ACNet: Strengthening the Kernel Skeletons for Powerful CNN via Asymmetric Convolution Blocks

Xiaohan Ding [1] Yuchen Guo [2] Guiguang Ding [1] Jungong Han [3]
[1] Beijing National Research Center for Information Science and Technology (BNRist);
School of Software, Tsinghua University, Beijing, China
[2] Department of Automation, Tsinghua University;
Institute for Brain and Cognitive Sciences, Tsinghua University, Beijing, China
[3] WMG Data Science, University of Warwick, Coventry, United Kingdom
dxh17@mails.tsinghua.edu.cn yuchen.w.guo@gmail.com
dinggg@tsinghua.edu.cn jungonghan77@gmail.com

Abstract

As designing appropriate Convolutional Neural Network (CNN) architecture in the context of a given application usually involves heavy human works or numerous GPU hours, the research community is soliciting the architecture-neutral CNN structures, which can be easily plugged into multiple mature architectures to improve the performance on our real-world applications. We propose Asymmetric Convolution Block (ACB), an architecture-neutral structure as a CNN building block, which uses 1D asymmetric convolutions to strengthen the square convolution kernels. For an off-the-shelf architecture, we replace the standard square-kernel convolutional layers with ACBs to construct an Asymmetric Convolutional Network (ACNet), which can be trained to reach a higher level of accuracy. After training, we equivalently convert the ACNet into the same original architecture, thus requiring no extra computations anymore. We have observed that ACNet can improve the performance of various models on CIFAR and ImageNet by a clear margin. Through further experiments, we attribute the effectiveness of ACB to its capability of enhancing the model's robustness to rotational distortions and strengthening the central skeleton parts of square convolution kernels.

1. Introduction

Convolutional Neural Network (CNN) has achieved great success in visual understanding, which makes them useful for various applications in wearable devices, security systems, mobile phones, automobiles, *etc*. As the front-end devices are usually limited in computational resources and demand real-time inference, these applications require CNN that delivers high accuracy with the constraints of a certain level of computational budgets. Thus it may not be practical to enhance the model by simply employing more trainable parameters and complicated connections. Therefore, we consider it meaningful to improve the performance of CNN with no extra inference-time computations, memory footprint, or energy consumption.

On the other hand, along with the advancements in the CNN architecture designing literature, the performance of the off-the-shelf models has been significantly improved. However, when the existing models cannot meet our specific needs, we may not be allowed to customize a new architecture at the costs of heavy human works or numerous GPU hours [36]. Recently, the research community is soliciting innovative architecture-neutral CNN structures, *e.g*., SE blocks [14] and quasi-hexagonal kernels [30], which can be directly combined with various up-to-date architectures to improve the performance on our real-world applications.

Some recent investigations on CNN architectures focus on **1)** how the layers are connected with each other, *e.g*., simply stacked together [20, 28], through identity mapping [13, 31, 35] or densely connected [15] and **2)** how the outputs of different layers are combined to increase the quality of learned representations [16, 31, 32, 33]. Considering this, in quest of a generic architecture-neutral CNN structure which can be combined with numerous architectures, we seek to strengthen standard convolutional layers by digging into an orthogonal aspect: *the relationship between the weights and their spatial locations in the kernels*.

In this paper, we propose Asymmetric Convolution Block (ACB), an innovative structure as a building block to replace the standard convolutional layers with square kernels, *e.g*., 3×3 layers, which are widely used in modern CNN. Concretely, for the replacement of a $d \times d$ layer,

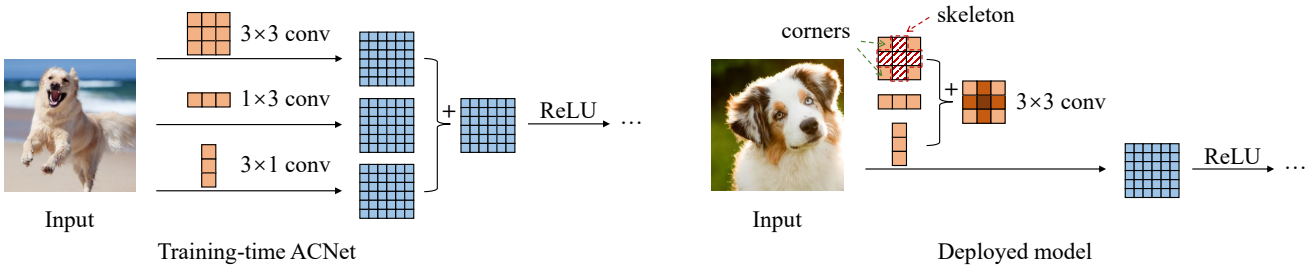


Figure 1: Overview of ACNet. For example, we replace every 3×3 layer with an ACB comprising three layers with 3×3, 1×3 and 3×1 kernels, respectively, and their outputs are summed up. When the training is completed, we convert the model back into the same structure as the original by adding the asymmetric kernels in each ACB onto the skeleton, which is the crisscross part of the square kernel, as marked on the figure. In practice, this conversion is implemented by building a new model with the original structure and using the converted learned parameters of the ACNet to initialize it.

we construct an ACB comprising three parallel layers with $d \times d$, $1 \times d$ and $d \times 1$ kernels, respectively, of which the outputs are summed up to enrich the feature space (Fig. 1). As the introduced $1 \times d$ and $d \times 1$ layers have non-square kernels, we refer to them as the asymmetric convolutional layers, following [33]. Given an off-the-shelf architecture, we construct an Asymmetric Convolutional Network (ACNet) by replacing every square-kernel layer with an ACB and train it until convergence. After that, we equivalently convert the ACNet into the same original architecture by adding the asymmetric kernels in each ACB onto the corresponding positions of the square kernels. Due to the additivity of convolutions with compatible kernel sizes (Fig. 2), which is obvious but has long been ignored, the resulting model can produce the same outputs as the training-time ACNet. As will be shown in our experiments (Sect. 4.1, 4.2), doing so can improve the performance of several benchmark models on CIFAR [19] and ImageNet [3] by a clear margin. Better still, ACNet **1)** introduces NO hyper-parameters, such that it can be combined with different architectures without careful tuning; **2)** is simple to implement on the mainstream CNN frameworks like PyTorch [26] and Tensorflow [1]; **3)** requires NO extra inference-time computational burdens compared to the original architecture.

Through our further experiments, we have partly explained the effectiveness of ACNet. It is observed that a square convolution kernel distributes its learned knowledge unequally, as the weights on the central crisscross positions (which are referred to as the *"skeleton"* of the kernel) are usually larger in magnitude, and removing them causes higher accuracy drop, compared to those in the corners. In each ACB, we add the horizontal and vertical kernels onto the skeletons, thus explicitly making the skeletons more powerful, following the nature of square kernels. Interestingly, the weights on the corresponding positions of the square, horizontal and vertical kernels are randomly initialized and have a possibility to grow opposite in sign, thus summing them up may result in a stronger or weaker skeleton. However, we have empirically observed a consistent phenomenon that the model always learn to enhance the skeletons at every layer. This observation may shed light on future researches on the relationship among the weights at different spatial locations. The codes are available at https://github.com/ShawnDing1994/ACNet.

Our contributions are summarized as follows.

- We propose to use asymmetric convolutions to explicitly enhance the representational power of a standard square-kernel layer in a way that the asymmetric convolutions can be fused into the square kernels with NO extra inference-time computations needed, rather than approximate a square-kernel layer like many prior works [4, 17, 18, 23, 25, 33].
- We propose ACB as a novel architecture-neutral CNN building block. We can construct an ACNet by simply replacing every square-kernel convolutional layer in a mature architecture with an ACB without introducing any hyper-parameters, such that its effectiveness can be combined with the numerous advancements in the CNN architecture designing literature.
- We have improved the accuracy of several common benchmark models on CIFAR-10, CIFAR-100, and ImageNet by a clear margin.
- We have justified the significance of skeletons in standard square convolution kernels and demonstrated the effectiveness of ACNet in enhancing such skeletons.
- We have shown that ACNet can enhance the model's robustness to rotational distortions, which may inspire further studies on the rotational invariance problem.

2. Related work

2.1. Asymmetric convolutions

Asymmetric convolutions are typically used to approximate an existing square-kernel convolutional layer for compression and acceleration. Some prior works [4, 17] have shown that a standard $d \times d$ convolutional layer can be fac-

torized as a sequence of two layers with $d \times 1$ and $1 \times d$ kernels to reduce the parameters and required computations. The theory behind is quite simple: if a 2D kernel has a rank of one, the operation can be equivalently transformed into a series of 1D convolutions. However, as the learned kernels in deep networks have distributed eigenvalues, their intrinsic rank is higher than one in practice, thus applying the transformation directly to the kernels results in significant information loss [18]. Denton *et al*. [4] tackled this problem by finding a low-rank approximation in an SVD-based manner then finetuning the upper layers to restore the performance. Jaderberg *et al*. [17] succeeded in learning the horizontal and vertical kernels by minimizing the ℓ-2 reconstruction error. Jin *et al*. [18] applied structural constraints to make the 2D kernels separable and obtained comparable performance as conventional CNN with $2\times$ speed-up.

On the other hand, asymmetric convolutions are also widely employed as an architectural design element to save the parameters and computations. For example, in Inception-v3 [33], the 7×7 convolutions are replaced by a sequence of 1×7 and 7×1 convolutions. However, the authors found out that such replacement is not equivalent as it did not work well on the low-level layers. ENet [25] also adopted this approach for the design of an efficient semantic segmentation network, where the 5×5 convolutions are decomposed, allowing to increase the receptive field with reasonable computational budgets. EDANet [23] used a similar method to decompose the 3×3 convolutions, resulting in a 33% saving in the number of parameters and required computations with minor performance degradation.

In contrast, we use 1D asymmetric convolutions not to factorize any layers as part of the architectural designs but enrich the feature space during training and then fuse their learned knowledge into the square-kernel layers.

2.2. Architecture-neutral CNN structures

We intend not to modify the CNN architecture but use some architecture-neutral structures to enhance the off-the-shelf models. Thus the effectiveness of our method is supplementary to the advancements achieved by the innovative architectures. Specifically, a CNN structure can be called architecture-neutral if it **1)** makes no assumptions on the specific architecture, thus can be applied on various models, and **2)** brings universal benefits. For example, SE blocks [14] can be appended after a convolutional layer to rescale the feature map channels with learned weights, resulting in a clear accuracy improvement at reasonable costs of extra parameters and computational burdens. As another example, auxiliary classifier [32] can be inserted into the model to assist in supervising the learning process, which can indeed improve the performance by an observable margin but requires extra human works to tune the hyper-parameters.

By contrast, ACNet introduces *NO hyper-parameters* during training and requires *NO extra parameters or computations* during inference. Therefore, in real-world applications, the developer can use ACNet to enhance a variety of models without exhausting parameter tunings, and the end-users can enjoy the performance improvement without slowing down the inference. Better still, since we introduce no custom structures into the deployed model, it can be future compressed via techniques including connection pruning [9, 12], channel pruning [5, 6, 22, 24], quantization [2, 10, 27], feature map compacting [34], *etc*.

3. Asymmetric Convolutional Network

3.1. Formulation

For a convolutional layer with a kernel size of $H \times W$ and D filters which takes a C-channel feature map as input, we use $\boldsymbol{F} \in \mathbb{R}^{H \times W \times C}$ to denote the 3D convolution kernel of a filter, $\boldsymbol{M} \in \mathbb{R}^{U \times V \times C}$ for the input, which is a feature map with a spatial resolution of $U \times V$ and C channels, and $\boldsymbol{O} \in \mathbb{R}^{R \times T \times D}$ for the output with D channels, respectively. For the j-th filter at such a layer, the corresponding output feature map channel is

$$\boldsymbol{O}_{:,:,j} = \sum_{k=1}^{C} \boldsymbol{M}_{:,:,k} * \boldsymbol{F}^{(j)}_{:,:,k}\,, \tag{1}$$

where $*$ is the 2D convolution operator, $\boldsymbol{M}_{:,:,k}$ is the k-th channel of $\boldsymbol{M}$ in the form of a $U \times V$ matrix, and $\boldsymbol{F}^{(j)}_{:,:,k}$ is the k-th input channel of $\boldsymbol{F}^{(j)}$, *i.e*., a 2D kernel of $H \times W$.

In modern CNN architectures, batch normalizations [16] are widely adopted to reduce overfitting and accelerate the training process. As a common practice, a batch normalization layer is usually followed by a linear scaling transformation to enhance the representational power. Compared to Eq. 1, the output channel then becomes

$$\boldsymbol{O}_{:,:,j} = (\sum_{k=1}^{C} \boldsymbol{M}_{:,:,k} * \boldsymbol{F}^{(j)}_{:,:,k} - \mu_j)\frac{\gamma_j}{\sigma_j} + \beta_j\,, \tag{2}$$

where μ_j and σ_j are the values of channel-wise mean and standard deviation of batch normalization, γ_j and β_j are the learned scaling factor and bias term, respectively.

3.2. Exploiting the additivity of convolution

We seek to employ asymmetric convolutions in a way that they can be equivalently fused into the standard square-kernel layers, such that no extra inference-time computational burdens are introduced. We notice a useful property of convolution: if several 2D kernels with *compatible* sizes operate on the same input with the same stride to produce outputs of the same resolution, and their outputs are summed up, we can add up these kernels *on the corresponding positions* to obtain an equivalent kernel which will produce the same output. That is, the *additivity* may hold for

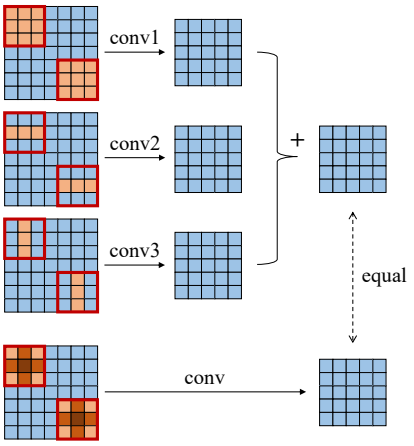


Figure 2: We use sliding windows to provide some intuitions of the additivity of 2D convolutions with different kernel sizes. Here we have three convolutional layers with a kernel size of 3×3, 1×3 and 3×1, respectively, which take the same input. We only depict the sliding window at the top-left and bottom-right corners, for example. It can be observed that the key for the additivity to hold is that the three layers can *share the same sliding window*. Therefore, if we add the kernels of conv2 and conv3 to conv1 on the corresponding positions, using the resulting kernel to operate on the original input will produce the same result, which can be easily verified only using the distributive property of multiplication (Eq. 5). Best viewed in color.

2D convolutions, even with different kernel sizes,

$$\boldsymbol{I} * \boldsymbol{K}^{(1)} + \boldsymbol{I} * \boldsymbol{K}^{(2)} = \boldsymbol{I} * (\boldsymbol{K}^{(1)} \oplus \boldsymbol{K}^{(2)}) , \quad (3)$$

where $\boldsymbol{I}$ is a matrix, $\boldsymbol{K}^{(1)}$ and $\boldsymbol{K}^{(2)}$ are two 2D kernels with compatible sizes, and $\oplus$ is the element-wise addition of the kernel parameters on the corresponding positions. Note is that $\boldsymbol{I}$ may need to be appropriately clipped or padded.

Here *compatible* means that we can "patch" the smaller kernel onto the bigger. Formally, this kind of transformation on layer p and q is feasible if

$$\boldsymbol{M}^{(p)} = \boldsymbol{M}^{(q)} , H_p \leq H_q , W_p \leq W_q , D_p = D_q . \quad (4)$$

E.g., 3×1 and 1×3 kernels are compatible with 3×3.

This can be easily verified by investigating the calculation of convolution in the form of sliding windows (Fig. 2). For a certain filter with kernel $\boldsymbol{F}^{(j)}$, a certain point y on the output channel $\boldsymbol{O}_{:,:,j}$ is given by

$$y = \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} F^{(j)}_{h,w,c} X_{h,w,c} , \quad (5)$$

where $\boldsymbol{X}$ is the corresponding sliding window on input $\boldsymbol{M}$. Obviously, when we sum up two output channels produced

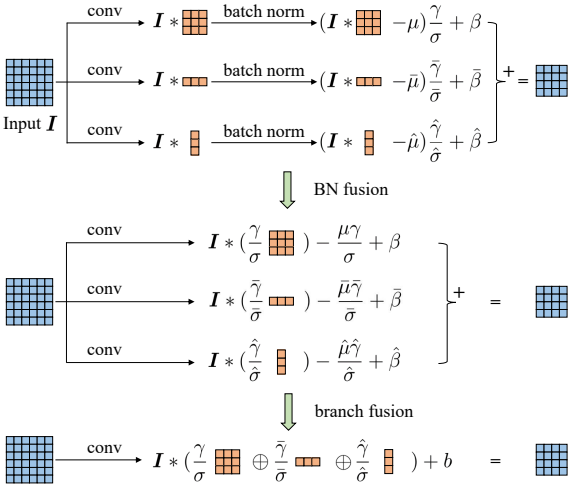


Figure 3: BN and branch fusion. Let $\boldsymbol{I}$ be an arbitrary channel of the input feature map $\boldsymbol{M}$, for each branch, we first equivalently fuse the parameters of batch normalization into the convolution kernel and a bias term, then add up the fused kernels and bias terms to obtain a single layer.

by two filters, the additivity (Eq. 3) holds if for each point y on one channel, its corresponding point on the other channel *shares the same sliding window* $\boldsymbol{X}$.

3.3. ACB for free inference-time improvements

In this paper, we focus on 3×3 convolutions, which are heavily used in modern CNN architectures. Given an architecture, we construct an ACNet by simply replacing every 3×3 layer (together with the following batch normalization layer, if any) with an ACB which comprises three parallel layers with kernel size 3×3, 1×3 and 3×1, respectively. Similar to the common practice in standard CNN, each of the three layers is followed by batch normalization, which is referred to as a branch, and the outputs of three branches are summed up as the output of ACB. Note that we can train the ACNet using the same configurations as the original model without any extra hyper-parameters to be tuned.

As will be shown in Sect. 4.1 and Sect. 4.2, the ACNet can be trained to reach a higher level of accuracy. When the training is completed, we seek to convert every ACB to a standard convolutional layer which produces identical outputs. By doing so, we can obtain a more powerful network which requires no extra computations, compared to a normally trained counterpart. This conversion is achieved through two steps, namely, BN fusion and branch fusion.

BN fusion. The *homogeneity* of convolution allows the following batch normalization and linear scaling transformation to be equivalently fused into the convolutional layer with an added bias. It can be observed from Eq. 2 that for

each branch, if we construct a new kernel as $\frac{\gamma_j}{\sigma_j}\boldsymbol{F}^{(j)}$ along with an added bias term $-\frac{\mu_j\gamma_j}{\sigma_j}+\beta_j$, we will produce the same output, which can be easily verified.

Branch fusion. We merge the three BN-fused branches into a standard convolutional layer by adding the asymmetric kernels onto the corresponding positions of the square kernel. In practice, this transformation is implemented by building a network of the original structure and using the fused weights for initialization, thus we can produce the same outputs as the ACNet with the same computational budgets as the original architecture. Formally, for every filter j, let $\boldsymbol{F}'^{(j)}$ be the fused 3D kernel, b_j be the obtained bias term, $\bar{\boldsymbol{F}}^{(j)}$ and $\hat{\boldsymbol{F}}^{(j)}$ be the kernels of the corresponding filter at the 1×3 and 3×1 layer, respectively, we have

$$\boldsymbol{F}'^{(j)} = \frac{\gamma_j}{\sigma_j}\boldsymbol{F}^{(j)} \oplus \frac{\bar{\gamma}_j}{\bar{\sigma}_j}\bar{\boldsymbol{F}}^{(j)} \oplus \frac{\hat{\gamma}_j}{\hat{\sigma}_j}\hat{\boldsymbol{F}}^{(j)}\,, \tag{6}$$

$$b_j = -\frac{\mu_j\gamma_j}{\sigma_j} - \frac{\bar{\mu}_j\bar{\gamma}_j}{\bar{\sigma}_j} - \frac{\hat{\mu}_j\hat{\gamma}_j}{\hat{\sigma}_j} + \beta_j + \bar{\beta}_j + \hat{\beta}_j\,. \tag{7}$$

Then we can easily verify that for an arbitrary filter j,

$$\boldsymbol{O}_{:,:,j} + \bar{\boldsymbol{O}}_{:,:,j} + \hat{\boldsymbol{O}}_{:,:,j} = \sum_{k=1}^{C} \boldsymbol{M}_{:,:,k} * \boldsymbol{F}'^{(j)}_{:,:,k} + b_j\,, \tag{8}$$

where $\boldsymbol{O}_{:,:,j}$, $\bar{\boldsymbol{O}}_{:,:,j}$ and $\hat{\boldsymbol{O}}_{:,:,j}$ are the outputs of the original 3×3, 1×3 and 3×1 branch, respectively. Fig. 3 shows an example on a single input channel for more intuitions.

Of note is that though an ACB can be equivalently transformed into a standard layer, the equivalence only holds at inference-time because the training dynamics are different, thus giving rise to different performance. The non-equivalence of the training process is due to the random initialization of kernel weights, and the gradients derived by different computation flows they participate in.

4. Experiments

We have conducted abundant experiments to verify the effectiveness of ACNet in improving the performance of CNN across a range of datasets and architectures. Concretely, we pick an off-the-shelf architecture as the baseline, build an ACNet counterpart, train it from scratch, convert it into the same structure as the baseline, and test it to collect the accuracy. For the comparability, all the models are trained until the complete convergence, and every pair of baseline and ACNet uses identical configurations, *e.g.*, learning rate schedules and batch sizes.

4.1. Performance improvements on CIFAR

In order to preliminarily evaluate our method on various CNN architectures, we experiment with several representative benchmark models including Cifar-quick [29], VGG-16 [28], ResNet-56 [13], WRN-16-8 [35] and DenseNet-40 [15] on CIFAR-10 and CIFAR-100 [19].

Table 1: Top-1 accuracy of ACNets and the normally trained baselines on CIFAR-10.

Model	Base Top-1	ACNet Top-1	Top-1 ↑
Cifar-quick	83.13	84.24	1.11
VGG	94.12	94.47	0.35
ResNet-56	94.31	95.09	0.78
WRN-16-8	95.56	96.15	0.59
DenseNet-40	94.29	94.84	0.55

Table 2: Top-1 accuracy of ACNets and the normally trained baselines on CIFAR-100.

Model	Base Top-1	ACNet Top-1	Top-1 ↑
Cifar-quick	53.22	54.30	1.08
VGG	74.56	75.20	0.64
ResNet-56	73.58	74.04	0.46
WRN-16-8	78.65	79.44	0.79
DenseNet-40	73.14	73.41	0.27

For Cifar-quick, VGG-16, ResNet-56, and DenseNet-40, we train the models using a staircase learning rate of 0.1, 0.01, 0.001 and 0.0001 following the common practice. For WRN-16-8, we follow the training configurations reported in the original paper [35]. We use the data augmentation techniques adopted by [13], *i.e.*, padding to 40×40, random cropping and left-right flipping.

As can be observed from Table. 1 and Table. 2, the performance of all the models is consistently lifted by a clear margin, suggesting that the benefits of ACBs can be combined with various architectures.

4.2. Performance improvements on ImageNet

We then move on to the effectiveness validation of our method on the real-world applications through a series of experiments on ImageNet [3] which comprises 1.28M images for training and 50K for validation from 1000 classes. We use AlexNet [20], ResNet-18 [13] and DenseNet-121 [15] as the representatives for the plain-style, residual and densely connected architectures, respectively. Every model is trained with a batch size of 256 for 150 epochs, which is longer than the usually adopted benchmarks (*e.g.*, 90 epochs [13]), such that the accuracy improvement cannot be simply attributed to the incomplete convergence of the base models. For the data augmentation, we employ the standard pipeline including bounding box distortion, left-right flipping and color shift, as a common practice. Especially, the plain version of AlexNet we use comes from the Tensorflow GitHub [8], which is composed of five stacked con-

Table 3: Accuracy of the ACNet counterparts of AlexNet, ResNets, DenseNet-121 and the baselines on ImageNet.

Model	Base Top-1	ACNet Top-1	Top-1 ↑	Base Top-5	ACNet Top-5	Top-5 ↑
AlexNet	55.92	**57.44**	1.52	79.53	**80.73**	1.20
ResNet-18	70.36	**71.14**	0.78	89.61	**89.96**	0.35
DenseNet-121	75.15	**75.82**	0.67	92.45	**92.77**	0.32

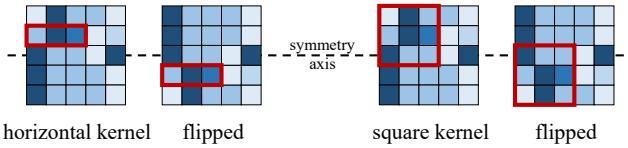


Figure 4: Compared to square kernels, horizontal kernels are more robust to up-down flipping. As shown above, the 1×3 kernel will produce the same results on the symmetric positions of the flipped inputs, but the 3×3 kernel will not.

volutional layers and three fully-connected layers with no local response normalizations (LRN) or cross-GPU connections. For the faster convergence, we apply batch normalization [16] on its every convolutional layer. Of note is that since the first two layers of AlexNet use 11×11 and 5×5 kernels, respectively, it is possible to extend ACBs to have larger asymmetric kernels. However, we still only use 1×3 and 3×1 convolutions for these two layers, because such large-scale convolutions are becoming less favored in modern CNN, making large ACBs less useful.

As shown in Table. 3, the single-crop Top-1 accuracy of AlexNet, ResNet-18 and DenseNet-121 is lifted by 1.52%, 0.78% and 1.18%, respectively. In practice, aiming at the same target of accuracy, we can use ACNet to enhance a more efficient model to achieve the target with less inference time, energy consumption, and storage space. On the other hand, with the same constraints on computational budgets or model size, we can use ACNet to improve the accuracy by a clear margin such that the gained performance can be viewed as free benefits, from the viewpoint of end-users.

4.3. Ablation studies

Though we have empirically justified the effectiveness of ACNet, we still desire to find some explanations. In this subsection, we seek to investigate ACNet through a series of ablation studies. Specifically, we focus on the following three design decisions: the usage of **1)** horizontal kernels, **2)** vertical kernels, and **3)** batch normalization in every branch. For the comparability, we train several AlexNet and ResNet-18 models on ImageNet with different ablations using the same training configurations. Of note is that if the batch normalizations in the branches are removed, we batch-normalize the output of the whole ACB instead, *i.e.*, the position of batch normalization layer is changed from pre-summation to post-summation.

As can be observed from Table. 4, removing any of the three designs degrades the model. However, though the horizontal and vertical convolutions can both improve the performance, there may exist some difference because the horizontal and vertical directions are treated unequally in practice, *e.g.*, we usually perform random left-right but no up-down image flipping to augment the training data. Therefore, if an upside-down image is fed into the model, the original 3×3 layers should produce meaningless results, which is natural, but a horizontal kernel will produce the same outputs as on the original image at the axially symmetric locations (Fig. 4). *I.e.*, a part of the ACB can still extract the correct features. Considering this, we assume that ACBs may enhance the model's robustness to rotational distortions, enabling the model to generalize better on the unseen data.

We then test the formerly trained models with rotationally distorted images from the whole validation set including counterclockwise 90° rotation, 180° rotation, and up-down flipping. Naturally, the accuracy of every model is significantly reduced, but the models with horizontal kernels deliver observably higher accuracy on the 180° rotated and up-down flipped images. *E.g.*, the ResNet-18 equipped with only horizontal kernels delivers an accuracy slightly lower than that of the counterpart with only vertical kernels on the original inputs, but 0.75% higher on the 180° rotated inputs. And when compared with the base model, its accuracy is 0.34% / 1.27% higher on the original / 180° flipped images, respectively. Predictably, the models exert similar performance on the 180° rotated and up-down flipped inputs, as 180° rotation plus left-right flipping is equivalent to up-down flipping, and the model is robust to left-right flipping due to the data augmentation methods.

In summary, we have shown that ACBs, especially the horizontal kernels inside, can enhance the model's robustness to rotational distortions by an observable margin. Though this may not be the primary reason for the effectiveness of ACNet, we consider it promising to inspire further researches on the rotational invariance problem.

4.4. ACB enhances the skeletons of square kernels

Intuitively, as adding the horizontal and vertical kernels onto the square kernel can be viewed as a means to explicitly enhance the skeleton part, we seek to explain the effec-

Table 4: Top-1 accuracy of the ACNets with different design configurations and rotational distortions on ImageNet.

Model	Horizontal kernel	Vertical kernel	BN in branch	Original input	Rotate 90°	Rotate 180°	Up-down flip
AlexNet				55.92	28.18	31.41	31.62
AlexNet		✓	✓	57.10	29.65	32.86	33.02
AlexNet	✓		✓	57.25	29.97	33.74	33.74
AlexNet	✓	✓	✓	**57.44**	**30.49**	**33.98**	**33.82**
AlexNet	✓	✓		56.18	28.81	32.12	32.33
ResNet-18				70.36	41.00	41.95	41.86
ResNet-18		✓	✓	70.78	41.61	42.47	42.66
ResNet-18	✓		✓	70.70	42.06	**43.22**	43.05
ResNet-18	✓	✓	✓	**71.14**	**42.20**	42.89	**43.10**
ResNet-18	✓	✓		70.82	41.70	42.92	42.90

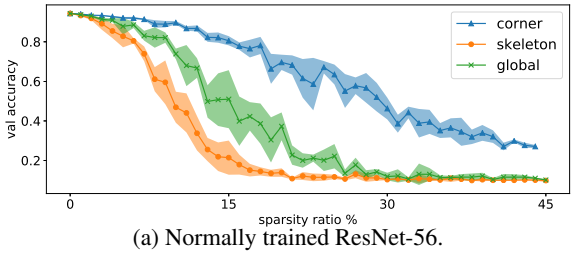
tiveness of ACNet by investigating the difference between the skeleton and the weights at the corners.

Inspired by the CNN pruning methods [9, 11, 12], we start from removing some weights at different spatial locations and observing the performance drop using ResNet-56 on CIFAR-10. Concretely, we randomly set some individual weights in the kernels to zero and test the model. As shown in Fig. 5a, for the curve labeled as *corner*, we randomly select the weights from the four corners of every 3×3 kernel and set them to zero in order to attain a given global sparsity ratio of *every* convolutional layer. Note that as $4/9 = 44.4\%$, a sparsity ratio of 44% means removing most of the weights at the four corners. For *skeleton*, we randomly select the weights only from the skeleton of every kernel. For *global*, every individual weight in the kernel has an equal chance to be chosen. The experiments are repeated five times with different random seeds, and the *mean*$\pm$*std* curves are depicted.

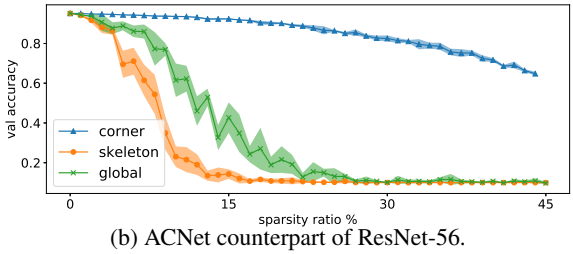
As can be observed, all the curves show a tendency of decreasing as the sparsity ratio increases, but not monotonically, due to the random effects. It is obvious that removing the weights from the corners causes less damage to the model, but pruning the skeletons does more harm. This phenomenon suggests that the skeleton weights are more important to the model's representational capacity.

We continue to verify if this observation holds for ACNet. We convert the ACNet counterpart via BN and branch fusion, then conduct the same experiments on it. As shown in Fig. 5b, we observe an even more significant gap, *e.g.*, pruning almost all the corner weights only degrades the model's accuracy to above 60%. On the other hand, pruning the skeletons causes more damage, as the model is destroyed when the global sparsity ratio attained by pruning the skeletons merely reaches 13%, *i.e.*, $13\% \times 9/5 = 23.4\%$ weights of the skeletons are removed.

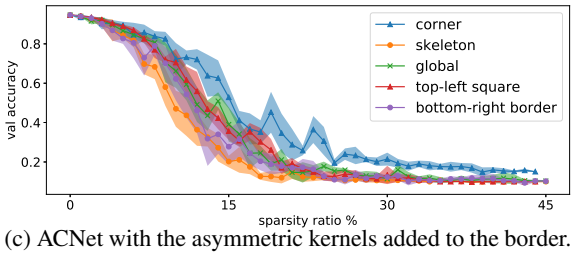
Then we explore the cause of the above observations by investigating the numeric values of the kernels. We use the



(a) Normally trained ResNet-56.



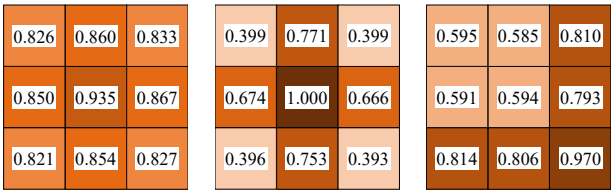
(b) ACNet counterpart of ResNet-56.



(c) ACNet with the asymmetric kernels added to the border.

Figure 5: Validation accuracy of different ResNet-56 models on CIFAR-10 with increasing sparsity ratios attained by pruning weights at different locations of the 3×3 kernels.

magnitude (*i.e.*, absolute value) as the metric for the importance of parameters, which is adopted by many prior CNN pruning works [7, 9, 12, 21]. Specifically, we add up all the fused 2D kernels in a convolutional layer, perform a layer-wise normalization by the max value, and finally obtain an average of the normalized kernels of all the layers. Formally, let $\boldsymbol{F}^{(i,j)}$ be the 3D kernel of the j-th filter at the i-th



(a) Normal. (b) ACNet, skeleton. (c) ACNet, border.

Figure 6: The average kernel magnitude matrix $\boldsymbol{A}$ of ResNet-56 models trained in different ways on CIFAR-10.

3×3 layer, L be the number of all such layers, max and abs be the max and element-wise absolute value, respectively, the average kernel magnitude matrix is computed as

$$\boldsymbol{A} = \frac{1}{L} \sum_{i=1}^{L} \frac{\boldsymbol{S}^{(i)}}{max(\boldsymbol{S}^{(i)})}, \tag{9}$$

where the sum of absolute kernels of layer i is

$$\boldsymbol{S}^{(i)} = \sum_{j=1}^{D_i} \sum_{k=1}^{C_i} abs(\boldsymbol{F}^{(i,j)}_{:,:,k}). \tag{10}$$

We present the $\boldsymbol{A}$ values of the normally trained ResNet-56 and the fused ACNet counterpart in Fig. 6a and Fig. 6b, where the numeric value and color at a certain grid indicate the average relative importance of the parameter on the corresponding position across all the 3×3 layers, *i.e.*, a larger value and darker background color indicates a higher average importance of the parameter.

As can be observed from Fig. 6a, the normally trained ResNet-56 distributes the magnitude of the parameters in an imbalance manner, *i.e.*, the central point has the largest magnitude, and the points at the four corners have the smallest. Fig. 6b shows that ACNet aggravates such imbalance, as the $\boldsymbol{A}$ values of the four corners are decreased to below 0.400, and the skeleton points have the $\boldsymbol{A}$ values above 0.666. In particular, the central point has an $\boldsymbol{A}$ value of 1.000, which means that this location has a dominant importance consistently in *every* single 3×3 layer. It is noteworthy that the weights on the corresponding positions of the square, horizontal and vertical kernels have a possibility to grow opposite in sign, thus summing them up may result in a larger or smaller magnitude. But we have observed a consistent phenomenon that the model always learn to enhance the skeletons at *every layer*.

We continue to study how the model will behave if we add the asymmetric kernels onto the other positions rather than the central skeletons. Specifically, we train an ACNet counterpart of ResNet-56 using the same training configurations as before, but shift the horizontal convolutions one pixel towards the bottom on the inputs and shift the vertical convolutions towards the right. Accordingly, during branch fusion, we add the BN-fused asymmetric kernels to the bottom-right borders of the square kernels (Fig. 6c) in order for an equivalent resulting network. It is observed that such ACBs can also enhance the borders, but not as intensively as the regular ACBs do to the skeletons. The model delivers an accuracy of 94.67%, which is 0.42% lower than the regular ACNet (Table. 1). Moreover, similar pruning experiments are conducted on the fused model (Fig. 5c). As observed, pruning the corners still delivers the best accuracy, and pruning the enhanced bottom-right borders gives no better results than the top-left 2×2 squares, *i.e.*, though the magnitudes of the borders have increased, the other parts remain essential to the whole kernels.

In summary: **1)** the skeletons are inherently more important than the corners in standard square kernels; **2)** ACB can significantly enhance the skeletons, resulting in improved performance; **3)** adding the horizontal and vertical kernels to the borders degrades the model's performance compared to regular ACBs; **4)** doing so can also increase the magnitude of the borders but cannot diminish the importance of the other parts. Therefore, we partly attribute the effectiveness of ACNet to its capability of further strengthening the skeletons. Intuitively, ACNet follows the nature of the square convolution kernels.

5. Conclusion

In order to improve the performance of various CNN architectures, we proposed Asymmetric Convolution Block (ACB), which sums up the outputs of three convolutional branches with square, horizontal and vertical kernels, respectively. We construct an Asymmetric Convolutional Network (ACNet) by replacing the square-kernel layers in a mature architecture with ACBs and convert it into the original architecture after training. We have evaluated ACNet by improving various plain-style, residual and densely connected models on CIFAR and ImageNet. We have shown that ACNet can enhance the model's robustness to rotational distortions by an observable margin, and explicitly strengthening the skeletons following the nature of square kernels. Of note is that ACNet introduces NO hyper-parameters to be tuned, requires NO extra inference-time computations, and is simple to implement using mainstream frameworks.

Acknowledgement

This work was supported by the National Key R&D Program of China (No. 2018YFC0807500), National Natural Science Foundation of China (No. 61571269), National Postdoctoral Program for Innovative Talents (No. BX20180172), and the China Postdoctoral Science Foundation (No. 2018M640131). Corresponding author: Guiguang Ding, Yuchen Guo.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. 2

[2] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016. 3

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 2, 5

[4] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014. 2, 3

[5] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4943–4953, 2019. 3

[6] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. In *International Conference on Machine Learning*, pages 1607–1616, 2019. 3

[7] Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 7

[8] GoogLe. Tensorflow-alexnet. https://github.com/tensorflow/models/blob/master/research/slim/nets/alexnet.py, 2017. 5

[9] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016. 3, 7

[10] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015. 3

[11] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 7

[12] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015. 3, 7

[13] 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. 1, 5

[14] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 1, 3

[15] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017. 1, 5

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 1, 3, 6

[17] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 2, 3

[18] Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014. 2, 3

[19] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009. 2, 5

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1, 5

[21] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 7

[22] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763. IEEE, 2017. 3

[23] Shao-Yuan Lo, Hsueh-Ming Hang, Sheng-Wei Chan, and Jing-Jhih Lin. Efficient dense modules of asymmetric convolution for real-time semantic segmentation. *arXiv preprint arXiv:1809.06323*, 2018. 2, 3

[24] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017. 3

[25] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016. 2, 3

[26] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 2

[27] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016. 3

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 5

[29] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012. 5

[30] Zhun Sun, Mete Ozay, and Takayuki Okatani. Design of kernels in convolutional neural networks for image classification. In *European Conference on Computer Vision*, pages 51–66. Springer, 2016. 1

[31] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 1

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1, 3

[33] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 1, 2, 3

[34] Yunhe Wang, Chang Xu, Chao Xu, and Dacheng Tao. Beyond filters: Compact feature map for portable deep model. In *International Conference on Machine Learning*, pages 3703–3711, 2017. 3

[35] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 1, 5

[36] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 1