Diverse Branch Block: Building a Convolution as an Inception-like Unit

Xiaohan Ding [1*] Xiangyu Zhang [2] Jungong Han [3] Guiguang Ding [1†]
[1] Beijing National Research Center for Information Science and Technology (BNRist); School of Software, Tsinghua University, Beijing, China
[2] MEGVII Technology
[3] Computer Science Department, Aberystwyth University, SY23 3FL, UK
dxh17@mails.tsinghua.edu.cn zhangxiangyu@megvii.com
jungonghan77@gmail.com dinggg@tsinghua.edu.cn

Abstract

We propose a universal building block of Convolutional Neural Network (ConvNet) to improve the performance without any inference-time costs. The block is named Diverse Branch Block (DBB), which enhances the representational capacity of a single convolution by combining diverse branches of different scales and complexities to enrich the feature space, including sequences of convolutions, multi-scale convolutions, and average pooling. After training, a DBB can be equivalently converted into a single conv layer for deployment. Unlike the advancements of novel ConvNet architectures, DBB complicates the training-time microstructure while maintaining the macro architecture, so that it can be used as a drop-in replacement for regular conv layers of any architecture. In this way, the model can be trained to reach a higher level of performance and then transformed into the original inference-time structure for inference. DBB improves ConvNets on image classification (up to 1.9% higher top-1 accuracy on ImageNet), object detection and semantic segmentation. The PyTorch code and models are released at https://github.com/DingXiaoH/DiverseBranchBlock.

1. Introduction

Improving the performance of Convolutional Neural Network (ConvNet) has always been a heated research topic. On one hand, the advancements in architecture design, *e.g.*, the Inception models [27, 28, 26, 15], have revealed that the multi-branch topology and combination of various paths with different scales and complexities can enrich the feature space and improve the performance. However, the complicated structure usually slows down the inference, as a combination of small operators (*e.g.*, concatenation of 1×1 conv and pooling) is not friendly to the devices with strong parallel computing powers like GPU [21].

On the other hand, more parameters and connections usually lead to higher performance, but the size of ConvNet we deploy cannot increase arbitrarily due to the business requirements and hardware constraints. Considering this, we usually judge the quality of a ConvNet by the trade-off between performance and inference-time costs such as the latency, memory footprint and number of parameters. In the common cases, we train the models on powerful GPU workstations and deploy them onto efficiency-sensitive devices, so we consider it acceptable to improve the performance with the costs of more training resources, as long as the deployed model keeps the same size.

In this paper, we seek to insert complicated structures into numerous ConvNet architectures to improve the performance while keeping the original inference-time costs. To this end, we decouple the training-time and inference-time network structure by *complicating the model only during training* and converting it back into the original structure for inference. Naturally, we require such extra training-time structures to be **1)** effective in improving the training-time model's performance and **2)** able to transform into the original inference-time structure.

For the usability and universality, we upgrade the basic ConvNet component, $K \times K$ conv, into a powerful block named Diverse Branch Block (DBB) (Fig. 1). As a building block, DBB is complementary to the other efforts to improve ConvNet, *e.g.*, architecture design [12, 24, 13, 21, 23, 35], neural architecture search [2, 39, 22, 20, 19], data augmentation and training methods [25, 5, 33] and fulfills the above two requirements by the following two properties:

*This work is supported by The National Key Research and Development Program of China (No. 2017YFA0700800), the National Natural Science Foundation of China (No.61925107, No.U1936202) and Beijing Academy of Artificial Intelligence (BAAI). Xiaohan Ding is funded by the Baidu Scholarship Program 2019 (http://scholarship.baidu.com/). This work is done during Xiaohan Ding's internship at MEGVII.
†Corresponding author.

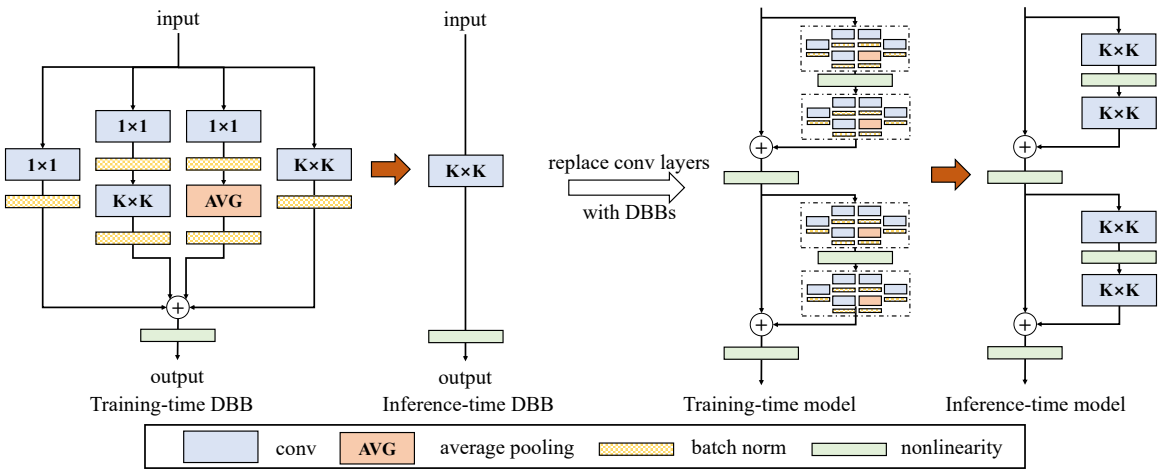


Figure 1: A representative design of Diverse Branch Block (DBB). The block can be equivalently transformed into a regular conv layer for deployment, so that we can complicate the training-time microstructures of ConvNet without affecting the macro architecture (*e.g.*, ResNet) or the inference-time structure. Note that it is only an instance we used, and one may utilize the six transformations summarized in this paper (Sect. 3.2) to customize a DBB with more complicated structures.

- A DBB adopts a multi-branch topology with multi-scale convolutions, sequential 1×1 - $K \times K$ convolutions, average pooling and branch addition. Such operations with various receptive fields and paths with different complexities can enrich the feature space, just like the Inception architectures.
- A DBB can be equivalently transformed into a single conv for inference. Given an architecture, we can replace some regular conv layers with DBB to build more complicated miscrostructures for training, and convert it back into the original structure so that there will be no extra inference-time costs.

More precisely, we do not derive the parameters for inference before each forwarding. Instead, we *convert the model after training once for all*, then we only save and use the resultant model, and the trained model can be discarded. The idea of converting a DBB into a conv can be categorized into *structural re-parameterization*, which means parameterizing a structure with the parameters transformed from another structure, together with a concurrent work [9]. Though a DBB and a regular conv layer have the same inference-time structure, the former has higher representational capacity. Through a series of ablation experiments, we attribute such effectiveness to the *diverse connections* (paths with different scales and complexities) which resembles Inception units, and the *training-time nonlinearity* brought by batch normalization [15]. Compared to some counterparts with duplicate paths or purely linear branches (Fig. 6), DBB shows better performance (Table. 4).

We summarize our contributions as follows.

- We propose to incorporate abundant microstructures into various ConvNet architectures to improve the performance but keep the original macro architecture.
- We propose DBB, a universal building block, and summarize six transformations (Fig. 2) to convert a DBB into a single convolution, so it is cost-free for the users.
- We present a representative Inception-like DBB instance and show that it improves the performance on ImageNet [7] (*e.g.*, up to 1.9% higher top-1 accuracy), COCO detection [18] and Cityscapes [4].

2. Related Work

2.1. Multi-branch Architectures

Inception [15, 26, 27, 28] architectures employed multi-branch structures to enrich the feature space, which proved the significance of diverse connections, various receptive fields and the combination of multiple branches. DBB borrows the idea of using multi-branch topology, but the difference lies in that **1)** DBB is a building block that can be used on numerous architectures, and **2)** each branch of DBB can be converted into a conv, so that the combination of such branches can be merged into a single conv, which is much faster than a real Inception unit. We will show the superiority of diverse branches over duplicate ones (Table. 4), and the most interesting discovery is that combining two branches with different representational capacity (*e.g.*, a 1×1 conv and a 3×3 conv) is better than two strong-capacity branches (*e.g.*, two 3×3 convolutions), which may in turn shed light on ConvNet architecture design.

2.2. ConvNet Components for Better Performance

There have been some novel components to improve ConvNets. For examples, Squeeze-and-Excitation (SE)

block [14] and Efficient Convolutional Attention (ECA) block [29] utilizes the attention mechanism to recalibrate the features, Octave Convolution [3] reduces the spatial redundancy of regular convolution, Deformable Convolution [6] augments the spatial sampling locations with learnable offsets, dilated convolution expands the receptive field [32], BlurPool [34] brings back the shift-invariance, *etc*. DBB is complementary to these components because it only upgrades a fundamental building block: the conv layer.

2.3. Structural Re-parameterization

This paper and a concurrent work, RepVGG [9], are the first to use structural re-parameterization to term the methodology that parameterizes a structure with the parameters transformed from another structure. ExpandNet [10], DO-Conv [1] and ACNet [8] can also be categorized into structural re-parameterization in the sense that they convert a block into a conv. For example, ACNet uses Asymmetric Convolution Block (ACB, as shown in Fig. 6d) to strengthen the skeleton of conv kernel (*i.e*., the crisscross part). Compared to DBB, it is also designed for improving ConvNet without extra inference-time costs. However, the difference is that ACNet was motivated by an observation that the parameters of the skeleton were larger in magnitude and thus sought to make them even larger, whereas we focus on a different perspective. We found out that average pooling, 1×1 conv, and 1×1 - $K \times K$ sequential conv are more effective, as they provide paths with different complexities, and allow the usage of more training-time nonlinearity. Besides, ACB can be viewed as a special case of DBB, since the $1 \times K$ and $K \times 1$ conv layers can be augmented to $K \times K$ via Transform VI (Fig. 2) and merged into the square kernel via Transform II.

2.4. Other ConvNet Re-parameterization Methods

Some works can be referred to as re-parameterization, but not *structural* re-parameterization. For examples, a recent NAS [20, 39] method [2] used meta-kernels to re-parameterize a kernel and supplemented the widths and heights of such meta-kernels into the search space. Soft Conditional Computation (SCC) [31] or CondConv [30] can be viewed as data-dependent kernel re-parameterization, as it generated the weights for multiple kernels of the same shape, then derived a kernel as the weighted sum of all such kernels to participate in the convolution. Note that SCC introduced a significant number of parameters into the deployed model. These re-parameterization methods differ from ACB and DBB in that the former "re-param" means deriving a set of new parameters with some meta parameters (*e.g*., the meta kernels [2]) then using the new parameters for the other computations, while the latter means converting the parameters of a trained model to parameterize another one.

3. Diverse Branch Block

3.1. The Linearity of Convolution

The parameters of a conv layer with C input channels, D output channels and kernel size $K \times K$ reside in the conv kernel, which is a 4th-order tensor $\boldsymbol{F} \in \mathbb{R}^{D \times C \times K \times K}$, and an optional bias $\boldsymbol{b} \in \mathbb{R}^{D}$. It takes a C-channel feature map $\boldsymbol{I} \in \mathbb{R}^{C \times H \times W}$ as input and outputs a D-channel feature map $\boldsymbol{O} \in \mathbb{R}^{D \times H' \times W'}$, where H' and W' are determined by K, padding and stride configurations. We use $\circledast$ to denote the convolution operator, and formulate the bias-adding as replicating the bias $\boldsymbol{b}$ into $\mathrm{REP}(\boldsymbol{b}) \in \mathbb{R}^{D \times H' \times W'}$ and adding it onto the results of convolution. Formally,

$$\boldsymbol{O} = \boldsymbol{I} \circledast \boldsymbol{F} + \mathrm{REP}(\boldsymbol{b}) \,. \tag{1}$$

The value at (h, w) on the j-th output channel is given by

$$\boldsymbol{O}_{j,h,w} = \sum_{c=1}^{C} \sum_{u=1}^{K} \sum_{v=1}^{K} \boldsymbol{F}_{j,c,u,v} \boldsymbol{X}(c,h,w)_{u,v} + \boldsymbol{b}_j \,, \tag{2}$$

where $\boldsymbol{X}(c,h,w) \in \mathbb{R}^{K \times K}$ is the sliding window on the c-th channel of $\boldsymbol{I}$ corresponding to the position (h, w) on $\boldsymbol{O}$. Such a correspondence is determined by the padding and stride. The linearity of conv can be easily derived from Eq. 2, which includes the *homogeneity* and *additivity*,

$$\boldsymbol{I} \circledast (p\boldsymbol{F}) = p(\boldsymbol{I} \circledast \boldsymbol{F}) \,, \forall p \in \mathbb{R} \,, \tag{3}$$

$$\boldsymbol{I} \circledast \boldsymbol{F}^{(1)} + \boldsymbol{I} \circledast \boldsymbol{F}^{(2)} = \boldsymbol{I} \circledast (\boldsymbol{F}^{(1)} + \boldsymbol{F}^{(2)}) \,. \tag{4}$$

Note that the additivity holds only if the two convolutions have the same configurations (*e.g*., number of channels, kernel size, stride, padding, *etc*.), so that they share the same sliding window correspondence X.

3.2. A Convolution for Diverse Branches

In this subsection, we summarize six transformations (Fig. 2) to transform a DBB with batch normalization (BN), branch addition, depth concatenation, multi-scale operations, average pooling and sequences of convolutions.

Transform I: a conv for conv-BN We usually equip a conv with a BN layer, which performs channel-wise normalization and linear scaling. Let j be the channel index, $\boldsymbol{\mu}_j$ and $\boldsymbol{\sigma}_j$ be the accumulated channel-wise mean and standard deviation, $\boldsymbol{\gamma}_j$ and $\boldsymbol{\beta}_j$ be the learned scaling factor and bias term, respectively, the output channel j becomes

$$\boldsymbol{O}_{j,:,:} = ((\boldsymbol{I} \circledast \boldsymbol{F})_{j,:,:} - \boldsymbol{\mu}_j) \frac{\boldsymbol{\gamma}_j}{\boldsymbol{\sigma}_j} + \boldsymbol{\beta}_j \,. \tag{5}$$

The homogeneity of conv enables to fuse BN into the preceding conv for inference. In practice, we simply build a single conv with kernel $\boldsymbol{F}'$ and bias $\boldsymbol{b}'$, assign the values

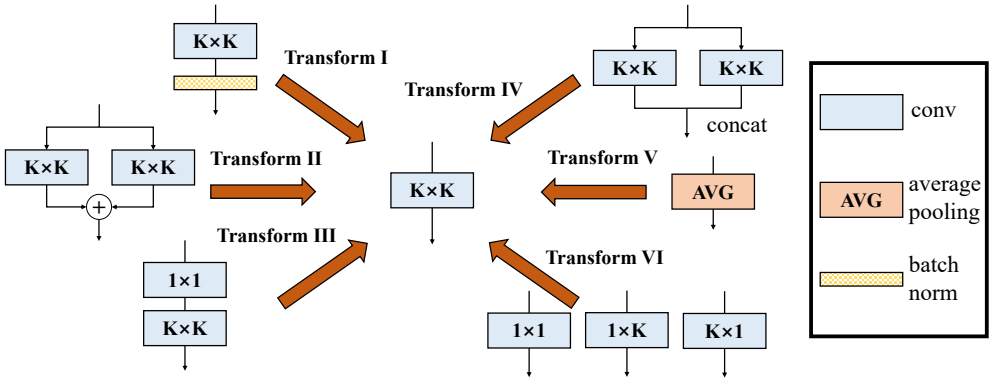


Figure 2: Six transformations we use to implement an inference-time DBB by a regular convolutional layer.

transformed from the parameters of the original conv-BN sequence, then save the model for inference. By Eq. 1, 5, we construct $\boldsymbol{F}'$ and $\boldsymbol{b}'$ for every output channel j as

$$\boldsymbol{F}'_{j,:,:,:} \leftarrow \frac{\boldsymbol{\gamma}_j}{\boldsymbol{\sigma}_j}\boldsymbol{F}_{j,:,:,:}\,, \qquad \boldsymbol{b}'_j \leftarrow -\frac{\boldsymbol{\mu}_j\boldsymbol{\gamma}_j}{\boldsymbol{\sigma}_j} + \boldsymbol{\beta}_j\,. \tag{6}$$

Transform II: a conv for branch addition The additivity ensures that if the outputs of two or more conv layers with the same configurations are added up, we can merge them into a single conv. For a conv-BN, we should perform Transform I first. Obviously, we merge two convolutions by

$$\boldsymbol{F}' \leftarrow \boldsymbol{F}^{(1)} + \boldsymbol{F}^{(2)}\,, \qquad \boldsymbol{b}' \leftarrow \boldsymbol{b}^{(1)} + \boldsymbol{b}^{(2)}\,. \tag{7}$$

The above formulas only apply to conv layers with the same configurations. Though merging such branches can strengthen the model to some extent (Table. 4), we wish to combine diverse branches to further improve the performance. In the following, we introduce some forms of branches that can be equivalently transformed into a single conv. After constructing $K \times K$ conv for each branch via multiple transformations, we use Transform II to merge all such branches into one single conv.

Transform III: a conv for sequential convolutions We can merge a sequence of 1×1 conv - BN - $K \times K$ conv - BN into one single $K \times K$ conv. We temporarily assume the conv is dense (*i.e.*, number of groups $g = 1$). The group-wise case with $g > 1$ will be realized by Transform IV. We assume the kernel shapes of the 1×1 and $K \times K$ layers are $D \times C \times 1 \times 1$ and $E \times D \times K \times K$, respectively, where D can be arbitrary. We first fuse the two BN layers into the two conv layers to obtain $\boldsymbol{F}^{(1)} \in \mathbb{R}^{D\times C\times 1\times 1}$, $\boldsymbol{b}^{(1)} \in \mathbb{R}^D$, $\boldsymbol{F}^{(2)} \in \mathbb{R}^{E\times D\times K\times K}$, and $\boldsymbol{b}^{(2)} \in \mathbb{R}^E$. The output is

$$\boldsymbol{O}' = (\boldsymbol{I} \circledast \boldsymbol{F}^{(1)} + \mathrm{REP}(\boldsymbol{b}^{(1)})) \circledast \boldsymbol{F}^{(2)} + \mathrm{REP}(\boldsymbol{b}^{(2)})\,. \tag{8}$$

We desire the expressions of the kernel and bias of a single conv, $\boldsymbol{F}'$ and $\boldsymbol{b}'$, which satisfies

$$\boldsymbol{O}' = \boldsymbol{I} \circledast \boldsymbol{F}' + \mathrm{REP}(\boldsymbol{b}')\,. \tag{9}$$

Applying the additivity of conv to Eq. 8, we have

$$\boldsymbol{O}' = \boldsymbol{I} \circledast \boldsymbol{F}^{(1)} \circledast \boldsymbol{F}^{(2)} + \mathrm{REP}(\boldsymbol{b}^{(1)}) \circledast \boldsymbol{F}^{(2)} + \mathrm{REP}(\boldsymbol{b}^{(2)})\,. \tag{10}$$

As $\boldsymbol{I} \circledast \boldsymbol{F}^{(1)}$ is 1×1 conv, which performs only channel-wise linear combination but no spatial aggregation, we can merge it into the $K \times K$ conv by linearly recombining the parameters in $K \times K$ kernel. It is easy to verify that such a transformation can be accomplished by transpose conv,

$$\boldsymbol{F}' \leftarrow \boldsymbol{F}^{(2)} \circledast \mathrm{TRANS}(\boldsymbol{F}^{(1)})\,, \tag{11}$$

where $\mathrm{TRANS}(\boldsymbol{F}^{(1)}) \in \mathbb{R}^{C\times D\times 1\times 1}$ is the tensor transposed from $\boldsymbol{F}^{(1)}$. The second term of Eq. 10 is convolutions on constant matrices, so the outputs are also constant matrices. Formally, let $\boldsymbol{P} \in \mathbb{R}^{H\times W}$ be a constant matrix where every entry equals p, $*$ be the 2D conv operator, $\boldsymbol{W}$ be a 2D conv kernel, the result is a constant matrix proportional to p and the sum of all the kernel elements, *i.e.*,

$$(\boldsymbol{P} * \boldsymbol{W})_{:,:} = p\mathrm{SUM}(\boldsymbol{W})\,. \tag{12}$$

Based on this observation, we construct $\hat{\boldsymbol{b}}$ as

$$\hat{\boldsymbol{b}}_j \leftarrow \sum_{d=1}^{D}\sum_{u=1}^{K}\sum_{v=1}^{K} \boldsymbol{b}_d^{(1)}\boldsymbol{F}^{(2)}_{j,d,u,v}\,, 1 \le j \le E\,. \tag{13}$$

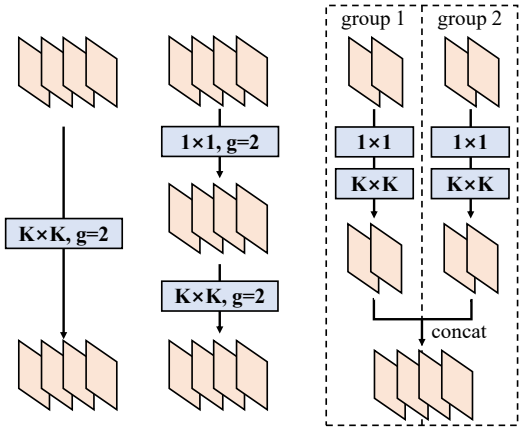
Then it is easy to verify

$$\mathrm{REP}(\boldsymbol{b}^{(1)}) \circledast \boldsymbol{F}^{(2)} = \mathrm{REP}(\hat{\boldsymbol{b}})\,. \tag{14}$$

Then we have

$$\boldsymbol{b}' \leftarrow \hat{\boldsymbol{b}} + \boldsymbol{b}^{(2)}\,. \tag{15}$$

Notably, for a $K \times K$ conv that zero-pads the input, Eq. 8 does not hold because $\boldsymbol{F}^{(2)}$ does not convolve on the result of $\boldsymbol{I} \circledast \boldsymbol{F}^{(1)} + \mathrm{REP}(\boldsymbol{b}^{(1)})$ (but an additional circle of zero pixels). The solution is to either **A)** configure the first conv with padding and the second without, or **B)** pad by $\boldsymbol{b}^{(1)}$. An efficient implementation of the latter is customizing the first BN to **1)** batch-normalize the input as usual, **2)** calculate $\boldsymbol{b}^{(1)}$ (Eq. 6), **3)** pad the batch-normalized result with $\boldsymbol{b}^{(1)}$, *i.e.*, pad every channel j with a circle of $\boldsymbol{b}_j^{(1)}$ instead of 0.



(A) Groupwise conv (B) Training-time 1×1-K×K (C) The perspective from Transform IV

Figure 3: An example of converting 1×1 - $K \times K$ sequence with the number of groups $g > 1$. Assume the input and output are 4-channel feature maps and $g = 2$, the 1×1 and $K \times K$ layers should be configured with $g = 2$, too. For the transformation, we split the layers into g groups, perform Transform III separately, and Transform IV to concatenate the resultant kernels and biases.

Transform IV: a conv for depth concatenation Inception units use depth concatenation to combine branches. But when such branches each contain only one conv with the same configurations, the depth concatenation is equivalent to a conv with a kernel concatenated along the axis differentiating the output channels (*e.g.*, the first axis in our formulation). Given $\boldsymbol{F}^{(1)} \in \mathbb{R}^{D_1 \times C \times K \times K}$, $\boldsymbol{b}^{(1)} \in \mathbb{R}^{D_1}$, $\boldsymbol{F}^{(2)} \in \mathbb{R}^{D_2 \times C \times K \times K}$, $\boldsymbol{b}^{(2)} \in \mathbb{R}^{D_2}$, we concatenate them into $\boldsymbol{F}' \in \mathbb{R}^{(D_1+D_2) \times C \times K \times K}$, $\boldsymbol{b}' \in \mathbb{R}^{D_1+D_2}$. Obviously,

$$\begin{aligned} &\text{CONCAT}(\boldsymbol{I} \circledast \boldsymbol{F}^{(1)} + \text{REP}(\boldsymbol{b}^{(1)}), \boldsymbol{I} \circledast \boldsymbol{F}^{(2)} + \text{REP}(\boldsymbol{b}^{(2)})) \\ &= \boldsymbol{I} \circledast \boldsymbol{F}' + \text{REP}(\boldsymbol{b}') \,. \end{aligned} \tag{16}$$

Transform IV is especially useful for generalizing Transform III to the groupwise case. Intuitively, a groupwise conv splits the input into g parallel groups, convolves separately, then concatenates the outputs. To replace a g-group conv, we build a DBB where all the conv layers have the same groups g. For converting the 1×1 - $K \times K$ sequence, we equivalently split it into g groups, perform Transform III separately, and concatenate the outputs (Fig. 3).

Transform V: a conv for average pooling An average pooling with kernel size K and stride s applied to C channels is equivalent to a conv with the same K and s. Such a kernel $\boldsymbol{F}' \in \mathbb{R}^{C \times C \times K \times K}$ is constructed by

$$\boldsymbol{F}'_{d,c,:,:} = \begin{cases} \dfrac{1}{K^2} & \text{if } d = c \,, \\ 0 & \text{elsewise} \,. \end{cases} \tag{17}$$

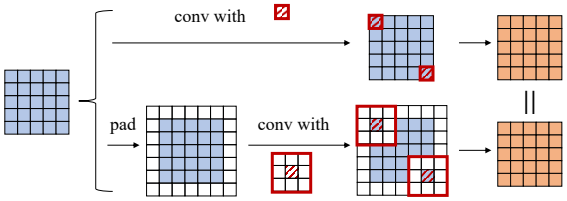


Figure 4: An example of converting a 1×1 layer to 3×3 via Transform VI. To align the starting and ending points of sliding windows (shown at the top-left and bottom-right corners), the 3×3 layer should pad the input by one pixel.

Just like a common average pooling, it performs downsampling when $s > 1$ but is actually smoothing when $s = 1$.

Transform VI: a conv for multi-scale convolutions Considering a $k_h \times k_w$ ($k_h \leq K, k_w \leq K$) kernel is equivalent to a $K \times K$ kernel with some zero entries, we can transform a $k_h \times k_w$ kernel into $K \times K$ via zero-padding. Specifically, 1×1, $1 \times K$ and $K \times 1$ conv are particularly practical as they can be efficiently implemented. The input should be padded to align the sliding windows (Fig. 4).

3.3. An Inception-like DBB Instance

We present a representative instance of DBB (Fig. 1), while its universality and flexibility enable numerous feasible instances. Like Inception, we use 1×1, 1×1 - $K \times K$, 1×1 - AVG to enhance the original $K \times K$ layer. For the 1×1 - $K \times K$ branch, we set the internal channels equal to the input and initialize the 1×1 kernel as an identity matrix. The other conv kernels are initialized regularly [11]. A BN follows every conv or AVG layer, which provides training-time nonlinearity. Without such nonlinearity, the performance gain will be marginal (Table. 4). Notably, for a depthwise DBB, every conv shall have the same number of groups, and we remove the 1×1 path and the 1×1 conv in the 1×1 - AVG path because 1×1 depthwise conv is just a linear scaling.

4. Experiments

We use several benchmark architectures on CIFAR [16], ImageNet [7], Cityscapes [4] and COCO detection [18] to evaluate the capability of DBB for improving ConvNet performance, and then investigate the significance of diverse connections and training-time nonlinearity.

4.1. Datasets, Architectures and Configurations

We first summarize the experimental configurations (Table. 1). On **CIFAR-10/100**, we adopt the standard data augmentation techniques [12]: padding to 40×40, random cropping and left-right flipping. We use VGG-16 [24] for a quick sanity check. Following ACNet [8], we replace the two hidden fully-connected (FC) layers by global average

Table 1: Experimental configurations.

Dataset	Architecture	GPUs	Epochs / iterations	Batch size	Init learn rate	Weight decay	Data augmentation
CIFAR-10/100	VGG-16	1	600 epochs	128	0.1	1×10^{-4}	crop + flip
ImageNet	AlexNet	4	90 epochs	512	0.1	5×10^{-4}	crop + flip
ImageNet	MobileNet	8	90 epochs	256	0.1	4×10^{-5}	crop + flip
ImageNet	ResNet-18/50	8	120 epochs	256	0.1	1×10^{-4}	+ color jitter + PCA lighting [17]
COCO detection	CenterNet [38]	8	126k iters	128	0.02	1×10^{-4}	+ color jitter + PCA lighting [17]
Cityscapes	PSPNet [37]	8	200 epochs	16	0.01	1×10^{-4}	same as [36]

Table 2: Top-1 accuracy of the original model, ACNet [8] and DBB-Net. The results on CIFAR are average of 5 runs

Dataset	Architecture	Original	ACNet	DBB-Net	Accuracy ↑
CIFAR-10	VGG-16	93.95±0.03	94.43±0.03	**94.62**±0.02	0.67
CIFAR-100	VGG-16	74.05±0.10	75.30±0.04	**75.72**±0.07	1.67
ImageNet	AlexNet	57.23	58.43	**59.19**	1.96
ImageNet	MobileNet	71.89	72.14	**72.88**	0.99
ImageNet	ResNet-18	69.54	70.53	**70.99**	1.45
ImageNet	ResNet-50	76.14	76.46	**76.71**	0.57

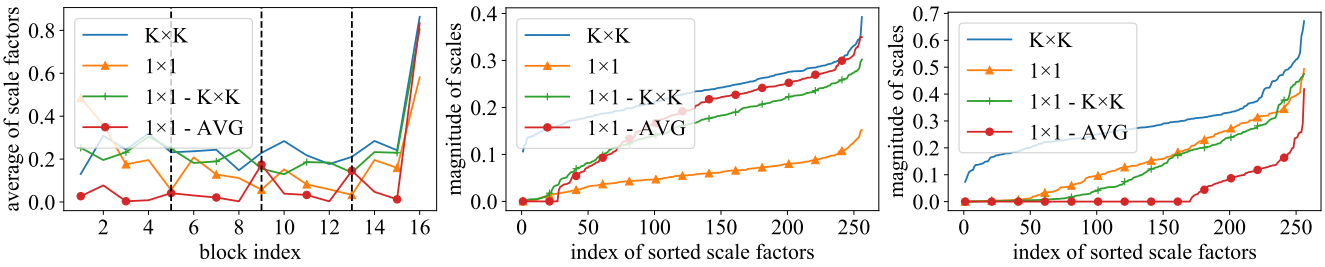
pooling followed by one FC of 512 neurons. For the fair comparison, we equip each conv layer in the original models of VGG with BN. Then we use **ImageNet-1K**, which comprises 1.28M images for training and 50K for validation. For the data augmentation, we employ the standard pipeline including random cropping, left-right flipping for small models like AlexNet [17] and MobileNet [13], and additional color jitter and a PCA-based lighting for ResNet-18/50 [12]. Specifically, we use the same AlexNet as ACNet [8], which is composed of five stacked conv layers followed by three FC layers with no local response normalizations. We insert BN after each conv layer as well. For the simplicity, we use cosine learning rate decay on CIFAR and ImageNet with an initial value of 0.1. On **COCO detection**, we train CenterNet [38] in 126k iterations with a learning rate initialized as 0.02 and multiplied by 0.1 at the 81k and 108k iterations respectively. On **Cityscapes**, we simply adopt the official implementation and default configurations [36] of PSPNet [37] for the better reproducibility: poly learning rate with base of 0.01 and power of 0.9 for 200 epochs.

For each architecture, we replace every $K \times K\,(1 < K < 7)$ conv and its following BN by a DBB to construct a DBB-Net. We do not experiment with larger kernels (*e.g.*, the first 7×7 and 11×11 conv of ResNet and AlexNet) because they are less favored in model architectures. All the models are trained with identical configurations. After training, the DBB-Nets are converted into the same structure as the original model and tested. All the experiments are accomplished with PyTorch.

4.2. DBB for Free Improvements

Table. 2 shows that the DBB-Nets exhibit a clear and consistent boost of performance on CIFAR and ImageNet: DBB improves VGG-16 on CIFAR-10 and CIFAR-100 by 0.67% and 1.67%, AlexNet on ImageNet by 1.96%, MobileNet by 0.99%, and ResNet-18/50 by 1.45%/0.57%, respectively. Even though ACB [8] (Fig. 6d) is a special case of DBB, we still choose it as a competitor to compare with. Concretely, we add $K \times 1$ and $1 \times K$ branches to construct ACBs, and train with the same settings. The superiority of DBB-Net over ACNet suggests that combining paths with Inception-like different complexities may benefit the model more than aggregating features generated by multi-scale convolutions. Notably, the comparisons are biased towards the original models, as we adopt the hyper-parameters reported in the original papers (*e.g.*, weight decay of 10^{-4} on ResNets), which have been tuned on the original models but may be less suitable for the DBB-Nets.

We continue to verify the significance of every branch by showing the scaling factors γ of the four BN layers before the addition. Specifically, for each of the 16 3×3 DBBs of ResNet-18 (because it originally has 16 3×3 conv layers), we compute the average of absolute value of the four scaling vectors. Table. 5a shows that the $K \times K$, 1×1 and $1 \times 1 - K \times K$ branches have comparable magnitude of scaling factors, suggesting that the three branches are important. An interesting discovery is that the 1×1 - AVG branch is more important for a stride-2 DBB, suggesting the average pooling is more useful as downsampling than smoothing. Fig. 5b shows the absolute values of scaling factors of the 9th block with the four γ vectors respectively sorted for the better readability. It is observed that the $K \times K$ branch have a larger minimum scale, and the scales of the other three branches have a wide range. The phenomenon is quite different for the 10th block (Fig. 5c), which has stride=1: the scales of 1×1 - AVG branch are close to zero for more



(a) Average magnitude of scale factors. (b) Sorted scale factors of stride-2 layer. (c) Sorted scale factors of stride-1 layer.

Figure 5: Left: the average magnitude of scaling factors of BN in every DBB across different layers. Vertical dashed lines indicate the stage transition with stride-2 DBB. Middle and right: the magnitude of scaling factors sorted in ascending order of the 9th DBB (stride-2) and the 10th DBB (stride-1).

Table 3: Object detection and semantic segmentation.

Backbone	ImageNet top-1	COCO AP	Cityscapes mIoU
Original Res18	69.54	29.83	70.18
DBB-Res18	70.99	30.68	71.35

than 160 channels but relatively large for the others, and the scales of 1×1 branch are larger than the 9th block, which suggests that 1×1 conv is more useful with stride=1. Such a diversity of distributions of scaling factors suggest that the DBB-Net learns a diverse combination of the diverse branches for each block, and the discoveries may shed light on other research areas like architecture design.

4.3. Object Detection and Semantic Segmentation

We use the ImageNet-pretrained ResNet-18 models to verify the generalization performance on object detection and semantic segmentation. Specifically, we build two CenterNets/PSPNets where the only difference is the backbone (the original ResNet-18 or DBB-ResNet-18), load the ImageNet-pretrained (not yet transformed) weights, train on COCO/Cityscapes, perform the transformations and test.

4.4. Ablation Studies

We conduct a series of ablation studies on ResNet-18 to verify the significance of diverse connections and training-time nonlinearity. Specifically, we first ablate some branches from DBB and observe the change in performance, then compare DBB to some counterparts with duplicate branches or purely linear combination of branches, as shown in Fig. 6. For the purely linear counterpart, we use no BN before the branch addition, but the sum passes through BN. Again, all the models are trained from scratch with the same settings as before and converted into the same original structure for testing. We present in Table. 4 the final accuracy and the training costs.

Table. 4 shows that removing any branch degrades the performance, suggesting that every branch matters. It is also observed that using any of the three branches can lift the accuracy to above 70%. Seen from the training-time parameters *vs*. accuracy, one may use a lightweight DBB with only the 1×1 and 1×1 - AVG branches for lower accuracy but more efficient training, if the training resources are limited. The Double/Triple Duplicate blocks also improve the accuracy, but not as much as diverse branches do. We have two especially interesting discoveries when comparing DBB to duplicate blocks with the same number of branches:

- A 1×1 conv can be viewed as a degraded 3×3 conv with many zero entries, which has weaker representational capacity than the latter, but the accuracy is 70.15% for ($K \times K$ + 1×1) and 69.81% for double $K \times K$. In other words, a weak-capacity component plus a strong-capacity one is better than two strong components.
- Similarly, the DBB with ($K \times K$ + 1×1 + (1×1 - AVG)) outperforms triple $K \times K$ (70.40% >70.29%), though the latter has 2.3× training-time parameters as the former, suggesting that the representational capacity of ConvNet is determined by not only the amount of parameters but also the diversity of connections.

To verify if the improvements are due to the different initialization, we construct a baseline (denoted by "baseline + init") by transforming the full-featured DBB-Net right after random initialization, using the resultant weights to initialize a regular ResNet-18, and then training it with the same settings. The final accuracy is 69.67%, which is hardly higher than the baseline with regular initialization, suggesting that initialization is not the key.

We continue to validate the training-time nonlinearity brought by the BN in branches. In the above discussions, we have noticed that even duplicate branches with BN can improve the performance, as such training-time nonlinearity makes the block more powerful than a single conv. When the BN layers are moved from pre- to post-addition, the block (from the input to the branch addition) becomes

Table 4: Top-1 accuracy of ResNet-18 on ImageNet with different blocks. The training speed (batches/second) is recorded on the same machine with eight 1080Ti GPUs. The training-time eval speed (batches/s) is tested with the original (*i.e.*, not yet transformed) model on a single GPU with a batch size of 128. For reference, the parameters and speed of *every inference-time model* (because all the models end up with the same inference-time structure) are 11.68M and 19.95 batches/s.

	Block	Original $K \times K$	1×1	1×1 - $K \times K$	1×1 - AVG	Accuracy	Training param (M)	Training speed	Training-time eval speed
With BN	DBB	1	✓	✓	✓	70.99	26.33	4.06	4.11
	DBB	1		✓	✓	70.36	25.09	4.16	4.30
	DBB	1	✓		✓	70.40	14.18	4.31	6.64
	DBB	1	✓	✓		70.74	25.08	4.21	6.33
	DBB	1	✓			70.15	12.93	4.38	14.2
	DBB	1		✓		70.20	23.84	4.22	7.31
	DBB	1			✓	70.02	12.95	4.33	7.59
	Baseline	1				69.54	11.69	4.44	19.24
	Baseline + init	1				69.67	11.69	4.44	19.24
	Double Duplicate	2				69.81	22.69	4.36	11.04
	Triple Duplicate	3				70.29	33.70	4.20	7.75
Purely Linear	DBB	1	✓	✓	✓	70.12	26.20	-	-
	DBB	1	✓			69.83	12.91	-	-
	Double Duplicate	2				69.59	22.68	-	-

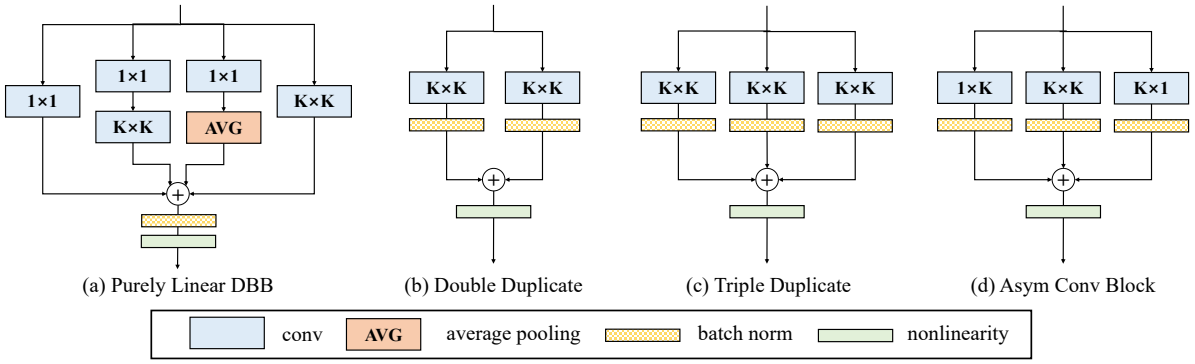


Figure 6: Counterparts to compare against DBB.

purely linear during training. In this case, the Double Duplicate block hardly improves the performance (69.54% $\rightarrow$ 69.59%), and the DBB of ($K \times K$ + 1×1) improves not as much as the comparable DBB with BN (69.83% $<$ 70.15%), suggesting that diverse connections can improve the model even without training-time nonlinearity.

We also present the training speed and the inference speed of the training-time models in Table. 4, which shows that increasing the training-time parameters does not significantly slow down the training speed. Notably, the actual training speed is influenced by the data preprocessing, cross-GPU communication, implementation of backpropagation, *etc*., hence such data are for reference only. In industry, the researchers and engineers usually have abundant training resources but strict restrictions on the inference-time costs, so they may intend to train the models for tens of extra days for very minor performance improvements. In these application scenarios, one may find DBB particularly useful for building powerful ConvNets with only reasonable extra training costs.

5. Conclusions

We proposed a ConvNet building block named DBB, which implements the combination of diverse branches via a single convolution. DBB allows us to improve the performance of off-the-shelf ConvNet architectures with absolutely no extra inference-time costs. Through controlled experiments, we demonstrated the significance of diverse connections and training-time nonlinearity, which make a DBB more powerful than a regular conv layer, though they end up with the same inference-time structure.

References

[1] Jinming Cao, Yangyan Li, Mingchao Sun, Ying Chen, Dani Lischinski, Daniel Cohen-Or, Baoquan Chen, and Changhe Tu. Do-conv: Depthwise over-parameterized convolutional layer. *arXiv preprint arXiv:2006.12030*, 2020. 3

[2] Shoufa Chen, Yunpeng Chen, Shuicheng Yan, and Jiashi Feng. Efficient differentiable neural architecture search with meta kernels. *arXiv preprint arXiv:1912.04749*, 2019. 1, 3

[3] Yunpeng Chen, Haoqi Fan, Bing Xu, Zhicheng Yan, Yannis Kalantidis, Marcus Rohrbach, Shuicheng Yan, and Jiashi Feng. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 3434–3443. IEEE, 2019. 3

[4] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 3213–3223. IEEE Computer Society, 2016. 2, 5

[5] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 113–123, 2019. 1

[6] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 764–773. IEEE Computer Society, 2017. 3

[7] 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

[8] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1911–1920, 2019. 3, 5, 6

[9] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. *arXiv preprint arXiv:2101.03697*, 2021. 2, 3

[10] Shuxuan Guo, Jose M Alvarez, and Mathieu Salzmann. Expandnets: Linear over-parameterization to train compact convolutional networks. *Advances in Neural Information Processing Systems*, 33, 2020. 3

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015. 5

[12] 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, 6

[13] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 1, 6

[14] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(8):2011–2023, 2020. 3

[15] 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, 2

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

[17] 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. 6

[18] 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 *European conference on computer vision*, pages 740–755. Springer, 2014. 2, 5

[19] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 1

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 1, 3

[21] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018. 1

[22] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 1

[23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 1

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

[25] Samarth Sinha, Animesh Garg, and Hugo Larochelle. Curriculum by smoothing. *arXiv e-prints*, pages arXiv–2003, 2020. 1

[26] 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, 2

[27] 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, 2

[28] 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

[29] Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. Eca-net: Efficient channel attention for deep convolutional neural networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 11531–11539. IEEE, 2020. 3

[30] Brandon Yang, Gabriel Bender, Quoc V. Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. 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, 8-14 December 2019, Vancouver, BC, Canada*, pages 1305–1316, 2019. 3

[31] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Soft conditional computation. *arXiv preprint arXiv:1904.04971*, 2019. 3

[32] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 3

[33] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 1

[34] Richard Zhang. Making convolutional networks shift-invariant again. In *International Conference on Machine Learning*, pages 7324–7334. PMLR, 2019. 3

[35] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018. 1

[36] Hengshuang Zhao. Official pspnet. https://github.com/hszhao/semseg, 2020. 6

[37] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6230–6239. IEEE Computer Society, 2017. 6

[38] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019. 6

[39] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 1, 3