Convolutional Networks with Oriented 1D Kernels

Alexandre Kirchmeyer *
Carnegie Mellon University
akirchme@cs.cmu.edu

Jia Deng
Princeton University
jiadeng@princeton.edu

Abstract

In computer vision, 2D convolution is arguably the most important operation performed by a ConvNet. Unsurprisingly, it has been the focus of intense software and hardware optimization and enjoys highly efficient implementations. In this work, we ask an intriguing question: can we make a ConvNet work without 2D convolutions? Surprisingly, we find that the answer is yes—we show that a ConvNet consisting entirely of 1D convolutions *can do just as well as 2D on ImageNet classification. Specifically, we find that one key ingredient to a high-performing 1D ConvNet is* oriented 1D kernels*: 1D kernels that are oriented not just horizontally or vertically, but also at other angles. Our experiments show that oriented 1D convolutions can not only replace 2D convolutions but also augment existing architectures with large kernels, leading to improved accuracy with minimal FLOPs increase. A key contribution of this work is a highly-optimized custom CUDA implementation of oriented 1D kernels, specialized to the depthwise convolution setting. Our benchmarks demonstrate that our custom CUDA implementation almost perfectly realizes the theoretical advantage of 1D convolution: it is faster than a native* horizontal *convolution for any* arbitrary angle. *Code is available at https://github.com/princeton-vl/Oriented1D.*

1. Introduction

Convolutional Networks (ConvNets) [27, 25] are widely used in computer vision. They have been successfully applied to a variety of tasks [15, 40, 16, 63] and domains [25, 34, 23, 39, 59], and many new ConvNet-based building blocks [47, 8, 57] and design practices [44, 50, 32] have emerged over the years.

In the computer vision context, a 2D convolution is arguably the most important operation performed by a ConvNet. In virtually all ConvNet architectures [25, 18, 32], 2D convolution is the default choice and accounts for the bulk of the computation. Unsurprisingly, 2D convolution has been the focus of intense software and hardware optimization and enjoys highly efficient implementations.

In this work, we ask an intriguing question: can we make a ConvNet work without 2D convolution? Surprisingly, we find that the answer is yes—we show that a ConvNet consisting entirely of *1D convolutions* can do as well on ImageNet classification, a surprising result given that 2D convolution has been the go-to design choice.

Specifically, we find that one key ingredient to a high-performing 1D ConvNet is *oriented 1D kernels*: 1D kernels that are oriented not just horizontally or vertically, but also at other angles. This is a novel finding—although horizontal and vertical 1D convolutions have been frequently used in the past, 1D kernels oriented at arbitrary angles have not been well studied.

Oriented 1D kernels are motivated by the fact that 2D kernels can be approximated by 1D kernels, which are more efficient computationally. In particular, it is well known that convolution with a separable 2D kernel (i.e. rank 1 as a matrix) is equivalent to consecutive convolutions with a vertical 1D kernel and a horizontal 1D kernel, leading to significant efficiency gain. However, in practice, not all learned 2D kernels are rank 1; if we only use vertical and horizontal 1D kernels, 2D kernels with a full rank, such as diagonal matrices, are poorly approximated, which can lead to a loss in accuracy: for example, the network may be less able to detect a 45° edge. This is when oriented 1D kernels can be helpful. By orienting a 1D kernel at more angles, we expand the set of 2D kernels that can be well approximated by 1D kernels while retaining the efficiency of 1D kernels.

Oriented 1D kernels are also motivated by the increasing use of large 2D kernels in recent convolutional architectures. Large 2D kernels improve the modeling of long-range dependencies, which have been shown to result in better accuracy [32, 12, 30]. However, large 2D kernels are significantly more expensive because the cost scales quadratically. A 31×31 kernel is 19 times more costly in terms of multiply-add operations than the standard 7×7 kernels. This motivates oriented 1D kernels as an alternative for modeling long-range dependencies, because the cost of 1D kernels scale only linearly with the kernel size.

The concept of oriented 1D kernels is simple, but nontrivial to implement in a way that realizes its efficiency advantage over 2D kernels. This is because on a GPU, the pat-

*Work done during an internship at Princeton University.

tern of memory access matters as much as, if not more than, the number of floating point operations. Applying a 1D kernel oriented at an arbitrary angle requires accessing non-contiguous data; a naive implementation can easily negate the theoretical advantage of 1D kernels due to poor management of memory access. In addition, it is important for the implementation to not introduce significant memory overhead, which could be incurred by some naive implementations. Note that while horizontal/vertical 1D convolutions are well supported and optimized by existing libraries, 1D convolutions at an arbitrary angle is not.

A key contribution of this work is a highly-optimized custom CUDA implementation of oriented 1D kernels, specialized to the setting of depthwise convolution [5], where each kernel is applied to only one depth channel. We optimize for depthwise convolution, because it has become an essential building block in recent state-of-art architectures [51, 32], with superior accuracy-efficiency trade-offs. In addition, we find depthwise convolution to be the more useful setting in our experiments. Experiments show that our custom CUDA implementation almost perfectly realizes the theoretical advantage of 1D convolution: our 1D convolution at an *arbitrary angle* is faster than the native *horizontal* 1D convolution in PyTorch, which is highly optimized and achieves over 96% of the theoretical speedup over 2D convolution. Notably, our implementation incurs minimal memory overhead; it uses less than 5% more GPU memory than the native horizontal 1D convolution in PyTorch. Our implementation is open-sourced as a plug-and-play PyTorch module at `https://github.com/princeton-vl/Oriented1D`.

With our custom implementation, our experiments show that oriented 1D convolution can not only replace 2D convolution but also augment existing architectures with large kernels, leading to improved accuracy with minimal FLOPs increase. We expect our implementation to be a useful primitive for further innovation in neural architectures for computer vision.

Our main contributions are two-fold. First, we present the novel finding that state-of-the-art accuracy can be achieved with oriented 1D convolution alone, and that oriented 1D convolution can be a useful primitive to augment existing architectures. Second, we introduce an optimized CUDA implementation of depthwise oriented 1D convolution that nearly maxes out the theoretical efficiency of 1D convolution.

2. Related Work

Modern ConvNets. In recent years we have witnessed the emergence of novel training techniques [32, 50, 51] and block designs [5, 20, 44] inspired by transformers [32] and neural architecture search [50]. Depthwise convolutions [5] have become essential components of many ConvNets [44, 50, 32] for their superior accuracy/computation trade-off. In search of better accuracy/computation trade-offs, recent research has looked at better scaling [51, 57], fused operations [51, 12], neural architectural search [50] and sparsity [30] amongst others. Our work fits into this search for better efficiency by looking at 1D kernels that have the potential to scale better with kernel size.

1D convolutions. The use of 1D kernels has previously been explored in the context of image representation learning by [38, 48, 46, 3]. These approaches rely primarily on decomposing a 2D convolution into a horizontal and vertical convolution [41], and are applied in ConvNets in a parallel [38, 16] or stacked design [46]. However the use of 1D kernels often results in a drop in accuracy [3]. Some approaches explore how to overcome this drawback, through SVD decomposition [10], or L2 reconstruction loss [24]. Our approach tries to address this problem differently: we generalize 1D kernels to allow for non-horizontal and non-vertical kernels, whilst preserving the linear cost and advantages of 1D kernels. A recent work [28] has explored a similar idea using diagonal kernels, but they only studied *replacing* horizontal/vertical kernels with diagonal/anti-diagonal kernels, and reported better accuracy than horizontal/vertical kernels but worse accuracy than the 2D baseline. In contrast, we study 1D kernels oriented at more varied angles *including* horizontal and vertical, while achieving no worse accuracy than the 2D baseline. Moreover, their work implements diagonal/anti-diagonal kernels by masking 2D kernels, thus gaining no efficiency advantage over 2D kernels, whereas we provide an optimized implementation that achieves near-perfect speedup.

Oriented kernels. The use of oriented kernels is not new and dates back at least to steerable filters [14]. A commonly used approach is to learn a decomposition of a kernel on a basis which supports rotation, like harmonic functions [54], wavelet transforms [36] or steerable filters[53]. The end goal is often to make the networks invariant or equivariant to rotation, and other transformations [6, 53]. Our approach is simpler than methods using external kernel functions. This allows us to design fast implementations that can be integrated easily into existing architectures. Contrary to group equivariant [6] methods, we do not attempt to design rotation-equivariant networks but only to expand the expressiveness of 1D kernels with arbitrary orientations.

Large kernel design. The emergence of ConvNets benefitting from large kernels [32] has defied the long-established and implied superiority of small kernels since first initiated by VGGNet [45]. This has led to a wave of research exploring new ways to improve performance with bigger kernel sizes. RepLKNet [12] achieves better performance with 31×31 kernels by integrating a small 3×3 or 5×5 kernel that compensates for the difficulty in learning large kernels. SlaK [30] pushes this principle further with rectangular 5×51 and 51×5 kernels combined with dynamic sparsity. Other lines of research include dilated convolutions [58], deformable convolutions [8, 66] and continuous convolutions [43, 42]. Our approach differentiates itself from these past works by studying large oriented 1D kernels, a type of large kernels that offers unique efficiency advantages but have not been studied in the context of modern ConvNet architectures.

3. Oriented 1D kernels

In this section we introduce oriented 1D kernels for depthwise convolution, provide a computational analysis to justify their use and design models to test their effectiveness

3.1. Definition

An oriented kernel is a kernel which is not always applied along the horizontal or vertical axis but along any axis as specified by an angle/direction θ. We illustrate this concept in Figure 1. For more expressivity, we allow θ to change at every channel, and call this per-channel angle θ_c. Definition 1 translates this intuition into a mathematical formulation. A justification can be found in the supplementary material.

Definition 1 (Depthwise convolution of oriented 1D kernel) *Let* $\mathbf{x} \in \mathbb{R}^{N\times H\times W\times C}$ *be the input of a depthwise convolution by an oriented 1D kernel of weights* $\mathbf{w} \in \mathbb{R}^{K\times C}$ *and per-channel angles* $\boldsymbol{\theta} \in \mathbb{R}^{C}$*. We define the output* $\mathbf{y} \in \mathbb{R}^{N\times P\times Q\times C}$ *as:* $\forall n \in [\![0, N-1]\!], p \in [\![0, P-1]\!], q \in [\![0, Q-1]\!], c \in [\![0, C-1]\!]$,

$$\mathbf{y}_{npqc} = \sum_{k=0}^{K-1} \mathbf{x}_{nhwc}\mathbf{w}_{kc} \tag{1}$$

$$\text{where} \begin{cases} h = str \cdot p + \lfloor -(k - pad) \cdot \sin \boldsymbol{\theta}_c \rfloor \\ w = str \cdot q + \lfloor\ (k - pad) \cdot \cos \boldsymbol{\theta}_c \rfloor \end{cases} \tag{2}$$

N is the batch size, C is the number of input channels, K is the 1D kernel size, H, W and P, Q are input and output dimensions, str is the stride and pad is the padding.

The use of rounding in Equation (2) is necessary because of the discrete nature of input and filter coordinates. As an alternative to rounding down the filter offsets, bilinear interpolation could be considered. However in our experience, interpolation increases the computational cost substantially, making it impractical. See Supplementary material for more details.

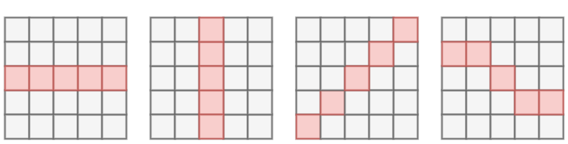

Figure 1: Examples of oriented 1D kernels

In this paper we only consider fixed $\boldsymbol{\theta}$, meaning that $\boldsymbol{\theta}$ is not learnt. Instead, we pick a fixed number D of angles $0, \frac{1}{D}180°, \frac{2}{D}180°..., \frac{D-1}{D}180°$ and partition the channels into D equal groups such that angle i is associated to group i. In other words, for every channel of group i, it is assigned to angle $\frac{i}{D}180°$. In practice, we consider $D \in \{2, 4, 8\}$ and the case $D = C$. For $D = 4$ and $C = 512$, this means that there are 4 groups of 128 channels, with respective angles $0, 45°, 90°, 135°$. We will call D the number of directions as it is more visual.

3.2. Computational cost analysis

Depthwise convolutions are nearly always used in conjunction with pointwise convolutions with normalizations [2, 22] and/or non-linearities [1, 19] added in between. The idea is that depthwise convolutions mix spatial information and pointwise convolutions mix channel information [17]. The combination of both is called a depthwise separable convolution (DSC) [3, 44] and allows depthwise convolutions to be used as 2D convolution approximators.

In this subsection, we show that switching to oriented 1D kernels leads to substantial speedups in computing both depthwise convolutions and DSCs.

To demonstrate this, notice that a DSC with C' output channels is the mix of a depthwise convolution and of a pointwise convolution, with respective computational costs of $NCHWK^2$ and $NCHWC'$ Multiply-Adds (MADs). Normalizing by the input size $NCHW$, a DSC induces a cost of:

$$\frac{NCHWK^2 + NCHWC'}{NCHW} = K^2 + C'$$

By replacing the 2D kernel with a 1D kernel, the expression reduces to:

$$\frac{NCHWK + NCHWC'}{NCHW} = K + C'$$

According to our benchmarks Table 1 on an NVIDIA RTX 3090 with Pytorch 1.11, pointwise convolutions are at least 20× more efficient than depthwise convolutions. As such, depthwise convolutions account for more than 50% of the workload when $K^2 \geq C'/20$. For typical ConvNets, $C' \sim 10^3$, therefore the expression simplifies to $K \approx 7$, which is the kernel size used by ConvNeXt. This implies that we can expect a significant gain by replacing 2D kernels by 1D kernels on a network like ConvNeXt. This observation provides the basis for our study of oriented 1D kernels and motivates our search for fast implementations.

Table 1 provides a summary of the theoretical and practical speedups that we get from switching to oriented 1D kernels. First, we observe that replacing the 2D 7×7 kernel with the 1D 1×31 kernel reduces MADs by 58% even though we also switch to a larger kernel size. Second, we see that 1D kernels can be as efficient as 2D kernels: on PyTorch, switching to 1D leads to a 53% practical runtime

Variant	MADs	Kernel Size K	Inference Runtime	Training Runtime	MADs Ratio	Runtime Ratio	Efficiency	Memory Usage
Convolution (PyTorch)	$C'K^2$	7	34.0±0.2ms	78.4±0.3ms				9.7G
Depthwise 2D (PyTorch)	K^2	7	8.3±0.1ms	22.8±0.1ms	0.096	2.3	24.3	0.77G
Pointwise (PyTorch)	C'	1	4.2±0.1ms	9.8±0.1ms				9.7G
Depthwise 2D (PyTorch)	K^2	7	8.3±0.1ms	22.8±0.1ms	1.58	1.53	0.97	0.77G
Depthwise 1D (PyTorch)	K	31	5.2±0.1ms	14.9±0.3ms				0.77G
Depthwise 1D (PyTorch)	K	31	5.2±0.1ms	14.9±0.3ms	1	1.51	1.51	0.77G
Oriented Depthwise 1D (Ours)	K	31	**4.2±0.1ms**[1]	**9.9±0.2ms**[1]				0.77G
2D Depthwise Separable Conv. (DSC)	$K^2 + C'$	7	12.7±0.1ms	37.4±0.2ms	1.03	1.56	1.49	2.3G
Oriented 1D DSC (Ours)	$K + C'$	31	**8.4±0.1ms**	**23.8m±0.2ms**				2.3G

Table 1: **Pairwise comparisons of the practical and theoretical performances of 2D and 1D convolutions.** Oriented 1D kernels lead to significant speedups in theory and practice. We benchmark on a 56×56 input with FP32 on an NVIDIA RTX3090, against PyTorch[37] 1.11/CuDNN 8.2[4] with $N = 64$ and $C = C' = 512$. We compute the mean and standard deviation for 100 runs, preceded by 10 dry runs. *MADs* (Multiply-Adds) are divided by the input size $NCHW$ for better readability. *Inference Runtime* measures only forward pass, *Training Runtime* includes backpropagation. *Memory Usage* measures peak memory usage difference before/after. Here, *Efficiency* means *Runtime Ratio* (practical gains) divided by *MADs Ratio* (theoretical gains), and measures how efficient an implementation is compared to theory (higher is better). For [1], we aggregate runs over all integer angles $0°, ..., 359°$ to demonstrate that our implementation is fast for all angles.

improvement which closely matches the theoretical 58% MAD improvement. Third, our implementation is more efficient than PyTorch: even though it supports oriented kernels, it is up to 51% faster for all angles. Finally, we demonstrate that oriented 1D leads to significant overall speedups, as replacing the 2D 7×7 DSC with the oriented 1D $K = 31$ DSC reduces runtime as a whole by 35%.

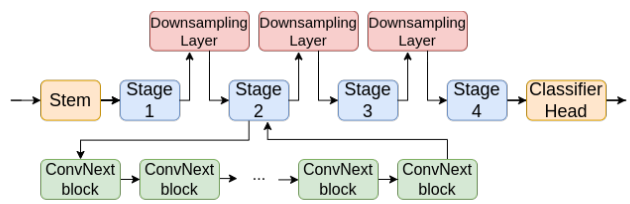

Figure 2: Illustration of ConvNeXt which acts as the base architecture for all of our models.

4. Model Instantiation

To demonstrate the effectiveness of oriented 1D kernels, we gradually improve a ConvNeXt [32] architecture with 1D kernels. We propose 3 different models: *ConvNeXt1D*, *ConvNeXt1D++* and *ConvNeXt2D++* to show that oriented 1D ConvNets can be made as accurate as 2D ConvNets, and that we can use oriented 1D kernels to improve the accuracy of existing ConvNets. We structure our findings as follows: 1) *Transition from 2D to 1D*, 2) *1D-augmented design*, 2) *1D/2D-mixed design*.

4.1. ConvNeXt reference

Our models all use ConvNeXt [32] depicted in Figure 2 as base architecture, which can be decomposed into:

1. a *stem layer*, which downsamples an input image $4\times$ before feeding it to the rest of the network. It constitutes what we call the *stem design*.
2. 4 *stages* each composed of a series of ConvNext blocks as defined by the *block design*. They are responsible for the bulk of the ConvNet and are typically very deep[18].
3. A *downsampling layer* separating each *stage*, inducing a hierarchical structure that defines ConvNets [45, 18].

We define the "*2D Block* design" and "*2D Stem* design" after ConvNeXt's block and stem designs and present them in Figure 3a and Figure 3d. In short, the "*2D Block* design" used by ConvNeXt is an inverted bottleneck with depthwise convolutions [5], as proposed initially by MobileNetV2 [44]. It integrates GeLU [19] and LayerNorm [2], following similar adoption in transformers [11, 31]. ConvNeXt uses an aggressive downsampling strategy for its stem layer in the form of a 4×4 stride 4 convolution. ConvNeXt further innovates by using an even-sized 2×2 stride 2 convolution. Even-sized kernels tend to degrade performance because they introduce asymmetric padding [62, 55]. In our experience, modifying this layer leads to a drop in accuracy, which aligns with observations made in [32]. Consequently, we keep the original downsampling layer in our models.

In the following, we consider the ConvNeXt architecture and progressively improve the stem design and block design, using oriented 1D kernels. Table 2 presents a summary of our models and Figure 3 the designs that we use.

Model	Stem Design	Block Design
ConvNeXt	2D	2D
ConvNeXt1D	Depthwise 1D	1D
ConvNeXt1D++	Depthwise 1D	1D++
ConvNeXt2D++	2D	2D++

Table 2: **Model Summary.** We use ConvNeXt as baseline and propose **ConvNeXt1D**, a fully 1D ConvNet, **ConvNeXt1D++**, a 1D-augmented ConvNet, **ConvNeXt2D++**, a mixed 1D/2D ConvNet.

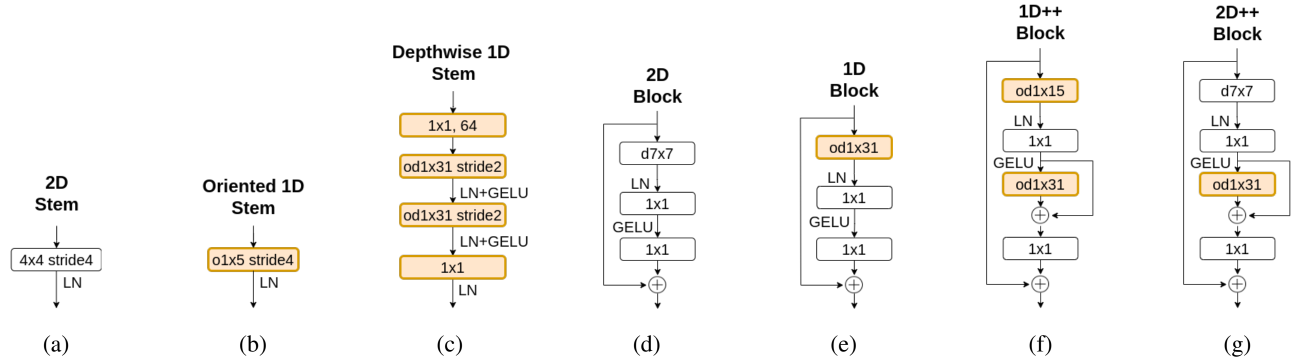

Figure 3: **Stem and block designs** used in our models. The 2D **ConvNeXt** baseline is a combination of **(a)+(d)**, our fully 1D **ConvNeXt1D** network is a mix of **(c)+(e)**, our 1D-augmented **ConvNeXt1D++** model is composed of **(c)+(f)** and **ConvNeXt2D++**, our mixed 1D/2D network, by **(a)+(g)**. Stem **(b)** is used as baseline for our ablation Table 6. *o* stands for oriented, *d* for depthwise.

4.2. Transition from 2D to 1D

Oriented 1D network. As a starting point, we construct a simple 1D baseline which we progressively improve to get *ConvNeXt1D*. It is obtained by taking a ConvNeXt architecture and doing a one-on-one translation of every 2D kernel to an oriented 1D kernel. The network is defined through its "*1D Stem* design" and "*1D Block* design" as shown in Figure 3b and Figure 3e, which are just the oriented 1D equivalents of the *2D Stem/Block* designs. Notice that the *1D Stem* is a 1×5 kernel, which is done to avoid even-sized kernels [55]. We use this network to evaluate our oriented 1D models in Table 6.

Depthwise oriented 1D Stem Design. Naively transitioning from 2D to 1D as defined above naturally leads to performance degradation as the number of parameters in the stem layer is reduced by more than 65%. To account for this, we propose a new stem design better suited for 1D networks, dubbed the *Depthwise 1D Stem*. It is inspired by RepLKNet [12] and is shown in Figure 3c. *Depthwise 1D Stem* is a series of depthwise and pointwise convolutions: the stem alternates between expanding and mixing channels, or downsampling the input and mixing spatial information. For this stem design, we introduce the parameter C_0, which represents the number of channels by which we initially expand the input. In ablation Table 9 we show that choosing C_0 large enough is necessary to preserve the accuracy of the model. This suggests that the number of spatial parameters is a key component in a ConvNet's accuracy.

Large kernels. Oriented 1D kernels are very advantageous from a long-range modeling perspective. Because their cost is linear in K, kernels of size 1×31 are cheaper than 2D kernels of size 7×7. The interest of having such large Ks is that the network is able to achieve global scale convolutions very early/deep in the network, in fact as soon as the 2[nd] stage. This stems from the observation that the input to stage 2 will be of size 28×28 because of the downsampling operated at the stem layer and end of stage 1. As $31 > 28$, this means that K is bigger than the input size, allowing it to encode all long-range dependencies at stage 2 and use the extra depth to model more complex spatial interactions. Note that we limit per-stage K to $[31, 31, 27, 15]$ or twice the stage input size, to avoid uninitialized weights.

Layer-wise rotation. We can improve the spatial mixing of oriented 1D kernels by adding an angular shift at every layer. We call this *layer-wise rotation*. By adding a layer-wise rotation of 90° at every layer, we can reproduce a horizontal kernel in one layer and a vertical kernel in the next layer, effectively approximating a 2D kernel in a 2-layer setup. We integrate this idea in all of our models to improve performance, as validated by ablation Table 7.

Downsampling layer. We claim that we can make a 2D network fully 1D without actually changing 2×2 downsampling layers. This is based on the observation that a 2×2 kernel can be seen as the sum of a diagonal and anti-diagonal kernel, which are special cases of oriented 1D kernels. See more details in the supplemental material.

ConvNeXt1D. We can now combine all of these findings to define our fully oriented 1D network *ConvNeXt1D*. It is constructed on top of ConvNeXt with the *Depthwise 1D Stem* and *1D Block* defined above, and uses a large kernel of size $K = 31$ with $D = 8$ directions, as suggested by our ablation study Table 8. We show in Section 5 that *ConvNeXt1D* is able to perform on par with ConvNeXt.

4.3. 1D-Augmented Block Design

With *ConvNeXt1D*, we have presented a 1D network that can compete with existing 2D networks. We now provide evidence that oriented 1D kernels can also improve the accuracy/computation trade-off of ConvNets by boosting performance for a negligible increase in FLOPS/parameters.

For that purpose, we propose the *ConvNeXt1D++* model, constructed on top of *ConvNeXt1D*. *ConvNeXt1D++* reuses the *Depthwise 1D Stem* that defines

ConvNeXt1D and introduces the improved *1D++ Block* design which builds upon *1D Block* and is presented in Figure 3f. The idea is to reduce $K = 31$ to $K = 15$ and insert into the *1D Block* a depthwise convolution of a large oriented 1×31 kernel. This depthwise convolution is added residually to the network. We specifically target the *inverted bottleneck* layer as shown in Figure 3f, where we expect long-range dependencies to help the most. The goal is to integrate global scale interactions into local representations. With this design, we obtain better performance compared to *ConvNeXt*, even though the extra FLOPS/parameters represent less than 5% of the total. See Section 5 for more results.

4.4. 1D/2D-Mixed Block Design

We now demonstrate the flexibility of our method by proposing the *ConvNext2D++* model. This model is constructed by extending the 2D ConvNext architecture with a new block design, the *2D++ Block*, as shown in Figure 3g. The *2D++ Block* design is constructed on top of the *ConvNeXt Block* and adds to it the same 1D augmentations that were included in the *1D++ Block*. Through this process, we show that oriented 1D kernels can be dropped in, *as is*, in existing architectures to improve long-range modeling and accuracy. Similarly as for *ConvNeXt1D++*, we get better performances than *ConvNeXt*, demonstrating that our approach is general and can be applied successfully to state-of-the-art models.

5. Experiments

In this section, we present experimental results conducted on ImageNet [9] image classification, and downstream taks including ADE20K [65] semantic segmentation and COCO [29] object detection. We summarize our training setup below and present details in supplementary material.

ImageNet Training. We use the same training settings as ConvNeXt [32] to train our models. In summary, we train for 300 epochs with 20 epochs of linear warm-up, using the AdamW optimizer [33], data augmentations and regularizations like RandAugment [7], Mixup [61], Random Erasing [64], Cutmix [60], Stochastic Depth [21] and Label Smoothing [49]. The learning rate follows a cosine decay schedule and is set initially to 0.004. We use a batch size of 4096 and weight decay of 0.05.

5.1. Training Setup

Model sizes. We evaluate our models using the Tiny and Base complexities defined by ConvNeXt [32], which are parameterized by the channel sizes $C = (C_1, C_2, C_3, C_4)$ and numbers of blocks $B = (B_1, B_2, B_3, B_4)$ for all 4 stages. For our models using the *Depthwise 1D Stem* design, we also introduce the number of channels C_0 by which the stem layer initially expands the input, as described in Section 4.2. We name our model variants respectively *ConvNeXt-T/B-1D, ConvNeXt-T/B-1D++* and *ConvNeXt-T/B-2D++* and aggregate the configurations in Table 4.

Size	C_0	C_1	C_2	C_3	C_4	B_1	B_2	B_3	B_4
Tiny/T	64	96	192	384	768	3	3	9	3
Base/B	64	128	256	512	1024	3	3	27	3

Table 4: Model configurations, taken directly from ConvNeXt [32]

5.2. Experimental Results

Image classification. Table 3 compares our models against state-of-the-art ConvNets like ConvNeXt [32], Re-

	Image classification				Semantic segmentation			Object detection				Throughput (img/s)	
Model	#Params	FLOPs	Acc	EMA Acc	#Params	FLOPs	mIoU	#Params	FLOPs	AP^{box}	AP^{mask}	Ours	PyTorch
ConvNext-T [32]	28.6M	4.5G		82.1	60M	939G	46.7	86M	741G	50.4	43.7		1020
SlaK-T [30]	30M	5.0G		82.5	65M	936G	47.6†			**51.3**	44.3		450
ConvNext-T (reprod.)	28.6M	4.5G	81.8	82.0	60M	939G	46.6	86M	741G	50.2	43.6	/	1020
ConvNext-T-1D	28.5M	4.4G	82.2	82.2	60M	927G	45.2	86M	739G	50.3	43.6	900	800
ConvNext-T-1D++	29.2M	4.7G	**82.4**	**82.7**	61M	927G	47.4	86M	739G	**51.3**	**44.5**	630	390
ConvNext-T-2D++	29.2M	4.8G	82.3	82.5	61M	939G	**48.1**	87M	741G	51.2	44.4	670	410
ConvNext-B [32]	89M	15.4G		83.8	122M	1170G	49.9	146M	964G	52.7	45.6		460
RepLKNet-31B [12]	79M	15.3G		83.5	112M	1170G	50.6	137M	965G	52.2	45.2		340
SlaK-B [30]	95M	17.1G		**84.0**	135M	1172G	50.2†						210
ConvNext-B (reprod.)	89M	15.4G	**83.6**	83.7	122M	1170G	49.4	146M	964G	52.4	45.2	/	460
ConvNext-B-1D	88M	15.3G	**83.6**	83.8	122M	1150G	49.4	145M	960G	52.8	45.7	450	430
ConvNext-B-1D++	90M	15.8G	83.5	83.8	123M	1149G	**50.7**	147M	960G	**52.9**	**46.0**	290	200
ConvNext-B-2D++	91M	15.9G	**83.6**	**84.0**	124M	1170G	50.2	148M	964G	**52.9**	45.8	290	200

Table 3: **Experimental results on ImageNet image classification, ADE20K semantic segmentation using UperNet, and COCO object detection using Cascade Mask R-CNN.** Overall, our models are competitive with ConvNeXt and state-of-the-art ConvNets on all tasks. Note that for downstream tasks, we use non-EMA backbones which means that performance improvements are not attributable to increased classification accuracy. Backbones are pre-trained on ImageNet-1K. We report image classification results with and without Exponential Moving Average (EMA). † SlaK only provides single-scale segmentation results. Throughput measures the inference speed of our proposed models on 1 NVIDIA RTX3090, *PyTorch* substitutes oriented kernels with competitive PyTorch horizontal kernels. FLOPs are computed on input sizes 224^2, 1280×800 and 2048×512 respectively.

pLKNet [12] and SlaK[30]. We see that both *ConvNeXt1D* and *ConvNeXt1D++* Tiny and Base perform on par with ConvNeXt, which shows that oriented 1D convolutions are able to perform just as well as 2D convolutions. Additionally, *ConvNeXt-T/B-2D++* are able to beat ConvNeXt-T/B by +0.5%/+0.3% which demonstrates that oriented 1D kernels are versatile as stand-alone networks and as extensions to existing networks.

Semantic segmentation. We fine-tune UperNet [56] on the ADE20K [65] dataset. We follow the same setup as ConvNeXt, that is we preserve all parameters apart from the backbone. Similarly as ConvNeXt, we use model non-EMA weights and multi-scale training for 160k iterations with a 512 crop size. We present our semantic segmentation results in Table 3. Overall, ConvNeXt-1D *Tiny* lags behind ConvNeXt but *Base* catches up to it. The augmented networks achieve higher mIoU than ConvNeXt-T/B by at least +0.8 / +0.8 mIoU. Note that for *Base* models, non-EMA image classification accuracy does not change: because our networks do not use EMA backbones, this means that the IoU improvement is not attributable to better image classification, but to the choice of architecture, thereby validating the effectiveness of oriented kernels on *Base* models.

Object detection. We follow ConvNeXt and fine-tune a Cascade Mask R-CNN [26] on the COCO dataset using multi-scale training and a 3× schedule. As shown in Table 3, ConvNeXt-1D achieves similar or better performances compared to ConvNeXt. The results confirm the expressiveness of augmented networks: they have +0.5 AP^{box} and +0.6 AP^{mask} compared to both RepLKNet[12] and ConvNeXt.

Inference speed metrics. We complement our classification results with inference speeds on a 224×224 input, batch size 64, and 1 NVIDIA RTX3090, as presented in Table 3. Even though 1D kernels are faster than 2D and the FLOPs increase is negligible, our ConvNeXt models are slower because of the modified Stem and Block designs. This suggests a model-level under-utilization due to memory related bottlenecks, which we will study as future work. To confirm our computational analysis, we replace oriented kernels with PyTorch 1D kernels. Note that we fuse certain 1D operations which improves 1D speed over 2D.

Additional architectures. To provide evidence that our 1D convolutions are generalizable, we compare the 2D ConvNets MobileNetV3 [13] and ConvMixer [52] against 1D equivalents, obtained by replacing every non-strided depthwise convolution with an oriented kernel. We train on ImageNet using the same setup as ConvNeXt. 1D models have 10%-25% higher inference throughput versus 2D models but lead to a drop of 0.8%-1% accuracy. Given that only substitution is done and no other tricks are used, this shows that 1D kernels show promise in working in other settings and architectures. Note that we target specifically an architecture which presents depthwise separable convolutions.

Model	Epochs	K	#Params	FLOPs	Throughput	Acc.
MobileNetv3-small 2D	120		5.5M	0.06G	6800 img/s	**60.9**
MobileNetv3-small 1D	120		5.5M	0.06G	**7200 img/s**	60.1
ConvMixer-768/32 2D	100	7	21.1M	20.7G	200img/s	**78.4**
ConvMixer-768/32 1D	100	15	20.3M	20.1G	**250img/s**	77.3

5.3. CUDA kernel speed comparison

We compare the speed of our oriented 1D kernel implementation versus the competitive PyTorch and RepLKNet CUTLASS [35] implementations and aggregate the results in Table 5. We conduct the experiments on a single NVIDIA RTX 3090, using single-precision FP32. In some instances, our custom kernel outperforms the CuDNN/CUTLASS implementation by a substantial margin. These algorithms rely on the assumption that computation is the main bottleneck, thus their design focuses on maximizing computational throughput in a cache-friendly manner, at the expense of reading more data. As a result, they are slower for smaller kernel sizes where data efficiency is critical. This also makes them hard to adapt to oriented kernels because they assume linear access patterns which results in bad performance, as we show in supplementary material.

		$K=7$								$K=31$							
	H, W	$\theta=0$	22.5	45	67.5	90	112.5	135	157.5	$\theta=0$	22.5	45	67.5	90	112.5	135	157.5
PyTorch/CuDNN	14^2	0.4								**0.4**							
CUTLASS [12]	14^2	0.6								0.6							
Ours	14^2	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	0.8	0.8	0.8	0.7	0.7	0.8	0.8	0.8
PyTorch/CuDNN	28^2	1.4								3.8							
CUTLASS [12]	28^2	2.3								**2.5**							
Ours	28^2	**0.9**	**0.9**	**0.9**	**0.9**	**1.0**	**0.9**	**0.9**	**0.9**	2.6	2.6	2.6	2.6	**2.5**	2.6	2.6	2.6
PyTorch/CuDNN	56^2	5.5								15.0							
CUTLASS [12]	56^2	9.4								10.2							
Ours	56^2	**3.2**	**3.3**	**3.2**	**3.2**	**3.2**	**3.2**	**3.2**	**3.2**	**9.8**	**9.8**	**9.8**	**9.9**	**9.9**	**9.9**	**10.0**	**10.0**

Table 5: **Runtime comparison** of our oriented 1D implementation on 1 NVIDIA RTX 3090 for $N=64, C=512$, FP32. The mean is taken over 100 runs, preceded by 10 dry runs. We benchmark against the very competitive PyTorch/CuDNN and RepLKNet CUTLASS [35] implementations on horizontal convolutions. Our implementation outperforms PyTorch consistently regardless of angle θ thanks to intelligent data access patterns, but falls off when computation becomes a bottleneck.

5.4. Ablation study

In this subsection, we carry out ablations to determine the influence on accuracy of characteristic oriented 1D kernel parameters. Table 6 compares oriented 1D models against non-oriented 2D baselines, Table 7 studies the impact of layerwise rotation, Table 8 studies the best combination of kernel size and direction and Table 9 looks at the best C_0. We comment the results in the next subsection.

Stem Design	Block Design	K	D	#Params	FLOPs	Top-1 Acc
2D	2D	7^2		28.6M	4.5G	**82.0**
2D	2D	31^2		30.0M	5.8G	81.6
Oriented 1D	1D	7	8	28.3M	4.4G	81.5
Oriented 1D	1D	31	8	28.5M	4.4G	81.8
Depthwise 2D	2D	7^2		28.6M	3.9G	82.0
Depthwise 2D	2D	31^2		34.8M	6.1G	81.4
Depthwise 1D	1D	7	8	28.3M	4.4G	82.0
Depthwise 1D	1D	31	8	28.5M	4.4G	**82.2**
Depthwise 2D	2D+2D	7^2+7^2		29.9M	4.3G	82.1
Depthwise 1D	1D++	15+31	8	29.2M	4.7G	**82.7**
2D	2D++	7^2+31	8	29.4M	4.7G	82.5

Table 6: **Comparison of oriented 1D models against 2D kernel baselines** on ConvNeXt-T. *Depthwise 2D* is the 2D transposition of *Depthwise 1D* and *2D+2D* is the 2D transposition of both *1D++* and *2D++* block designs. Overall, by switching to 1D and changing the stem, we are able to consistently match the performance of 2D baselines. We discuss these results in Section 5.5.

K	D	Layer-wise rotation	Top-1 Acc.
31	4	0°	82.01
31	4	alternating 90°	**82.11**

Table 7: **Layer-wise rotation** on ConvNeXt-1D. We see that adding layer-wise rotation increases accuracy.

5.5. Discussion

No loss in expressiveness. 1D ConvNeXt networks are *consistently* able to perform on par or outperfom their 2D equivalents according to Table 6. This shows that oriented 1D kernels can be made as expressive as 2D kernels provided they are integrated the right way.

Depthwise 1D stem. From Table 6, we see that our proposed *Depthwise 1D Stem* plays a key role in making 1D networks competitive with 2D networks, by reintroducing the stem spatial parameters that were lost switching to 1D.

Better long-range scaling with 1D. Table 6 shows that 2D models are unable to benefit from large kernel sizes K. This contrasts with the 1D kernel case and implies that 1D kernels exhibit better long-range scaling than 2D kernels.

K	D	#Params	FLOPs	Top-1 Acc.
7×7		28.6M	4.5G	**82.00**
7	2	28.3M	4.4G	81.80
7	4	28.3M	4.4G	81.90
7	8	28.3M	4.4G	**82.00**
7	C	28.3M	4.4G	81.65
15	2	28.4M	4.4G	81.84
15	4	28.4M	4.4G	82.05
15	8	28.4M	4.4G	**82.14**
15	C	28.4M	4.4G	82.12
31	2	28.5M	4.5G	81.67
31	4	28.5M	4.5G	82.01
31	8	28.5M	4.5G	**82.16**
31	C	28.5M	4.5G	81.92
63	2	28.5M	4.5G	81.73
63	4	28.5M	4.5G	81.92
63	8	28.5M	4.5G	**82.11**
63	C	28.5M	4.5G	81.97

Table 8: **Optimal K and D.** We compare ConvNext-T-1D with different kernel sizes K and directions D. No layer-wise rotation is used. $D = 8$ corresponds to the best number of directions and $K = 31$ seems the most advantageous in that case. Having too few $D = 2$ or too many directions $D = C$ seems to hurt performance.

K	D	C_0	#Params	FLOPs	Top-1 Acc.
31	4	4	28.5M	4.4G	78.9
31	4	16	28.5M	4.4G	81.67
31	4	64	28.5M	4.4G	**82.01**

Table 9: **Optimal C_0.** We compare ConvNext-T-1D with different stem channel sizes C_0. C_0 is directly tied to the number of spatial parameters in the stem. Increasing C_0 does not change the number of parameters yet results in a significant improvement in accuracy. This means that the spatial mixing done by depthwise convolutions is critical in achieving good performance. We fix this value to 64 in our models.

6. Conclusion

Our study suggests that oriented 1D kernels are viable alternatives to 2D kernels as they can be just as expressive, and are empirically better suited for larger kernel sizes.

Acknowledgement This work was partially supported by the National Science Foundation under Award IIS-1942981.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018. 3

[2] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. 3, 4

[3] Jiarong Chen, Zongqing Lu, Jing-Hao Xue, and Qingmin Liao. Xsepconv: Extremely separated convolution. *ArXiv*, abs/2002.12046, 2020. 2, 3

[4] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. 4

[5] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017. 2, 4

[6] Taco Cohen and Max Welling. Group Equivariant Convolutional Networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2990–2999. PMLR, June 2016. ISSN: 1938-7228. 2

[7] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*, pages 3008–3017. Computer Vision Foundation / IEEE, 2020. 6

[8] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 764–773, 2017. 1, 3

[9] 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, 2009. 6

[10] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. 2

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. 4

[12] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11963–11975, June 2022. 1, 2, 5, 6, 7

[13] Howard et al. Searching for mobilenetv3. In *ICCV*, 2019. 7

[14] W.T. Freeman and E.H. Adelson. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906, Sept. 1991. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence. 2

[15] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. 1

[16] Meng-Hao Guo, Cheng-Ze Lu, Qibin Hou, Zhengning Liu, Ming-Ming Cheng, and Shi-Min Hu. SegNeXt: Rethinking Convolutional Attention Design for Semantic Segmentation, Sept. 2022. Number: arXiv:2209.08575 arXiv:2209.08575 [cs]. 1, 2

[17] Daniel Haase and Manuel Amthor. Rethinking depthwise separable convolutions: How intra-kernel correlations lead to improved mobilenets. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 14588–14597. Computer Vision Foundation / IEEE, 2020. 3

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 4

[19] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 3, 4

[20] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018. 2

[21] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep Networks with Stochastic Depth. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 646–661, Cham, 2016. Springer International Publishing. 6

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. 3

[23] Jeremy Irvin, Pranav Rajpurkar, Michael Ko, Yifan Yu, Silviana Ciurea-Ilcus, Chris Chute, Henrik Marklund, Behzad Haghgoo, Robyn Ball, Katie Shpanskaya, Jayne Seekins, David A. Mong, Safwan S. Halabi, Jesse K. Sandberg, Ricky Jones, David B. Larson, Curtis P. Langlotz, Bhavik N. Patel, Matthew P. Lungren, and Andrew Y. Ng. Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison. In *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, 33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, pages 590–597. AAAI Press, 2019. Publisher Copyright: © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.; 33rd AAAI Conference on Artificial Intelligence,

AAAI 2019, 31st Annual Conference on Innovative Applications of Artificial Intelligence, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019 ; Conference date: 27-01-2019 Through 01-02-2019. 1

[24] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014. 2

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 1

[26] Durga Kumar and Xiaoling Zhang. Improving more instance segmentation and better object detection in remote sensing imagery based on cascade mask r-cnn. In *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pages 4672–4675, 2021. 7

[27] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. In *The handbook of brain theory and neural networks*, pages 255–258. MIT Press, Cambridge, MA, USA, Oct. 1998. 1

[28] Guoqing Li, Xuzhao Shen, Jiaojie Li, and Jiuyang Wang. Diagonal-kernel convolutional neural networks for image classification. *Digital Signal Processing*, 108:102898, Jan. 2021. 2

[29] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing. 6

[30] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Xuxi Chen, Qiao Xiao, Boqian Wu, Mykola Pechenizkiy, Decebal Mocanu, and Zhangyang Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. *arXiv preprint arXiv:2207.03620*, 2022. 1, 2, 6, 7

[31] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 9992–10002. IEEE, 2021. 4

[32] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11966–11976, 2022. 1, 2, 4, 6

[33] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 6

[34] Mark Martinez, Chawin Sitawarin, Kevin Finch, Lennart Meincke, Alex Yablonski, and Alain Kornhauser. Beyond Grand Theft Auto V for Training, Testing and Enhancing Deep Learning in Self Driving Cars, Dec. 2017. Number: arXiv:1712.01397 arXiv:1712.01397 [cs]. 1

[35] NVIDIA. CUTLASS: CUDA templates for linear algebra, 2022. 7

[36] Edouard Oyallon and Stephane Mallat. Deep roto-translation scattering for object classification. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2865–2873, Boston, MA, USA, June 2015. IEEE. 2

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019. 4

[38] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters – improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2

[39] Siamak Ravanbakhsh, Junier Oliva, Sebastien Fromenteau, Layne C. Price, Shirley Ho, Jeff Schneider, and Barnabás Póczos. Estimating cosmological parameters from the dark matter distribution. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 2407–2416. JMLR.org, 2016. 1

[40] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 1

[41] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. Learning Separable Filters. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2754–2761, June 2013. ISSN: 1063-6919. 2

[42] David W. Romero, David M. Knigge, Albert Gu, Erik J. Bekkers, Efstratios Gavves, Jakub M. Tomczak, and Mark Hoogendoorn. Towards a general purpose CNN for long range dependencies in ND. *CoRR*, abs/2206.03398, 2022. 3

[43] David W. Romero, Anna Kuzina, Erik J. Bekkers, Jakub Mikolaj Tomczak, and Mark Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. 3

[44] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 1, 2, 3, 4

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Inter-*

national Conference on Learning Representations, 2015. 2, 4
[46] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 4278–4284. AAAI Press, 2017. 2
[47] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. 1
[48] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. 2
[49] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, June 2016. ISSN: 1063-6919. 6
[50] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019. 1, 2
[51] Mingxing Tan and Quoc Le. EfficientNetV2: Smaller Models and Faster Training. In *Proceedings of the 38th International Conference on Machine Learning*, pages 10096–10106. PMLR, July 2021. ISSN: 2640-3498. 2
[52] Asher Trockman and J. Zico Kolter. Patches are all you need?, 2022. 7
[53] Maurice Weiler, Fred A. Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant cnns. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 849–858, 2018. 2
[54] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Harmonic networks: Deep translation and rotation equivariance. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7168–7177, 2017. 2
[55] Shuang Wu, Guanrui Wang, Pei Tang, Feng Chen, and Luping Shi. Convolution with even-sized kernels and symmetric padding. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 4, 5
[56] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 432–448, Cham, 2018. Springer International Publishing. 7
[57] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017. 1, 2
[58] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. 3
[59] Hang Yu, Laurence T. Yang, Qingchen Zhang, David Armstrong, and M. Jamal Deen. Convolutional neural networks for medical image analysis: State-of-the-art, comparisons, improvement and perspectives. *Neurocomputing*, 444:92–110, 2021. 1
[60] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features. pages 6023–6032, 2019. 6
[61] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. 6
[62] Richard Zhang. Making Convolutional Networks Shift-Invariant Again. In *Proceedings of the 36th International Conference on Machine Learning*, pages 7324–7334. PMLR, May 2019. ISSN: 2640-3498. 4
[63] Tu Zheng, Yifei Huang, Yang Liu, Wenjian Tang, Zheng Yang, Deng Cai, and Xiaofei He. Clrnet: Cross layer refinement network for lane detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 898–907, June 2022. 1
[64] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 13001–13008. AAAI Press, 2020. 6
[65] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5122–5130, 2017. 6, 7
[66] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets V2: more deformable, better results. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 9308–9316. Computer Vision Foundation / IEEE, 2019. 3