and dilation conditions. The numerical stability and the convergence of an orthogonal layer is dependent on the training hyper-parameters: mainly the number of iterations used in most methods, but the learning rate and weight decay can also play a significant role. We then need an evaluation method that scales along with the convolution and that can be used at the end of each training. On the other hand, as scalable methods can be imperfect, we also need a method that computes very precise bounds without making any assumptions on the layer parameters (like padding, or stride). In order to overcome this, we tested our layers with two distinct methods:

1. **Explicit SVD on Toeplitz Matrices:** Using the impulse response approach, we construct the Toeplitz matrix for any padding and stride, allowing direct computation of singular values. This method, though accurate, is computationally expensive for large input images.
2. **Product Bound for BCOP and RKO Kernels:** The upper bound for the BCOP kernel is computed using standard methods, while the SVD of the reshaped RKO kernel is used for direct evaluation.

Unit testing of the implementation. We used both of these two approaches in our unit tests. This enables us to ensure that the second method (which is faster and more scalable) is correct to check that our layer is effectively orthogonal. Also, our layer unlocks the use of the transposed convolution, which can be used to compute directly the equation of orthogonal layers:

$$(\mathcal{S}_s\mathcal{K})(\mathcal{S}_s\mathcal{K})^T = \mathcal{I} \quad \text{(row orthogonal)}$$
$$\texttt{conv}_{\mathbf{K}}(\texttt{ConvTranspose}_{\mathbf{K}}(x, \texttt{stride} = s), \texttt{stride} = s)$$

Naturally, the other direction can also be verified for column orthogonal layers.

To follow the optimization depicted in fig 2c, we tested each branch independently. For each branch, we tested multiple values for kernel size, stride, dilation, input channels, and output channels. For the kernel size, along with standard configurations of 3×3 and 5×5 kernels, we also covered cases for 1×1 kernels and even-sized kernels. For input/output channels, we covered various values and all the inequalities discussed in this paper (for instance when $c_o > c_i s^2$). We ran similar tests for transposed convolution. As the computation of the singular values using the explicit construction of the Toeplitz matrix is quite expensive, we

used it on small 8×8 images, this is also a good way to check for padding issues, as the kernel size is not negligible with respect to the image size. All the checks over the singular values for both methods were done with a tolerance of $1e^{-4}$.

Finally, we tested independently the properties of the block convolution and the batched block convolution.

This amounts to 1442 tests that have the following repartition:

- bloc convolution: 640 tests
- convolution: 418 tests
  - common configurations in CNN: 72 tests
  - extended strided configurations: 150 tests
  - even kernel size: 24 tests
  - depthwise: 24 tests
  - kernel size = stride : 100 tests
- conv transpose: 384 tests
- RKO: 370 tests

This test bank was of precious use to confirm that all parameters can be combined together in practice.

C. Using the content of this paper to improve Cayley, SLL, SOC, and Sandwish

In this section, we will explore how the content of this paper can be used to improve existing layers from the state of the art.

Improving skew orthogonal convolution (SOC)[52] This method uses the fact that an exponential of a skew-symmetric matrix is orthogonal. The initial implementation builds a skew-symmetric kernel and computes the exponential convolution. However, without proper tools to compute the exponential of a convolution kernel, this exponential was computed implicitly for each input by using the Taylor expansion of the exponential (see eq 11).

Theorem C.1 (Explicit conv exponential)**.** *We can use eq 4 to compute explicitly the exponential of a kernel* $\mathbf{K}$*:*

$$x + \frac{\mathbf{K} * x}{1!} + \frac{\mathbf{K} * \mathbf{K} * x}{2!} + \dots \tag{11}$$

$$= \left(Id + \mathbf{K} + \frac{\mathbf{K} \circledast \mathbf{K}}{2!} + \frac{\mathbf{K} \circledast \mathbf{K} \circledast \mathbf{K}}{3!} + \dots \right) * x \tag{12}$$

Equation 12 shows that we can compute the exponential of a convolution kernel a single time, while the formulation 11 needs to be done for each input x. In other words, we can apply one conv instead of n_{iter} convs. Note that the resulting kernel is then larger than the original one (as stated in table 1). In theory, this could unlock large speedups, but the gain is limited in practice as the implementation of convolution layers is optimized for small kernels and large images

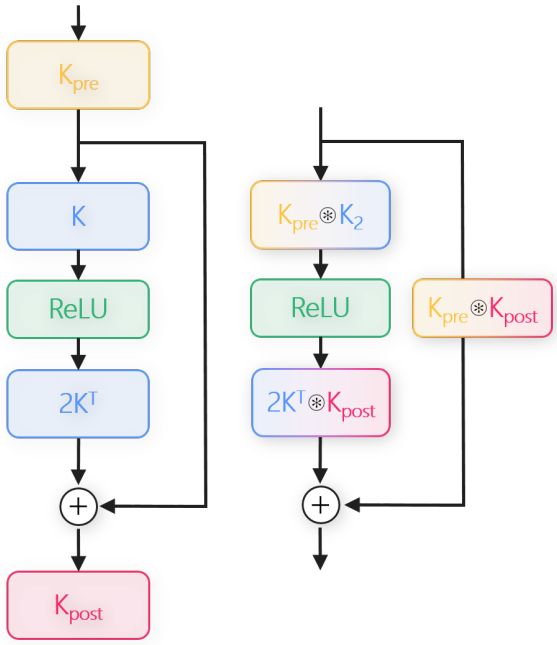


Figure 3. **The $\circledast$ can be used enable $s \neq 1$ and $c_i \neq c_o$ on SLL.** The flexibility of the $\circledast$ allows for operations resulting in a block with a similar structure as the original ResNet block.

[18]. However, the original implementation require the storage of n_{iter} maps, whereas our implementation only one. This, in practice, unlocks larger networks and batch sizes.

Also, it is possible to handle a change in the number of channels and striding using a similar approach as AOC layers.

Improving SDP-based Lipschitz Layers (SLL)[3] SLL layer for convolutions, proposed in [3], is a 1-Lipschitz layer defined as:

$$y = x - 2\mathbf{K}^T \star (\sigma(\mathbf{K} \star x + b))$$

Note that in the original paper, the equation is noted with product of two matrices $WT^{-\frac{1}{2}}$, for convolutions it represents toeplitz matrix, i.e. $WT^{-\frac{1}{2}} = \mathcal{K}$.

SLL layer does not natively support neither strides nor changes in the channel size. We propose to use the $\circledast$ to derive a block, based on SLL, that supports stride and $c_i \neq c_o$, and can replace the strided convolutions of the residual branch in architectures like ResNet.

A natural first step is to append a strided convolution after a SLL block. This layer, $conv_{K_{post}} \circ SLL$, can then be fused in the SLL block thanks to Proposition 2.3:

$$\begin{aligned} y =& \mathbf{K}_{post} \star_s (x - 2\mathbf{K}^T \star (\sigma(\mathbf{K} \star x + b))) \\ =& \mathbf{K}_{post} \star_s x - 2(\mathbf{K}_{post} \circledast \mathbf{K}^T) \star_s (\sigma(\mathbf{K} \star x + b))) \end{aligned}$$

This allows to build a block based on SLL and that supports stride and channel changes. However, this creates an asymmetry between the convolution before the activation and the one after the activation (that has a larger kernel size).

We propose also to add a second convolution before the SLL block, $conv_{K_{post}} \circ SLL \circ conv_{K_{pre}}$ allowing better control over the kernel size of each convolution:

$$\begin{aligned} y =& \mathbf{K}_{post} \star_s \mathbf{K}_{pre} \star x \\ & - 2(\mathbf{K}_{post} \circledast \mathbf{K}^T) \star_s (\sigma(\mathbf{K} \star \mathbf{K}_{pre} \star x + b))) \\ =& (\mathbf{K}_{post} \circledast \mathbf{K}_{pre}) \star_s x \\ & - 2(\mathbf{K}_{post} \circledast \mathbf{K}^T) \star_s (\sigma((\mathbf{K} \circledast \mathbf{K}_{pre}) \star x + b))) \end{aligned}$$

The proposed block is still a 1-Lipschitz layer (as a composition of 1-Lipschitz and orthogonal layers), and support efficiently strides and changes of kernel sizes. A visual description is provided in fig 3. This approach is more efficient than the explicit construction that uses 3 distinct convolutions, as kernels are merged once per batch, and intermediate activations of extra convolutions do not need to be stored backward. Typically, when $\mathbf{K}$, $\mathbf{K}_{pre}$ and $\mathbf{K}_{post}$ are 2×2 convolutions, this results in a residual block with two 3×3 convolutions in one branch and a single 4×4 convolution (with stride 2) in the second. This is very similar to transition blocks found in typical residual networks.

Improving Sandwich Layers. This approach aims to construct a 1-Lipschitz network globally rather than constraining each layer independently. In practice, this can be done either by (i) adding constraints between layers or (ii) creating layers that incorporate a non-linearity internally (a.k.a. sandwich layers). However, sandwich layers require an orthogonal matrix at their core. For convolutional layers, this is achieved by performing the orthogonalization of the layer in the Fourier domain, as described in the method from [55] and shown in their Algorithm 1.

Algorithm 1 Sandwish convolutional layer (from [61])

Require: $h_{\text{in}} \in \mathbb{R}^{p \times s \times s}$, $P \in \mathbb{R}^{(p+q) \times q \times s \times s}$, $d \in \mathbb{R}^q$

$\hat{h}_{\text{in}} \leftarrow \text{FFT}(h_{\text{in}})$

$\Psi \leftarrow \text{diag}(e^d)$, $\begin{bmatrix} \tilde{A} & \tilde{B} \end{bmatrix}^* \leftarrow \text{Cayley}(\text{FFT}(P))$

$\hat{h}[:, i, j] \leftarrow \sqrt{2}\tilde{B}[:, :, i, j]\hat{h}_{\text{in}}[:, i, j]$

$\hat{h} \leftarrow \text{FFT}(\sigma(\text{FFT}^{-1}(\hat{h}) + b))$

$\hat{h}_{\text{out}}[:, i, j] \leftarrow \sqrt{2}\tilde{A}[:, :, i, j]\Psi\hat{h}[:, i, j]$

$h_{\text{out}} \leftarrow \text{FFT}^{-1}(\hat{h}_{\text{out}})$

We can leverage AOC to construct the kernel of an orthogonal convolution, replacing the expensive operation performed in the Fourier domain. Thus, we can construct two kernels, $\mathbf{A}$ and $\mathbf{B}$, with appropriate constraints between the two and apply the rescaling and non-linearity directly in pixel space:

$$h_{\text{out}} = \sqrt{2}\mathbf{A}^\top \star \Psi\sigma\left(\sqrt{2}\Psi^{-1}\mathbf{B} \star h_{\text{in}} + b\right)$$

The use of the Fourier transform is costly for two reasons: first, it necessitates computation with complex values; and second, the cost of the operation depends on the input size, which can be prohibitive in large-scale settings with 224×224 images. Consequently, our approach can make such a layer more scalable.

Extending Applicability to Other Methods. Beyond the previously discussed approaches that show meaningful opportunities for improvement, our method can enhance a wide range of orthogonal convolutional layers. Specifically, we can incorporate our framework into any alternative orthogonal layers, enabling native support for strides in those layers. Furthermore, our approach can unlock features such as grouped convolutions, transposed convolutions, and dilations, broadening its utility and adaptability.

D. About the incomplete parametrization of BCOP and SC-fac

As mentioned earlier, both BCOP and SC-Fac exhibit an incomplete parametrization. BCOP has an incomplete parametrization for 2D convolutions, while SC-Fac offers a complete parametrization but only for separable convolutions.

D.1. Understanding the Limitations of BCOP

The limitations of BCOP parametrization have significant implications for its use in practical applications. Below, we provide a detailed discussion of the known limitations:

- The authors presented a counterexample involving a 2×2 convolution that is orthogonal but cannot be parametrized by BCOP. This highlights the incomplete nature of BCOP for parametrizing certain convolutional layers.
- However, this counterexample can be parametrized by a 3×3 BCOP convolution, which suggests that increasing the kernel size can potentially address the issue of incomplete parametrization.
- This does not imply that all 2×2 orthogonal convolutions can be parametrized using 3×3 BCOP convolutions, but it provides a useful starting point. It indicates that while BCOP may struggle with certain cases at smaller kernel sizes, increasing the kernel size could offer a pathway to improve coverage.

This problem is more complex than initially expected: the parametrization space of BCOP is disconnected. In simple terms, the disconnected nature of the parametrization

space means that certain transformations cannot be continuously reached from others within the same parametrization framework. Nevertheless, a BCOP convolution with c channels can have a connected component that represents all convolutions with $c/2$ channels. This property indicates that, although BCOP may not cover the entire space of orthogonal convolutions, it has subsets that can be effectively utilized for lower-dimensional problems.

Another noteworthy point is that the disconnected nature of the parametrization space could limit the efficiency of optimization algorithms that rely on continuous transformations during training. In practice, this means that certain optimal configurations may not be reachable through gradient-based methods, which could hinder the overall performance of models employing BCOP convolutions.

The issue of incomplete parametrization can be mitigated by increasing the number of channels and kernel size, highlighting the need for a scalable approach to address the challenge effectively. Increasing the number of channels provides more degrees of freedom, which may help cover more of the orthogonal convolution space while increasing the kernel size expands the range of spatial features that can be represented.

Research on the complete parametrization of orthogonal convolutions with controlled kernel sizes remains an open question, AOC could benefit from further improvements in this area.

E. Proofs

E.1. Proof of the $\circledast$ property:

Although already shown in previous papers [31], [64], we provide the proof in the 1D case to help the reader understand the mechanism of the $\circledast$ operator. Given a vector $x \in \mathbb{R}^{c_{in} \times w}$, and two kernels $\mathbf{A} \in \mathbb{R}^{c_{int} \times c_i \times k_A}$ and $\mathbf{B} \in \mathbb{R}^{c_o \times c_{int} \times k_B}$. We suppose that $\mathbf{A}$ is zero-padded, i.e., $\mathbf{A}_{c,n,i} = 0$ if $i < 0$ or $i \geq k_A$.

$$
\begin{aligned}
y &= \mathbf{A} \star x, \text{ such as,} \\
y_{c,j} &= \sum_{n=0}^{c_i-1} \sum_{j'=0}^{k_A-1} \mathbf{A}_{c,n,j'} x_{n,j-j'} \\
z &= \mathbf{B} \star y = (\mathbf{B} \circledast \mathbf{A}) \star x, \text{ such as,} \\
z_{m,l} &= \sum_{c=0}^{c_{int}-1} \sum_{i'=0}^{k_B-1} \mathbf{B}_{m,c,i'} y_{c,l-i'} \\
&= \sum_{c=0}^{c_{int}-1} \sum_{i'=0}^{k_B-1} \mathbf{B}_{m,c,i'} \sum_{n=0}^{c_i-1} \sum_{j'=0}^{k_A-1} \mathbf{A}_{c,k,j'} x_{k,l-i'-j'} \\
&= \sum_{k=0}^{c_i-1} \sum_{j'=0}^{k_A-1} \sum_{c=0}^{c_{int}-1} \sum_{i'=0}^{k_B-1} \mathbf{B}_{m,c,i'} \mathbf{A}_{c,k,j'} x_{k,l-i'-j'} \\
&= \sum_{k=0}^{c_i-1} \sum_{l'=0}^{k_A+k_B-1} \sum_{c=0}^{c_{int}-1} \sum_{i'=0}^{k_B-1} \mathbf{B}_{m,c,i'} \mathbf{A}_{c,k,l'-i'} x_{k,l-l'} \\
&= \sum_{k=0}^{c_i-1} \sum_{l'=0}^{k_A+k_B-1} (\mathbf{B} \circledast \mathbf{A})_{m,k,l'} x_{k,l-l'}
\end{aligned}
$$

with thus

$$
(\mathbf{B} \circledast \mathbf{A})_{m,n,i} = \sum_{c=0}^{c_{int}-1} \sum_{i'=0}^{k_B-1} \mathbf{B}_{m,c,i'} . \mathbf{A}_{c,n,i-i'}
$$

E.2. Proof of the construction of 1x2 kernel

Given the construction explained in section 2, we can show that given a symmetric orthogonal projectors $N \in \mathbb{R}^{c \times c}$ such as:

$$
N = N^2 = N^T \quad \text{and} \quad (I - N) = (I - N)^2 = (I - N)^T
$$

Using N and $I - N$ as $\mathbb{R}^{c \times c \times 1 \times 1}$ convolution kernels, we can build an orthogonal convolution kernel $\mathbf{P} \in \mathbb{R}^{c \times c \times 1 \times 2}$ by.

$$
\mathbf{P} = \texttt{stack}([\mathbf{N}, \mathbf{I} - \mathbf{N}], \texttt{axis} = -1) \Rightarrow \mathbf{P} \circledast \mathbf{P}^T = \mathbf{I}
$$

For readability, we can write $\mathbf{P}$ using a compact notation to describe stacked kernels:

$$
\mathbf{P} = [N, I - N]
$$

Proof: As we assume $s = 1$, we can say that $\mathbf{P}$ is orthogonal if $\mathbf{P} \circledast \mathbf{P}^T = \mathbf{P}^T \circledast \mathbf{P} = \mathbf{I}$. We can then compute

explicitly the resulting kernel for $\mathbf{P} \circledast \mathbf{P}^T$:

$$
\begin{aligned}
&\mathbf{P} \circledast \mathbf{P}^T = \\
&= [N, I-N] \circledast \left[(I-N)^T, N^T\right] \\
&= \left[N(I-N)^T, NN^T + (I-N)(I-N)^T, (I-N)N^T\right] \\
&= \left[N(I-N), N^2 + (I-N)(I-N), (I-N)N\right] \\
&= \left[N - N^2, N^2 + I - 2N + N^2, N - N^2\right] \\
&= \left[N - N, N + I - 2N + 2N^2, N - N\right] \\
&= [0, I, 0]
\end{aligned}
$$

The third line is the application of the $\circledast$ with two 1×2 convolutions. The following lines are based on the trivial application of the symmetric projector property.

Construction of symmetric projectors. Following the construction described in [31] a symmetric projectors $N \in \mathbb{R}^{c \times c}$ can be constructed using by using an orthogonal matrix $N_0 \in \mathbb{R}^{c \times \frac{c}{2}}$:

$$
\begin{aligned}
&\text{given } N_0 \in \mathbb{R}^{c \times \frac{c}{2}}, N_0^T N_0 = I \\
N = N_0 N_0^T \implies\ & N = N^T = N^2 \\
\implies\ & (I-N) = (I-N)^T = (I-N)^2
\end{aligned}
$$

E.3. RKO is orthogonal when $k = s$

The proof is given by the property that when $k = s$ the convolution can be decomposed into a reshaping operation (invertible downsampling with factor s) followed by a 1×1 convolution with kernel $K' \in \mathbb{R}^{c_o \times c_i s^2 \times 1 \times 1}$. The kernel K' is the orthogonal matrix built by RKO method.

E.4. AOC convolutions are orthogonal

The construction of our method consists in composing a BCOP kernel $\mathbf{K}_{BCOP} \in \mathbb{R}^{c \times c_i \times k-s \times k-s}$ followed by an RKO kernel $\mathbf{K}_{RKO} \in \mathbb{R}^{c_o \times c \times s \times s}$. As we already proved that each kernel is orthogonal, we know that

$$
\begin{aligned}
&\mathcal{K}_{BCOP}\, \mathcal{K}_{BCOP}^T \text{ when } c_i \geq c \\
&\mathcal{K}_{BCOP}^T\, \mathcal{K}_{BCOP} \text{ when } c_i \leq c
\end{aligned}
$$

and that

$$
\begin{aligned}
&(\mathcal{S}_s\, \mathcal{K}_{RKO})(\mathcal{S}_s\, \mathcal{K}_{RKO})^T \text{ when } c * s^2 \geq c_o \\
&(\mathcal{S}_s\, \mathcal{K}_{RKO})^T(\mathcal{S}_s\, \mathcal{K}_{RKO}) \text{ when } c * s^2 \leq c_o
\end{aligned}
$$

As c (the intermediate number of channels) is a free parameter we can demonstrate that our construction is orthogonal when the convolutions are either both row orthogonal, or both column orthogonal, i.e when:

$$
\begin{aligned}
&c_i \geq c \text{ and } c \geq \frac{c_o}{s^2} \\
\text{or} \quad &c_i \leq c \text{ and } c \leq \frac{c_o}{s^2}
\end{aligned}
$$

In the first case, when $c_i \geq \frac{c_o}{s^2}$The resulting convolution is orthogonal for any c such as $\frac{c_o}{s^2} \leq c \leq c_i$:

$$
\begin{aligned}
&(\mathcal{S}_s\, \mathcal{K}_{RKO}\, \mathcal{K}_{BCOP})(\mathcal{S}_s\, \mathcal{K}_{RKO}\, \mathcal{K}_{BCOP})^T \\
=&(\mathcal{S}_s\, \mathcal{K}_{RKO})\mathcal{K}_{BCOP}\, \mathcal{K}_{BCOP}^T(\mathcal{S}_s\, \mathcal{K}_{RKO})^T \\
=&(\mathcal{S}_s\, \mathcal{K}_{RKO})(\mathcal{S}_s\, \mathcal{K}_{RKO})^T \\
=&\mathcal{I}
\end{aligned}
$$

The maximum possible value for c is $c = c_i$. The second case when $c_i \leq c \leq \frac{c_o}{s^2}$ can be proven the same way.

E.5. We can apply stride directly on BCOP

When $c_i > c_o$, stride can be directly applied on the BCOP kernel without the need to construct the RKO kernel to handle striding. First we need to show that $\mathcal{S}_s\, \mathcal{S}_s^T = \mathcal{I}$. We can leverage the proof that RKO is orthogonal when $k = s$ and note that RKO can build the identity kernel. Then we can show that $(\mathcal{S}_s\, \mathcal{I})(\mathcal{S}_s\, \mathcal{I})^T = \mathcal{I}$ (def 2.5, given that $c_i < \frac{c_o}{s}$). This proves that $\mathcal{S}_s\, \mathcal{S}_s^T = \mathcal{I}$. However, the other direction is trivially false.

As $\mathcal{S}_s$ is row orthogonal, we can use proposition 2.6 to show that if $\mathcal{K}$ is row orthogonal, then the strided convolution is also row orthogonal. As $\mathcal{K}$ is built using BCOP, we know that it is row orthogonal if $c_i \geq c_o$. The strided version of the convolution using the BCOP kernel is thus also row orthogonal.

F. Architecture and training details

Layer Type	Configuration
Feature Extractor	3 x AOC (128 channels) AOC (256 channels, stride=2) 3 x AOC (256 channels) AOC (512 channels, stride=2) 3 x AOC (512 channels) AOC (1024 channels, stride=2) 4 x AOC (1024 channels) Flatten
Fully Connected	4 x OrthogonalDense (1024 units) OrthogonalDense (nb_classes)

Table 4. Summary of the network architecture used.

CIFAR 10 The CIFAR10 network architecture used in our evaluation is summarized in Table 4. Each block uses circular padding with a kernel size of 3, ensuring consistency in spatial dimensions throughout the network. This design explores the impact of deep, wide layers combined with orthogonality, aiming to maintain expressiveness while ensuring certifiable robustness. We used the ScheduleFree

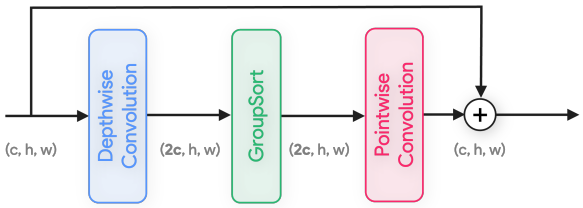


Figure 4. **We can construct complex blocks.** These blocks can reduce the number of parameters of our models, thanks to the flexibility of AOC.

optimizer [14] with a learning rate of 1×10^{-4} and no weight decay.

Layer Type	**Output Shape**	**Config**
Input	[224, 224]	
Convolution Layer 1	[112, 112]	5x5 kernel
Activation Layer 1	[112, 112]	MaxMin
Block 1		
Residual Depthwise Block x 3	[56, 56]	5x5 kernel
Convolution Layer	[28, 28]	3x3 kernel
Block 2		
Residual Depthwise Block x 3	[28, 28]	5x5 kernel
Convolution Layer	[14, 14]	3x3 kernel
Block 3		
Residual Depthwise Block x 3	[14, 14]	5x5 kernel
Convolution Layer	[7, 7]	3x3 kernel
Block 4		
Residual Depthwise Block x 3	[7, 7]	5x5 kernel
L2 Pooling	[1, 1]	3x3 kernel
Flatten	[2048]	
Fully Connected Layer	[1000]	

Table 5. **Summary of our architecture used on Imagenet-1K.**

Imagenet-1K We leveraged the flexibility of our layer to design more complex blocks tailored for ImageNet1K. Each block consists of a depthwise convolution that doubles the number of channels, followed by a MaxMin activation function [2] to enhance non-linearity, and a pointwise convolution that reduces the number of channels. These layers are encapsulated within a skip connection featuring a learnable factor to ensure a Lipschitz constant of 1, represented by $y = \alpha x + (1-\alpha) f(x)$ (where $\alpha \in [0,1]$). A visual description can be found in Fig. 4. The entire architecture consists of a sequence of such blocks interleaved with strided convolutions, an exhaustive description of the architecture is provided in table 5.

G. Reproducing the results from BCOP paper

Models	Accuracy	Provable Accuracy $\epsilon = \frac{36}{255}$
BCOP - Small net (original seeting)	72.2	58.26
AOC - Small net (all optimizations)	62.3	49.18
AOC - Small net (opt 2c removed)	71.8	58.25
AOC - Large net (all optimizations)	74.0	64.33

Table 6. **Mitigating AOC limitations in small scale setting**: as AOC uses less parameters for strided convolutions, this can impact its expressive power in small scale setting. However, by removing the optimization 2c this increase the number of parameters enough to correct this issue. Results from table 3 added for reference.

As mentioned earlier, AOC is built on top of components like BCOP and RKO. Hence, any accuracy gain must come from the fact that our implementation allows larger networks and more training steps within the same compute budget. To illustrate this, we reproduced the baseline results from [31] and switched the implementation to ours. Results are shown in table 6. We can observe a notable difference in performance between the original implementation and ours. This is due to the difference in the parametrization of strided convolutions: the original paper uses invertible downsampling to emulate striding followed by a standard convolution whose number of input channels is multiplied by s^2, such convolution has a $s^2\times$ more parameters than the proposed AOC strided convolution. This is accentuated by the specific small architecture used in BCOP paper (and reproduced in this experiment) which has 84% of the convolutional layers parameters that are located in strided convolutions. Also, invertible down-sampling has a 2×2 receptive field, which increases the global convolution's receptive field. However, when we remove the optimization depicted in fig. 2c and only resort to the parametrization described in fig. 1, the increase in the number of parameters results in improved results close to the results of the original paper. This observation is non-trivial since this modification is not equivalent to the original implementation: instead of parametrizing a $c_o \times 4\, c_i \times k \times k$ AOC parameterized one $\max(c_i, \frac{c_o}{s^2}) \times c_i \times k - s + 1 \times k - s + 1$ and one $c_o \times \max(c_i, \frac{c_o}{s^2}) \times s \times s$ kernels which are much smaller.

We recall that all the results of this paper were obtained with the optimization in Fig. 2c. We evaluated the unoptimized version in the same context as in table 2 and found, for a batch size of 512, a slowdown of $1.21\times$ in train time (instead of $1.13\times$) and no notable increase in train memory consumption.

Finally, it is worth noting that the optimization depicted in Fig. 2c should not affect the expressive power if we were

able to parameterize the complete set of orthogonal convolutions.