Aggregated Residual Transformations for Deep Neural Networks

Saining Xie[1] Ross Girshick[2] Piotr Dollár[2] Zhuowen Tu[1] Kaiming He[2]
[1]UC San Diego [2]Facebook AI Research
{s9xie,ztu}@ucsd.edu {rbg,kaiminghe}@fb.com

Abstract

We present a simple, highly modularized network architecture for image classification. Our network is constructed by repeating a building block that aggregates a set of transformations with the same topology. Our simple design results in a homogeneous, multi-branch architecture that has only a few hyper-parameters to set. This strategy exposes a new dimension, which we call "cardinality" (the size of the set of transformations), as an essential factor in addition to the dimensions of depth and width. On the ImageNet-1K dataset, we empirically show that even under the restricted condition of maintaining complexity, increasing cardinality is able to improve classification accuracy. Moreover, increasing cardinality is more effective than going deeper or wider when we increase the capacity. Our models, co-denamed ResNeXt, are the foundations of our entry to the ILSVRC 2016 classification task in which we secured 2nd place. We further investigate ResNeXt on an ImageNet-5K set and the COCO detection set, also showing better results than its ResNet counterpart. capacity?

1. Introduction

Research on visual recognition is undergoing a transition from "feature engineering" to "network engineering" [24, 23, 43, 33, 35, 37, 13]. In contrast to traditional hand-designed features (*e.g.*, SIFT [28] and HOG [5]), features learned by neural networks from large-scale data [32] require minimal human involvement during training, and can be transferred to a variety of recognition tasks [7, 10, 27]. Nevertheless, human effort has been shifted to designing better network architectures for learning representations.

Designing architectures becomes increasingly difficult with the growing number of hyper-parameters (width[1], filter sizes, strides, *etc.*), especially when there are many layers. The VGG-nets [35] exhibit a simple yet effective strategy of constructing very deep networks: stacking building blocks of the same shape. This strategy is inherited by ResNets [13] which stack modules of the same topology. This simple rule reduces the free choices of hyper-parameters, and depth is exposed as an *essential dimension* in neural networks. Moreover, we argue that the simplicity of this rule may reduce the risk of over-adapting the hyper-parameters to a specific dataset. The robustness of VGG-nets and ResNets has been proven by various visual recognition tasks [7, 10, 9, 27, 30, 13] and by non-visual tasks involving speech [41, 29] and language [4, 40, 19].

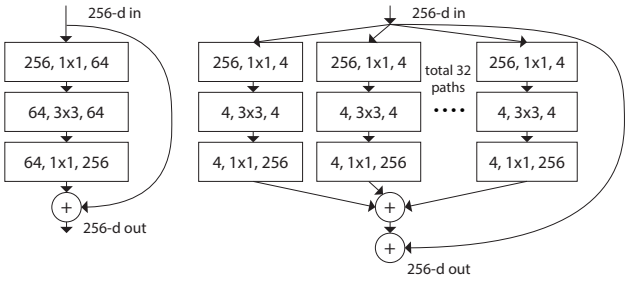

Figure 1. **Left**: A block of ResNet [13]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

Unlike VGG-nets, the family of Inception models [37, 16, 38, 36] have demonstrated that carefully designed topologies are able to achieve compelling accuracy with low theoretical complexity. The Inception models have evolved over time [37, 38], but an important common property is a *split-transform-merge* strategy. In an Inception module, the input is split into a few lower-dimensional embeddings (by 1×1 convolutions), transformed by a set of specialized filters (3×3, 5×5, *etc.*), and merged by concatenation. It can be shown that the solution space of this architecture is a strict subspace of the solution space of a single large layer (*e.g.*, 5×5) operating on a high-dimensional embedding. The split-transform-merge behavior of Inception modules is expected to approach the representational power of large and dense layers, but at a considerably lower computational complexity.

Despite good accuracy, the realization of Inception models has been accompanied with a series of complicating factors — the filter numbers and sizes are tailored for each individual transformation, and the modules are customized

[1]Width refers to the number of channels in a layer.

stage-by-stage. Although careful combinations of these components yield excellent neural network recipes, it is in general unclear how to adapt the Inception architectures to new datasets/tasks, especially when there are many factors and hyper-parameters to be designed.

In this paper, we present a simple architecture which adopts VGG/ResNets' strategy of repeating layers, while exploiting the split-transform-merge strategy in an easy, extensible way. A module in our network performs a set of transformations, each on a low-dimensional embedding, whose outputs are aggregated by summation. We pursuit a simple realization of this idea — the transformations to be aggregated are all of the same topology (*e.g*., Fig. 1 (right)). This design allows us to extend to any large number of transformations without specialized designs.

Interestingly, under this simplified situation we show that our model has two other equivalent forms (Fig. 3). The reformulation in Fig. 3(b) appears similar to the Inception-ResNet module [36] in that it concatenates multiple paths; but our module differs from all existing Inception modules in that all our paths share the same topology and thus the number of paths can be easily isolated as a factor to be investigated. In a more succinct reformulation, our module can be reshaped by Krizhevsky *et al*.'s grouped convolutions [23] (Fig. 3(c)), which, however, had been developed as an engineering compromise.

We empirically demonstrate that our aggregated transformations outperform the original ResNet module, even under the restricted condition of maintaining computational complexity and model size — *e.g*., Fig. 1(right) is designed to keep the FLOPs complexity and number of parameters of Fig. 1(left). We emphasize that while it is relatively easy to increase accuracy by increasing capacity (going deeper or wider), methods that increase accuracy while maintaining (or reducing) complexity are rare in the literature.

Our method indicates that *cardinality* (the size of the set of transformations) is a concrete, measurable dimension that is of central importance, in addition to the dimensions of width and depth. Experiments demonstrate that *increasing cardinality is a more effective way of gaining accuracy than going deeper or wider*, especially when depth and width starts to give diminishing returns for existing models.

Our neural networks, codenamed *ResNeXt* (suggesting the *next* dimension), outperform ResNet-101/152 [13], ResNet-200 [14], Inception-v3 [38], and Inception-ResNet-v2 [36] on the ImageNet classification dataset. In particular, a 101-layer ResNeXt is able to achieve better accuracy than ResNet-200 [14] but has only 50% complexity. Moreover, ResNeXt exhibits considerably simpler designs than all Inception models. ResNeXt was the foundation of our submission to the ILSVRC 2016 classification task, in which we secured second place. This paper further evaluates ResNeXt on a larger ImageNet-5K set and the COCO object detection dataset [26], showing consistently better accuracy than its ResNet counterparts. We expect that ResNeXt will also generalize well to other visual (and non-visual) recognition tasks. The code and models will be made publicly available.

2. Related Work

Multi-branch convolutional networks. The Inception models [37, 16, 38, 36] are successful multi-branch architectures where each branch is carefully customized. ResNets [13] can be thought of as two-branch networks where one branch is the identity mapping. Deep neural decision forests [21] are tree-patterned multi-branch networks with learned splitting functions.

Grouped convolutions. The use of grouped convolutions dates back to the AlexNet paper [23], if not earlier. The motivation given by Krizhevsky *et al*. [23] is for distributing the model over two GPUs. Grouped convolutions are supported by Caffe [18], Torch [3], and other libraries, mainly for compatibility of AlexNet. To the best of our knowledge, there has been little evidence on exploiting grouped convolutions to *improve* accuracy. A special case of grouped convolutions is channel-wise convolutions in which the number of groups is equal to the number of channels. Channel-wise convolutions are part of the separable convolutions in [34].

Compressing convolutional networks. Decomposition (at spatial [6, 17] and/or channel [6, 20, 15] level) is a widely adopted technique to reduce redundancy of deep convolutional networks and accelerate/compress them. Ioannou *et al*. [15] present a "root"-patterned network for reducing computation, and branches in the root are realized by grouped convolutions. These methods [6, 17, 20, 15] have shown elegant compromise of accuracy with lower complexity and smaller model sizes. Instead of compression, our method is an architecture that empirically shows stronger representational power.

Ensembling. Averaging a set of independently trained networks is an effective solution to improving accuracy [23], widely adopted in recognition competitions [32]. Veit *et al*. [39] interpret a single ResNet as an ensemble of shallower networks, which results from ResNet's *additive* behaviors [14]. Our method harnesses additions to aggregate a set of transformations. But we argue that it is imprecise to view our method as ensembling, because the members to be aggregated are trained jointly, not independently.

3. Method

3.1. Template

We adopt a highly modularized design following VGG/ResNets. Our network consists of a stack of residual blocks. These blocks have the same topology, and are

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix}\times6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5**×10^6	**25.0**×10^6
FLOPs		**4.1**×10^9	**4.2**×10^9

Table 1. (**Left**) ResNet-50. (**Right**) ResNeXt-50 with a 32×4d template (using the reformulation in Fig. 3(c)). Inside the brackets are the shape of a residual block, and outside the brackets is the number of stacked blocks on a stage. "C=32" suggests grouped convolutions [23] with 32 groups. *The numbers of parameters and FLOPs are similar between these two models.*

share a prameter?

subject to two simple rules inspired by VGG/ResNets: (i) if producing spatial maps of the same size, the blocks share the same hyper-parameters (width and filter sizes), and (ii) each time when the spatial map is downsampled by a factor of 2, the width of the blocks is multiplied by a factor of 2. The second rule ensures that the computational complexity, in terms of FLOPs (floating-point operations, in # of multiply-adds), is roughly the same for all blocks.

With these two rules, we only need to design a *template* module, and all modules in a network can be determined accordingly. So these two rules greatly narrow down the design space and allow us to focus on a few key factors. The networks constructed by these rules are in Table 1.

3.2. Revisiting Simple Neurons

The simplest neurons in artificial neural networks perform inner product (weighted sum), which is the elementary transformation done by fully-connected and convolutional layers. Inner product can be thought of as a form of aggregating transformation:

$$\sum_{i=1}^{D} w_i x_i, \tag{1}$$

where $\mathbf{x} = [x_1, x_2, ..., x_D]$ is a D-channel input vector to the neuron and w_i is a filter's weight for the i-th channel. This operation (usually including some output nonlinearity) is referred to as a "neuron". See Fig. 2.

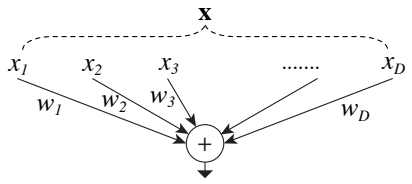

Figure 2. A simple neuron that performs inner product.

The above operation can be recast as a combination of *splitting, transforming, and aggregating*. (i) *Splitting*: the vector $\mathbf{x}$ is sliced as a low-dimensional embedding, and in the above, it is a single-dimension subspace x_i. (ii) *Transforming*: the low-dimensional representation is transformed, and in the above, it is simply scaled: $w_i x_i$. (iii) *Aggregating*: the transformations in all embeddings are aggregated by $\sum_{i=1}^{D}$.

3.3. Aggregated Transformations

Given the above analysis of a simple neuron, we consider replacing the elementary transformation ($w_i x_i$) with a more generic function, which in itself can also be a network. In contrast to "Network-in-Network" [25] that turns out to increase the dimension of depth, we show that our "Network-in-*Neuron*" expands along a new dimension.

Formally, we present aggregated transformations as:

$$\mathcal{F}(\mathbf{x}) = \sum_{i=1}^{C} \mathcal{T}_i(\mathbf{x}), \tag{2}$$

where $\mathcal{T}_i(\mathbf{x})$ can be an arbitrary function. Analogous to a simple neuron, $\mathcal{T}_i$ should project $\mathbf{x}$ into an (optionally low-dimensional) embedding and then transform it.

In Eqn.(2), C is the size of the set of transformations to be aggregated. We refer to C as *cardinality* [2]. In Eqn.(2) C is in a position similar to D in Eqn.(1), but C need not equal D and can be an arbitrary number. While the dimension of width is related to the number of simple transformations (inner product), we argue that the dimension of cardinality controls the number of more complex transformations. We show by experiments that cardinality is an essential dimension and can be more effective than the dimensions of width and depth.

In this paper, we consider a simple way of designing the transformation functions: all $\mathcal{T}_i$'s have the same topology. This extends the VGG-style strategy of repeating layers of the same shape, which is helpful for isolating a few factors and extending to any large number of transformations. We set the individual transformation $\mathcal{T}_i$ to be the bottleneck-shaped architecture [13], as illustrated in Fig. 1 (right). In this case, the first 1×1 layer in each $\mathcal{T}_i$ produces the low-dimensional embedding.

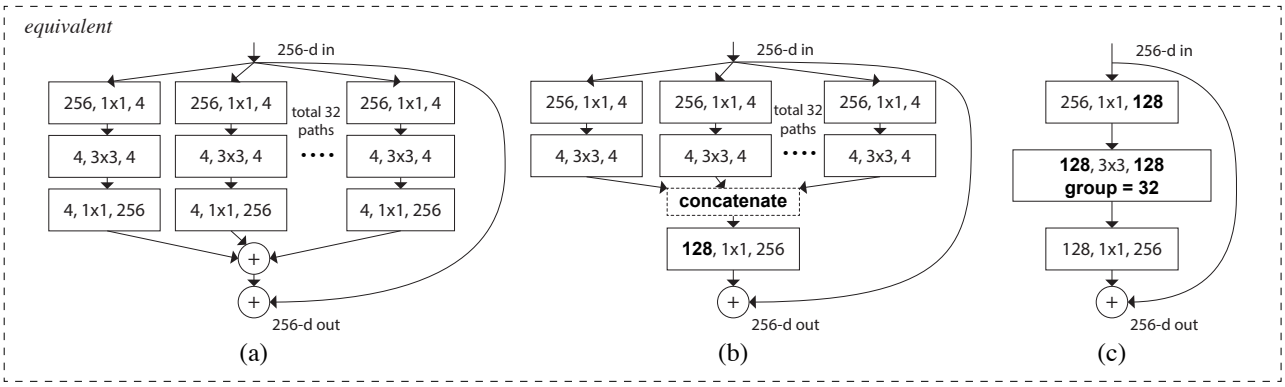

Figure 3. Equivalent building blocks of ResNeXt. **(a)**: Aggregated residual transformations, the same as Fig. 1 right. **(b)**: A block equivalent to (a), implemented as early concatenation. **(c)**: A block equivalent to (a,b), implemented as grouped convolutions [23]. Notations in **bold** text highlight the reformulation changes. A layer is denoted as (# input channels, filter size, # output channels).

The aggregated transformation in Eqn.(2) serves as the residual function [13] (Fig. 1 right):

$$\mathbf{y} = \mathbf{x} + \sum_{i=1}^{C} \mathcal{T}_i(\mathbf{x}), \tag{3}$$

where $\mathbf{y}$ is the output.

Relation to Inception-ResNet. Some tensor manipulations show that the module in Fig. 1(right) (also shown in Fig. 3(a)) is equivalent to Fig. 3(b).[2] Fig. 3(b) appears similar to the Inception-ResNet [36] block in that it involves branching and concatenating in the residual function. But unlike all Inception or Inception-ResNet modules, we share the same topology among the multiple paths. Our module requires minimal extra effort designing each path.

Relation to Grouped Convolutions. The above module becomes more succinct using the notation of *grouped convolutions* [23].[3] This reformulation is illustrated in Fig. 3(c). All the low-dimensional embeddings (the first 1×1 layers) can be replaced by a single, wider layer (*e.g.*, 1×1, 128-d in Fig 3(c)). Splitting is essentially done by the grouped convolutional layer when it divides its input channels into groups. The grouped convolutional layer in Fig. 3(c) performs 32 groups of convolutions whose input and output channels are 4-dimensional. The grouped convolutional layer concatenates them as the outputs of the layer. The block in Fig. 3(c) looks like the original bottleneck residual block in Fig. 1(left), except that Fig. 3(c) is a wider but sparsely connected module.

[2]An informal but descriptive proof is as follows. Note the equality: $A_1B_1 + A_2B_2 = [A_1, A_2][B_1; B_2]$ where $[\ ,\ ]$ is horizontal concatenation and $[\ ;\ ]$ is vertical concatenation. Let A_i be the weight of the last layer and B_i be the output response of the second-last layer in the block. In the case of $C = 2$, the element-wise addition in Fig. 3(a) is $A_1B_1 + A_2B_2$, the weight of the last layer in Fig. 3(b) is $[A_1, A_2]$, and the concatenation of outputs of second-last layers in Fig. 3(b) is $[B_1; B_2]$.

[3]In a group conv layer [23], input and output channels are divided into C groups, and convolutions are separately performed within each group.

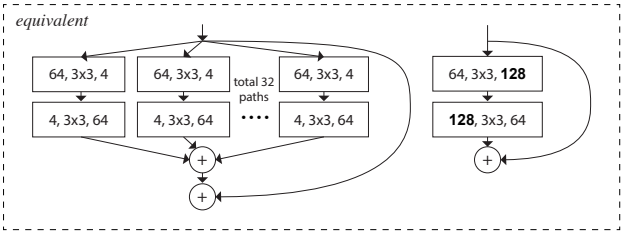

Figure 4. (**Left**): Aggregating transformations of depth $= 2$. (**Right**): An equivalent block, which is trivially wider.

We note that the reformulations produce nontrivial topologies only when the block has depth ≥ 3. If the block has depth $= 2$ (*e.g.*, the basic block in [13]), the reformulations lead to trivially a wide, dense module. See the illustration in Fig. 4.

Discussion. We note that although we present reformulations that exhibit concatenation (Fig. 3(b)) or grouped convolutions (Fig. 3(c)), such reformulations are not always applicable for the general form of Eqn.(3), *e.g.*, if the transformation $\mathcal{T}_i$ takes arbitrary forms and are heterogenous. We choose to use homogenous forms in this paper because they are simpler and extensible. Under this simplified case, grouped convolutions in the form of Fig. 3(c) are helpful for easing implementation.

3.4. Model Capacity

Our experiments in the next section will show that our models improve accuracy when maintaining the model complexity and number of parameters. This is not only interesting in practice, but more importantly, the complexity and number of parameters represent inherent capacity of models and thus are often investigated as fundamental properties of deep networks [8].

When we evaluate different cardinalities C while preserving complexity, we want to minimize the modification

cardinality C	1	2	4	8	32
width of bottleneck d	64	40	24	14	4
width of group conv.	64	80	96	112	128

Table 2. Relations between cardinality and width (for the template of conv2), with roughly preserved complexity on a residual block. The number of parameters is $\sim$70k for the template of conv2. The number of FLOPs is $\sim$0.22 billion (# params$\times$56$\times$56 for conv2).

of other hyper-parameters. We choose to adjust the width of the bottleneck (*e.g.*, 4-d in Fig 1(right)), because it can be isolated from the input and output of the block. This strategy introduces no change to other hyper-parameters (depth or input/output width of blocks), so is helpful for us to focus on the impact of cardinality.

In Fig. 1(left), the original ResNet bottleneck block [13] has $256 \cdot 64 + 3 \cdot 3 \cdot 64 \cdot 64 + 64 \cdot 256 \approx 70$k parameters and proportional FLOPs (on the same feature map size). With bottleneck width d, our template in Fig. 1(right) has:

$$C \cdot (256 \cdot d + 3 \cdot 3 \cdot d \cdot d + d \cdot 256) \tag{4}$$

parameters and proportional FLOPs. When $C = 32$ and $d = 4$, Eqn.(4) $\approx$ 70k. Table 2 shows the relationship between cardinality C and bottleneck width d.

Because we adopt the two rules in Sec. 3.1, the above approximate equality is valid between a ResNet bottleneck block and our ResNeXt on all stages (except for the subsampling layers where the feature maps size changes). Table 1 compares the original ResNet-50 and our ResNeXt-50 that is of similar capacity.[4] We note that the complexity can only be preserved approximately, but the difference of the complexity is minor and does not bias our results.

4. Implementation details

Our implementation follows [13] and the publicly available code of `fb.resnet.torch` [11]. On the ImageNet dataset, the input image is 224$\times$224 randomly cropped from a resized image using the scale and aspect ratio augmentation of [37] implemented by [11]. The shortcuts are identity connections except for those increasing dimensions which are projections (type B in [13]). Downsampling of conv3, 4, and 5 is done by stride-2 convolutions in the 3$\times$3 layer of the first block in each stage, as suggested in [11]. We use SGD with a mini-batch size of 256 on 8 GPUs (32 per GPU). The weight decay is 0.0001 and the momentum is 0.9. We start from a learning rate of 0.1, and divide it by 10 for three times using the schedule in [11]. We adopt the weight initialization of [12]. In all ablation comparisons, we evaluate the error on the single 224$\times$224 center crop from an image whose shorter side is 256.

[4]The marginally smaller number of parameters and marginally higher FLOPs are mainly caused by the blocks where the map sizes change.

Our models are realized by the form of Fig. 3(c). We perform batch normalization (BN) [16] right after the convolutions in Fig. 3(c).[5] ReLU is performed right after each BN, expect for the output of the block where ReLU is performed after the adding to the shortcut, following [13].

We note that the three forms in Fig. 3 are strictly equivalent, when BN and ReLU are appropriately addressed as mentioned above. We have trained all three forms and obtained the same results. We choose to implement by Fig. 3(c) because it is more succinct and faster than the other two forms.

5. Experiments

5.1. Experiments on ImageNet-1K

We conduct ablation experiments on the 1000-class ImageNet classification task [32]. We follow [13] to construct 50-layer and 101-layer residual networks. We simply replace all blocks in ResNet-50/101 with our blocks.

Notations. Because we adopt the two rules in Sec. 3.1, it is sufficient for us to refer to an architecture by the template. For example, Table 1 shows a ResNeXt-50 constructed by a template with cardinality = 32 and bottleneck width = 4d (Fig. 3). This network is denoted as ResNeXt-50 (**32**$\times$**4d**) for simplicity. We note that the input/output width of the template is fixed as 256-d (Fig. 3), and all widths are doubled each time when the feature map is subsampled (see Table 1).

Cardinality *vs*. Width. We first evaluate the trade-off between cardinality C and bottleneck width, under preserved complexity as listed in Table 2. Table 3 shows the results and Fig. 5 shows the curves of error *vs*. epochs. Comparing with ResNet-50 (Table 3 top and Fig. 5 left), the 32$\times$4d ResNeXt-50 has a validation error of 22.2%, which is **1.7%** lower than the ResNet baseline's 23.9%. With cardinality C increasing from 1 to 32 while keeping complexity, the error rate keeps reducing. Furthermore, the 32$\times$4d ResNeXt also has a much lower *training error* than the ResNet counterpart, suggesting that the gains are *not* from regularization but from *stronger representations*.

Similar trends are observed in the case of ResNet-101 (Fig. 5 right, Table 3 bottom), where the 32$\times$4d ResNeXt-101 outperforms the ResNet-101 counterpart by 0.8%. Although this improvement of validation error is smaller than that of the 50-layer case, the improvement of training error is still big (20% for ResNet-101 and 16% for 32$\times$4d ResNeXt-101, Fig. 5 right). In fact, more training data will enlarge the gap of validation error, as we show on an ImageNet-5K set in the next subsection.

[5]With BN, for the equivalent form in Fig. 3(a), BN is employed after aggregating the transformations and before adding to the shortcut.

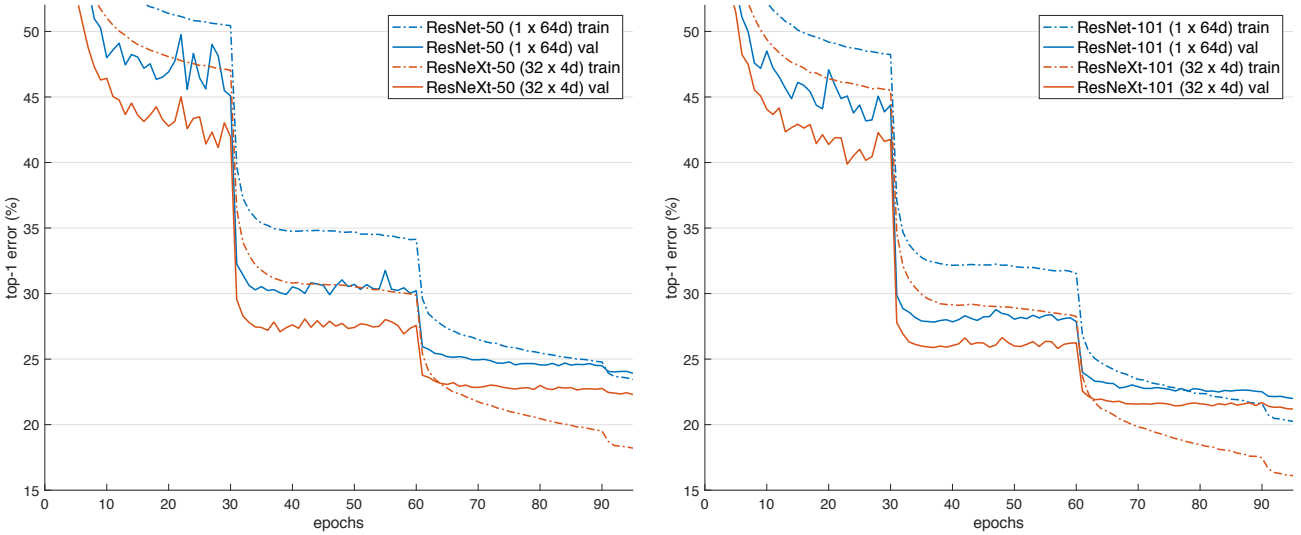

Figure 5. Training curves on ImageNet-1K. (**Left**): ResNet/ResNeXt-50 with preserved complexity (∼4.1 billion FLOPs, ∼25 million parameters); (**Right**): ResNet/ResNeXt-101 with preserved complexity (∼7.8 billion FLOPs, ∼44 million parameters).

	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

Table 3. Ablation experiments on ImageNet-1K. (**Top**): ResNet-50 with preserved complexity (∼4.1 billion FLOPs); (**Bottom**): ResNet-101 with preserved complexity (∼7.8 billion FLOPs). The error rate is evaluated on the single crop of 224×224 pixels.

Table 3 also suggests that *with complexity preserved*, increasing cardinality at the price of reducing width starts to show saturating accuracy when the bottleneck width is small. We argue that it is not worthwhile to keep reducing width in such a trade-off. So we adopt a bottleneck width no smaller than 4d in the following.

Increasing Cardinality *vs*. Deeper/Wider. Next we investigate increasing complexity by increasing cardinality C or increasing depth or width. The following comparison can also be viewed as with reference to 2× FLOPs of the ResNet-101 baseline. We compare the following variants that have ∼15 billion FLOPs. (i) **Going deeper** to 200 layers. We adopt the ResNet-200 [14] implemented in [11]. (ii) **Going wider** by increasing the bottleneck width. (iii) **Increasing cardinality** by doubling C.

	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-**200** [14]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

Table 4. Comparisons on ImageNet-1K when the number of FLOPs is increased to 2× of ResNet-101's. The error rate is evaluated on the single crop of 224×224 pixels. The highlighted factors are the factors that increase complexity.

Table 4 shows that increasing complexity by 2× consistently reduces error *vs*. the ResNet-101 baseline (22.0%). But the improvement is small when going deeper (ResNet-200, by 0.3%) or wider (wider ResNet-101, by 0.7%).

On the contrary, *increasing cardinality C shows much better results than going deeper or wider*. The 2×64d ResNeXt-101 (*i.e*., doubling C on 1×64d ResNet-101 baseline and keeping the width) reduces the top-1 error by 1.3% to 20.7%. The 64×4d ResNeXt-101 (*i.e*., doubling C on 32×4d ResNeXt-101 and keeping the width) reduces the top-1 error to 20.4%.

We also note that 32×4d ResNet-101 (21.2%) performs better than the deeper ResNet-200 and the wider ResNet-101, even though it has only ∼50% complexity. This again shows that cardinality is a more effective dimension than the dimensions of depth and width.

Residual connections. The following table shows the effects of the residual (shortcut) connections:

	setting	w/ residual	w/o residual
ResNet-50	1 × 64d	23.9	31.2
ResNeXt-50	32 × 4d	**22.2**	26.1

Removing shortcuts from the ResNeXt-50 increases the error by 3.9 points to 26.1%. Removing shortcuts from its ResNet-50 counterpart is much worse (31.2%). These comparisons suggest that the residual connections are helpful for *optimization*, whereas aggregated transformations are stronger *representations*, as shown by the fact that they perform consistently better than their counterparts with or without residual connections.

Performance. For simplicity we use Torch's built-in grouped convolution implementation, without special optimization. We note that this implementation was brute-force and not parallelization-friendly. On 8 GPUs of NVIDIA M40, training 32×4d ResNeXt-101 in Table 3 takes 0.95s per mini-batch, *vs*. 0.70s of ResNet-101 baseline that has similar FLOPs. We argue that this is a reasonable overhead. We expect carefully engineered lower-level implementation (*e.g*., in CUDA) will reduce this overhead. We also expect that the inference time on CPUs will present less overhead. Training the 2×complexity model (64×4d ResNeXt-101) takes 1.7s per mini-batch and 10 days total on 8 GPUs.

Comparisons with state-of-the-art results. Table 5 shows more results of single-crop testing on the ImageNet validation set. In addition to testing a 224×224 crop, we also evaluate a 320×320 crop following [14]. Our results compare favorably with ResNet, Inception-v3/v4, and Inception-ResNet-v2, achieving a single-crop top-5 error rate of 4.4%. In addition, our architecture design is much simpler than all Inception models, and requires considerably fewer hyper-parameters to be set by hand.

ResNeXt is the foundation of our entries to the ILSVRC 2016 classification task, in which we achieved 2nd place. We note that many models (including ours) start to get saturated on this dataset after using multi-scale and/or multi-crop testing. We had a single-model top-1/top-5 error rates of 17.7%/3.7% using the multi-scale dense testing in [13], on par with Inception-ResNet-v2's single-model results of 17.8%/3.7% that adopts multi-scale, multi-crop testing. We had an ensemble result of 3.03% top-5 error on the test set, on par with the winner's 2.99% and Inception-v4/Inception-ResNet-v2's 3.08% [36].

5.2. Experiments on ImageNet-5K

The performance on ImageNet-1K appears to saturate. But we argue that this is not because of the capability of the models but because of the complexity of the dataset. Next we evaluate our models on a larger ImageNet subset that has 5000 categories.

	224×224		320×320 / 299×299	
	top-1 err	top-5 err	top-1 err	top-5 err
ResNet-101 [13]	22.0	6.0	-	-
ResNet-200 [14]	21.7	5.8	20.1	4.8
Inception-v3 [38]	-	-	21.2	5.6
Inception-v4 [36]	-	-	20.0	5.0
Inception-ResNet-v2 [36]	-	-	19.9	4.9
ResNeXt-101 (**64 × 4d**)	20.4	5.3	**19.1**	**4.4**

Table 5. State-of-the-art models on the ImageNet-1K validation set (single-crop testing). The test size of ResNet/ResNeXt is 224×224 and 320×320 as in [14] and of the Inception models is 299×299.

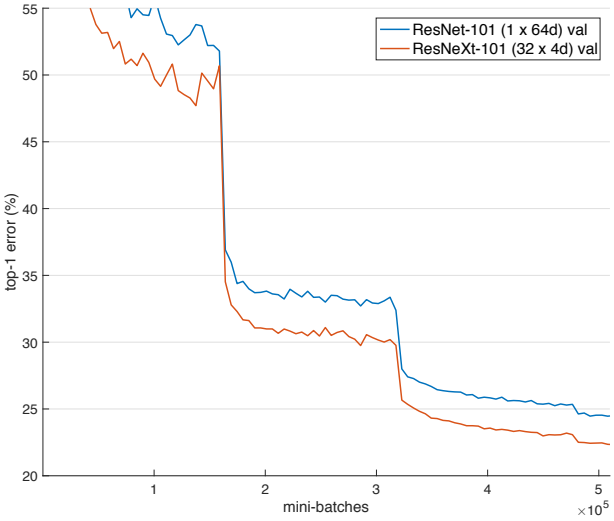

Figure 6. **ImageNet-5K** experiments. Models are trained on the 5K set and evaluated on the original 1K validation set, plotted as a 1K-way classification task. ResNeXt and its ResNet counterpart have similar complexity.

	setting	5K-way classification		1K-way classification	
		top-1	top-5	top-1	top-5
ResNet-50	1 × 64d	45.5	19.4	27.1	8.2
ResNeXt-50	32 × 4d	**42.3**	**16.8**	**24.4**	**6.6**
ResNet-101	1 × 64d	42.4	16.9	24.2	6.8
ResNeXt-101	32 × 4d	**40.1**	**15.1**	**22.2**	**5.7**

Table 6. Error (%) on **ImageNet-5K**. The models are trained on ImageNet-5K and tested *on the ImageNet-1K val set*, treated as a 5K-way classification task or a 1K-way classification task at test time. ResNeXt and its ResNet counterpart have similar complexity. The error is evaluated on the single crop of 224×224 pixels.

Our 5K dataset is a subset of the full ImageNet-22K set [32]. The 5000 categories consist of the original ImageNet-1K categories and additional 4000 categories that have the largest number of images in the full ImageNet set. The 5K set has 6.8 million images, about 5× of the 1K set. There is

no official train/val split available, so we opt to evaluate on the original ImageNet-1K validation set. On this 1K-class val set, the models can be evaluated as a 5K-way classification task (all labels predicted to be the other 4K classes are automatically erroneous) or as a 1K-way classification task (softmax is applied only on the 1K classes) at test time.

The implementation details are the same as in Sec. 4. The 5K-training models are all trained from scratch, and are trained for the same number of mini-batches as the 1K-training models (so 1/5× epochs). Table 6 and Fig. 6 show the comparisons under preserved complexity. ResNeXt-50 reduces the 5K-way top-1 error by **3.2%** comparing with ResNet-50, and ResNetXt-101 reduces the 5K-way top-1 error by **2.3%** comparing with ResNet-101. Similar gaps are observed on the 1K-way error. These demonstrate the stronger representational power of ResNeXt.

Moreover, we find that the models trained on the 5K set (with 1K-way error 22.2%/5.7% in Table 6) perform competitively comparing with those trained on the 1K set (21.2%/5.6% in Table 3), evaluated on the same 1K-way classification task on the validation set. This result is achieved without increasing the training time (due to the same number of mini-batches) and without fine-tuning. We argue that this is a promising result, given that the training task of classifying 5K categories is a more challenging one.

5.3. Experiments on CIFAR

We conduct more experiments on CIFAR-10 and 100 datasets [22]. We use the architectures as in [13] and replace the basic residual block by the bottleneck template of $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}$. Our networks start with a single 3×3 conv layer, followed by 3 stages each having 3 residual blocks, and end with average pooling and a fully-connected classifier (total 29-layer deep), following [13]. We adopt the same translation and flipping data augmentation as [13]. Implementation details are in Appendix A.

We compare two cases of increasing complexity based on the above baseline: (i) *increase cardinality* and fix all widths, or (ii) *increase width* of the bottleneck and fix cardinality $= 1$. We train and evaluate a series of networks under these changes. Fig. 7 shows the comparisons of test error rates *vs*. model sizes. We find that increasing cardinality is more effective than increasing width, consistent to what we have observed on ImageNet-1K. Table 7 shows the results and model sizes, comparing with the Wide ResNet [42] which is the best published record. Our model with a similar model size (34.4M) shows results better than Wide ResNet. Our larger method achieves 3.58% test error (average of 10 runs) on CIFAR-10 and 17.31% on CIFAR-100. To the best of our knowledge, these are the state-of-the-art results (with similar data augmentation) in the literature including unpublished technical reports.

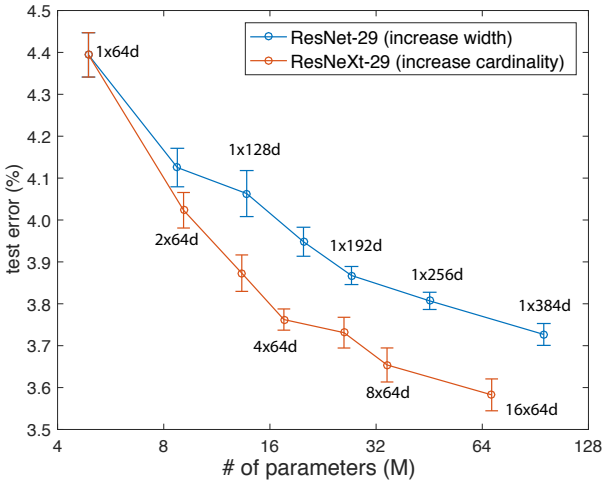

Figure 7. Test error *vs*. model size on CIFAR-10. The results are computed with 10 runs, shown with standard error bars. The labels show the settings of the templates.

	# params	CIFAR-10	CIFAR-100
Wide ResNet [42]	36.5M	4.17	20.50
ResNeXt-29, 8×64d	34.4M	3.65	17.77
ResNeXt-29, 16×64d	68.1M	**3.58**	**17.31**

Table 7. Test error (%) and model size on CIFAR. Our results are the average of 10 runs.

	setting	AP@0.5	AP
ResNet-50	1 × 64d	47.6	26.5
ResNeXt-50	32 × 4d	**49.7**	**27.5**
ResNet-101	1 × 64d	51.1	29.8
ResNeXt-101	32 × 4d	**51.9**	**30.0**

Table 8. Object detection results on the COCO `minival` set. ResNeXt and its ResNet counterpart have similar complexity.

5.4. Experiments on COCO object detection

Next we evaluate the generalizability on the COCO object detection set [26]. We train the models on the 80k training set plus a 35k val subset and evaluate on a 5k val subset (called `minival`), following [1]. We evaluate the COCO-style Average Precision (AP) as well as AP@IoU=0.5 [26]. We adopt the basic Faster R-CNN [31] and follow [13] to plug ResNet/ResNeXt into it. The models are pre-trained on ImageNet-1K and fine-tuned on the detection set. Implementation details are in Appendix B.

Table 8 shows the comparisons. On the 50-layer baseline, ResNeXt improves AP@0.5 by 2.1% and AP by 1.0%, without increasing complexity. ResNeXt shows smaller improvements on the 101-layer baseline. We conjecture that more training data will lead to a larger gap, as observed on the ImageNet-5K set.

A. Implementation Details: CIFAR

We train the models on the 50k training set and evaluate on the 10k test set. The input image is 32×32 randomly cropped from a zero-padded 40×40 image or its flipping, following [13]. No other data augmentation is used. The first layer is 3×3 conv with 64 filters. There are 3 stages each having 3 residual blocks, and the output map size is 32, 16, and 8 for each stage [13]. The network ends with a global average pooling and a fully-connected layer. Width is increased by 2× when the stage changes (downsampling), as in Sec. 3.1. We adopt the pre-activation style block as in [14]. The models are trained on 8 GPUs with a mini-batch size of 256, with a weight decay of 0.0005 and a momentum of 0.9. We start with a learning rate of 0.1 and train the models for 300 epochs, reducing the learning rate at the 150-th and 225-th epoch. Other implementation details are as in [11].

B. Implementation Details: Object Detection

We adopt the Faster R-CNN system [31]. For simplicity we do not share the features between RPN and Fast R-CNN. In the RPN step, we train on 8 GPUs with each GPU holding 2 images per mini-batch and 256 anchors per image. We train the RPN step for 120k mini-batches at a learning rate of 0.02 and next 60k at 0.002. In the Fast R-CNN step, we train on 8 GPUs with each GPU holding 1 image and 64 regions per mini-batch. We train the Fast R-CNN step for 120k mini-batches at a learning rate of 0.005 and next 60k at 0.0005, We use a weight decay of 0.0001 and a momentum of 0.9. Other implementation details are as in `https://github.com/rbgirshick/py-faster-rcnn`.

References

[1] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *CVPR*, 2016.

[2] G. Cantor. *Über unendliche, lineare Punktmannichfaltigkeiten, Arbeiten zur Mengenlehre aus den Jahren, 1872-1884*. 1884.

[3] R. Collobert, S. Bengio, and J. Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.

[4] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun. Very deep convolutional networks for natural language processing. *arXiv:1606.01781*, 2016.

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[6] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

[7] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014.

[8] D. Eigen, J. Rolfe, R. Fergus, and Y. LeCun. Understanding deep architectures using a recursive convolutional network. *arXiv:1312.1847*, 2013.

[9] R. Girshick. Fast R-CNN. In *ICCV*, 2015.

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[11] S. Gross and M. Wilber. Training and investigating Residual Nets. `https://github.com/facebook/fb.resnet.torch`, 2016.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[15] Y. Ioannou, D. Robertson, R. Cipolla, and A. Criminisi. Deep roots: Improving cnn efficiency with hierarchical filter groups. *arXiv:1605.06489*, 2016.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[17] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

[19] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu. Neural machine translation in linear time. *arXiv:1610.10099*, 2016.

[20] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *ICLR*, 2016.

[21] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulò. Deep convolutional neural decision forests. In *ICCV*, 2015.

[22] A. Krizhevsky. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

[23] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[24] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[25] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014.

[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*. 2014.

[27] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[28] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[29] A. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv:1609.03499*, 2016.

[30] P. O. Pinheiro, R. Collobert, and P. Dollar. Learning to segment object candidates. In *NIPS*, 2015.

[31] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

[33] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014.

[34] L. Sifre and S. Mallat. Rigid-motion scattering for texture classification. *arXiv:1403.1687*, 2014.

[35] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[36] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. In *ICLR Workshop*, 2016.

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[38] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[39] A. Veit, M. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow network. In *NIPS*, 2016.

[40] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144*, 2016.

[41] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig. The Microsoft 2016 Conversational Speech Recognition System. *arXiv:1609.03528*, 2016.

[42] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016.

[43] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. In *ECCV*, 2014.