# ResNet or DenseNet? Introducing Dense Shortcuts to ResNet

Chaoning Zhang* chaoningzhang1990@gmail.com
Philipp Benz* pbenz@kaist.ac.kr
Dawit Mureja Argaw
Seokju Lee
Junsik Kim
Francois Rameau
Jean-Charles Bazin
In So Kweon

Korea Advanced Institute of Science and Technology
* Equal Contribution

## Abstract

*ResNet or DenseNet? Nowadays, most deep learning based approaches are implemented with seminal backbone networks, among them the two arguably most famous ones are ResNet and DenseNet. Despite their competitive performance and overwhelming popularity, inherent drawbacks exist for both of them. For ResNet, the identity shortcut that stabilizes training might limit its representation capacity, and DenseNet mitigates it with multi-layer feature concatenation. However, the dense concatenation causes a new problem of requiring high GPU memory and more training time. Partially due to this, it is not a trivial choice between ResNet and DenseNet. This paper provides a unified perspective of dense summation to analyze them, which facilitates a better understanding of their core difference. We further propose dense weighted normalized shortcuts as a solution to the dilemma between them. Our proposed dense shortcut inherits the design philosophy of simple design in ResNet and DenseNet. On several benchmark datasets, the experimental results show that the proposed DSNet achieves significantly better results than ResNet, and achieves comparable performance as DenseNet but requiring fewer computation resources.*

## 1. Introduction

Deep neural networks (DNNs) have achieved state-of-the-art performance in numerous computer vision tasks [17, 38, 15, 31, 59, 58, 13, 24, 35, 25, 26], and the interpretation of DNNs has also be investigated from the lens of visualization [55, 33, 34] as well as robustness [47, 16, 5, 56, 57, 6, 4, 7]. AlexNet [27] and VGGNet [42] have been the pioneering works that demonstrate the potential of DNNs. Inspired by the success of these seminal works, the research focus of the community has shifted from feature engineering [32, 11] to network design engineering and numerous new network architectures have come out to boost the performance of DNNs. ResNets, reusing preceding features with the identity shortcut have achieved state-of-the-art performance on several benchmark datasets, such as ImageNet [12] and COCO detection dataset [30].

One of the reasons that make ResNet exceptionally popular is the simple design strategy which introduces only one identity shortcut. Despite its large success, the weakness of the identity shortcut has been analyzed by follow-up works. The identity shortcut skips the residual blocks to preserve features, and consequently might limit the representation power of the network [54, 59]. The drawback of the identity shortcut is that it causes the collapsing domain problem which reduces the learning capacity of the network [36] and [59] proposed to mitigate it with non-linear shortcuts.

Another simple yet effective technique, dense concatenation, has been proposed in DenseNet [21] to facilitate training deep networks. The DenseNet adopting dense concatenation to all subsequent layers to avoid using direct summation, preserves the features in preceding layers. DenseNet has been shown to have better feature use efficiency, outperforming ResNet with fewer parameters [21]. Nonetheless, DenseNet requires heavy GPU memory due to concatenation operations. The memory issue can be mitigated by memory-efficient implementation introduced in [37]. However, such an implementation is more complex from the engineering perspective, and it also further increases the training time of DenseNet by 20% [37]. The main reason that DenseNet requires more training time is that DenseNet uses many small convolutions in the network, which runs slower on GPU than compact large convolution with the same number of GFLOPS. In short, there is a dilemma in the choice between ResNet and DenseNet for practical applications in terms of the performance and GPU resources.

This paper proposes a dense normalized shortcut as an alternative dense connection technique to mitigate this

dilemma. The proposed dense normalized shortcut outperforms ResNet [17] with a significant margin, with negligible parameter overhead, and it achieves comparable performance as DenseNet but requiring less computation. Our proposed network structure adopts the same backbone (convolutional block design) as ResNet and replaces identity shortcut with our dense normalized shortcut. Our approach uses neither identity shortcut nor dense concatenation. From this perspective, this work is similar to FractalNet [28] whose structural layouts are precisely fractal. FractalNet explored to train deep networks using neither identity shortcut nor dense concatenation, however, its performance is less favourable. The non-linear shortcuts introduced in [59] leads to performance boost. However, without idenity shortcut, its performance decreases when the network is very deep.

Overall, this paper provides one unified perspective of dense summation to analyze ResNet and DenseNet, which facilitates a better understanding of their core differences. Based on this perspective, we propose dense weighted normalized shortcuts to alleviate the drawbacks of the existing two dense connection techniques. We evaluate the proposed DSNet on several benchmark datasets and the results show that it outperforms ResNet by a significant margin and also with fewer parameters achieves comparable (or slightly better) performance as DenseNet but requiring fewer computation resources.

## 2. Related works

Deep CNN network design has become a very hot research topic and numerous techniques contributed to the success of deep learning in the computer vision field. Those techniques can be roughly divided into two categories: micro-module design and macro-architecture design.

### 2.1. Micro-module design

Micro-modules, such as normalization modules [23], attention modules [19], group convolutions [60], and bottleneck design [17], can be inserted into existing macro-architecture networks to improve the performance. Among them, normalization techniques [23] are the most widely used and by default almost all the deep learning models adopt batch normalization [23] to improve the performance and speed up the convergence. The random sampling in batch normalization also contributes to improving the generalization capability of the model. The fast convergence caused by batch normalization is has been linked to the smoothed optimization landscape [40], while internal covariate shift is argued by [1] to be crucial for understanding how batch normalization works. Recently, it has been shown in [9] that batch normalization increases adversarial vulnerability due to shifting the model to rely more on the non-robust features. [8] shows that updating the moving average statistics of batch normalization with a few (32 for instance) corruption representation samples can significantly improve the model corruption robustness. Additionally, the effect of batch normalization on adversarial robustness has been performed in [52, 2].

Alternative normalization techniques, such as weight normalization [39], instance normalization [48] or layer normalization [3], have also been investigated for addressing the dependency between samples during training. Those alternative techniques can also speed up the convergence, however, often do not provide as good performance as batch normalization. Both instance normalization and layer normalization can be seen as a special case of the later proposed group normalization [51]. Similar to instance normalization and layer normalization, group normalization performs the normalization along the channel direction instead of batch direction, enabling it to work effectively in memory-intensive applications where only a small number of samples can be processed in one batch [51]. In previous works, the normalization techniques have been mainly used in the residual path but our work explores the effect of the normalization techniques in the dense shortcut path, somewhat similar to normalization in the non-linear shortcut [59]. We explore the normalization techniques in the proposed dense shortcut path mainly due to their lightweight property.

### 2.2. Macro-architecture design

The goal of macro-architecture is to design a more effective backbone architectures that can be adopted for a wide range of tasks. Famous macro-architectures include AlexNet [27], VGGNet [42], GoogleNet and its variants [45, 46], ResNet [17], and DenseNet [21]. As the pioneering networks in the deep learning field, AlexNet, VGGNet and GoogleNet are still widely used by many researchers to design a prototype network for their applications. However, there is a trend in the research community that ResNet and DenseNet have become more favourable choices due to their competitive performance and simple designs.

ResNet has two famous variants: WideResnet [54] and ResNext [53], which explore the dimensions of width and cardinality respectively. The original ResNet has demonstrated that identity shortcut can contribute to stabilizing the training of deep networks; however, the performance decreases when the network becomes extremely deep (for example, more than 200 layers). Preactivation-ResNet [18] mitigated this problem by re-ordering activations in the ResNet module. The performance gain of preactivation-ResNet over original ResNet could only be observed in extremely deep networks [18]. Later, it has been found that the extreme depth is unnecessary since it provides worse performance compared with the performance by increasing the width of the network with a similar number

of parameters [54]. ResNext further explored the influence of cardinality, which is more effective than increasing either depth or width. One common element in various variants ResNets is the identity shortcut. DenseNet effectively uses the concatenation technique instead of the identity shortcut. Moreover, CondenseNet [20], a variant of DenseNet, exploits the power of dense connection in a more extreme way. One of its most significant differences from the DenseNet is that layers with different resolution feature-maps are also densely connected. Furthermore, Dual path network [10] and mixed link network [49] integrate ResNet and DenseNet into one network by using both identity shortcut and dense concatenation. Our work differentiates from previous works in that our approach adopts neither identity shortcut nor dense concatenation. FractalNet [28] also explored to train ultra-deep networks relying on neither identity shortcut nor dense concatenation; however, it provides less favourable performance. Our work is also very similar to [59] which introduces non-linear shortcuts for improving the performance but still requires identity shortcut when the network is very deep.

## 3. Proposed approach

### 3.1. Background: Dense connection exists in ResNet and DenseNet

What is the difference between ResNet and DenseNet? As the name suggests, it seems that the difference lies in that ResNet only uses one preceding feature-map, while DenseNet uses features of all the preceding convolutional blocks. The shared philosophy that unifies ResNet and DenseNet is that they both connect to the feature-maps of all preceding convolutional blocks [21]. A similar finding has been revealed in [10, 49]. That is to say, dense connection exists in both ResNet and DenseNet [10, 49]. For a typical convolution in DNNs, we formulate it as:

$$f_l = H_l * f_{l-1}, \tag{1}$$

where $f_l$ and $f_{l-1}$ indicate the current feature-map and previous feature-map respectively; "$*$" indicates convolution operation and $H_l$ indicates the convolution weight. For simplicity, we do not take the bias term in the convolution into account. In VGG style network [42], $f_{l-1}$ is only the preceding feature-map. In DenseNet, however, $f_{l-1}$ connects the feature-maps of all preceding convolutional blocks as illustrated in Figure 1 (b):

$$Y_l = X_0/X_1/.../X_l, \tag{2}$$

where $X_i$ represents each of the preceding feature-maps and "$/$" represents the operation of concatenation. Replacing $f_{l-1}$ with above $Y_l$, we get

$$f_l = H_l * (X_0/X_1/.../X_l) \tag{3}$$

DenseNet

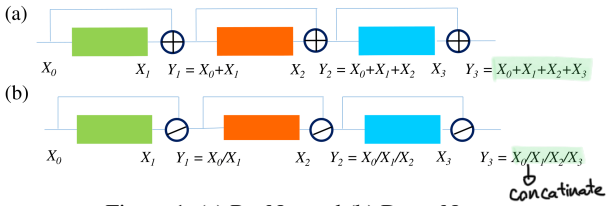


Figure 1. (a) ResNet and (b) DenseNet.

from which it is obvious that DenseNet uses the feature-maps of all preceding convolutional block outputs. In ResNet

$$Y_l = Y_{l-1} + X_l, \tag{4}$$

and it may seem that $Y_l$ only reuses preceding feature-map $Y_{l-1}$. However, as shown in Figure 1 (a) we can recursively extend this function and get

$$Y_l = X_0 + X_1 + ... + X_l, \tag{5}$$

Likewise, we insert the above $Y_l$ to Eq. 1, for ResNet we get

$$f_l = H_l * (X_0 + X_1 + ... + X_l), \tag{6}$$

ResNet

from which it is clear that ResNet also connects to the feature-maps of all preceding convolutional blocks [21, 10, 49]. The difference between ResNet and DenseNet is that ResNet adopts summation to connect all preceding feature-maps while DenseNet concatenates all of them [49].

### 3.2. Unified perspective with dense summation

As analyzed above, DenseNet is different from ResNet because they adopt different dense connection methods: summation vs concatenation. Here, we demonstrate that dense concatenation before convolution operation can be equivalent to dense summation after convolution, thus Eq. 3 can be reformulated as:

$$\begin{aligned} f_l &= H_l * (X_0/X_1/.../X_l) \\ &= (H_l^0/H_l^1/.../H_l^l) * (X_0/X_1/.../X_l) \\ &= H_l^0 * X_0 + H_l^1 * X_1 + \cdots + H_l^l * X_l, \end{aligned} \tag{7}$$

where $H_l = H_l^0/H_l^1/.../H_l^1$ simply dividing one convolution weight $H_l$ into multiple small convolution weights $H_l^i$. Note that $H_l$ indicates the first convolution in the convolutional block instead of the whole convolution block. Thus, $f_l$ is the feature-map after the first convolution in the convolutional block. Overall, the above equivalence is illustrated in Figure 2, where $n_{in}$ and $n_{out}$ are the number of input channels and output channels respectively. Similar observation has been revealed in [53]. For ResNet, we transform Eq. 6 into

$$\begin{aligned} f_l &= H_l * (X_0 + X_1 + ... + X_l) \\ &= H_l * X_0 + H_l * X_1 + \cdots + H_l * X_l. \end{aligned} \tag{8}$$

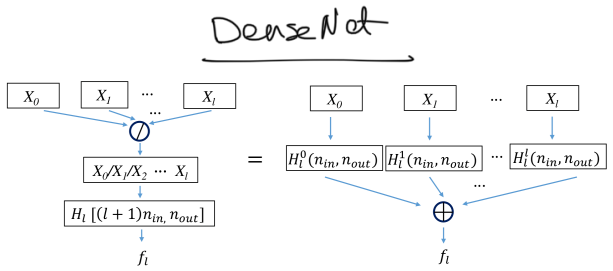


Figure 2. Equivalence of Dense concatenation before convolution and Dense summation after convolution.

We summarize Eq. 7 and Eq. 8 as follows,

$$f_l = H_l^0 * X_0 + H_l^1 * X_1 + \cdots + H_l^l * X_l \text{ for DenseNet,}$$
$$f_l = H_l * X_0 + H_l * X_1 + \cdots + H_l * X_l \text{ for ResNet.} \quad (9)$$

From Eq. 9 we observe that both ResNet and DenseNet have dense summation of $H_l(X_i)$. This interesting observation provides a more unified perspective to perceive ResNet and DenseNet in terms of their resemblance to each other. However, the main purpose of formulating this "unified perspective" is to better understand their core differences for demonstrating their pros and cons. By comparing the two formulas for DenseNet and ResNet in Eq. 9, we find that the core difference lies in that the convolution weight $H_l$ in ResNet is the same for every preceding layer output $X_i$ while $H_l^i$ is different for $X_i$ in DenseNet. This core difference results in other differences in practical use. The input channel $n_{in} * (l+1)$ in $H_l$ increases with the increase of $l$ and it is normally larger than that in $H_l$ of ResNet when $l$ becomes large. Due to the concatenation feature, $n_{out}$ in $H_l$ is normally very small but with more layers. Thus, DenseNet in practical use often requires more training time. Moreover, the concatenation nature resulting in large input channel $n_{in} * (l+1)$ also requires more GPU memory. However, the merit of DenseNet design is that it exhibits more flexibility of using previous feature-maps because each $H_l^i$ is different.

It is worth mentioning that Eq. 9 does not reflect their practical implementation. $H_l * (X_0/X_1/\ldots/X_l)$ is much faster than $H_l^0 * X_0 + H_l^1 * X_1 + \cdots + H_l^{l-1} * X_l$ on GPUs; and $H_l * (X_0 + X_1 + \cdots + X_l)$ is much more efficient than $H_l * X_0 + H_l * X_1 + \cdots + H_l * X_l$. Our above analysis only demonstrates the theoretical connection between ResNet and DenseNet.

### 3.3. Dense shortcut and DSNet

With the above unified perspective, the core difference between ResNet and DenseNet is revealed as whether the convolution parameters $H_l$ are shared for each preceding output. It then results in superior performance of DenseNet with the disadvantage of requiring more GPU resources. The difference originates from the adoption of different dense connection techniques, identity shortcut and dense concatenation. In this paper we propose one alternative dense connection that is motivated to alleviate their drawbacks. It introduces flexibility of using preceding feature-maps while still using the same $H_l$ for each preceding feature-map. Benchmarking ResNet formula in Eq. 9, we propose

$$f_l = H_l * DS_l^0(X_0) + H_l * DS_l^1(X_1) + \ldots + H_l * DS_l^{l-1}(X_{l-1}) + H_l * X_l. \quad (10)$$

which is equivalent to

$$f_l = H_l * (DS_l^0(X_0) + DS_l^1(X_1) + \ldots + DS_l^{l-1}(X_{l-1}) + X_l), \quad (11)$$

where "$DS()$" indicates dense shortcut. It refers to dense weighted normalized shortcut consisting of normalization and channel-wise weight. Specifically, $DS_l^i(X_i) = W_l^i \times N(X_i)$, where $W_l^i$ represents channel-wise weight and and $N$ indicates normalization operation. In this work, the term "dense shortcut" is equivalent to "dense weighted normalized shortcut" unless otherwise specified. Eq. 11 is illustrated in Figure 3 (a); however, we conjecture that the feature-map contains more useful feature in the aggregation output $Y_l$ than its corresponding single convolutional block output $X_l$, thus by replacing $X_l$ with $Y_l$ we propose another variant:

$$f_l = H_l * (DS_l^0(Y_0) + DS_l^1(Y_1) + \ldots + DS_l^{l-1}(Y_{l-1}) + X_l), \quad (12)$$

which is illustrated in Figure 3 (b). The conjecture that the feature of aggregation output $Y_l$ is more meaningful than $X_l$ is supported by the experimental results (see Table 1). Therefore we mainly adopt variant (b) in Figure 3 in this study. We term the proposed network adopting DS shortcut DSNet. DSNet adopts the same network backbone (convolutional block itself and block design) as ResNet [17]. The ResNet backbone is tailored for the identity shortcut, not for our proposed shortcut. It is conceivable that redesigning the backbone structure might further improve the performance of our DSNet, but this is beyond the scope of this work. The only difference between our DSNet and ResNet [17] is to replace identity shortcut with the proposed dense shortcut, *i.e.* the dense weighted normalized shortcut.

For introducing dense shortcuts in ResNet, one naive approach is to just densely connect all preceding feature-maps by replacing single identity shortcut in Eq. 4 with dense identity shortcut, and we get

$$Y_l = Y_{l-1} + Y_{l-2} + ... + Y_0 + X_l, \quad (13)$$

which can be recursively extended as:

$$Y_l = X_l + X_{l-1} + 2X_{l-2} + ... + (l-1)X_1 + lX_0. \quad (14)$$

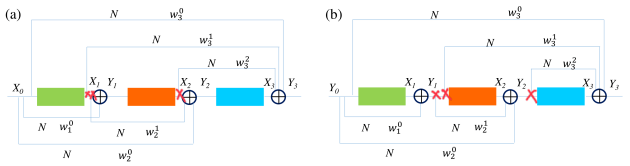

Figure 3. Proposed DSNet adopting dense (weighted normalized) shortcut. (a) Densely connected to $X_l$, (b) Densely connected to $Y_l$.

Comparing it with Eq. 5, we find that dense identity shortcut is equivalent to add extra constant at the end of each convolutional block. Such design denoted ResNet50-dense in Table 1 does not achieve better performance than original ResNet50. This demonstrates the failure of naive dense identity shortcut. Next, we will illustrate our motivation for the design of our DS shortcut.

The motivation of using normalization is to normalize all the preceding features into a similar scale to avoid any preceding feature to dominate the whole summation and facilitate the training. Note that no affine transformation is applied in the normalization process. The weighted summation is to provide the network freedom to assign proper weight to each normalized feature-map depending on its significance. It is cumbersome to manually decide the weights for each one, thus these weights are set to learnable parameters. Moreover, we empirically find that inserting weighted normalized shortcut within the convolutional block on the 3×3 convolution as in [59] can further improve the performance. For differentiation, We term it DS2Net.

### 3.4. Ablation study

We adopt the widely used ResNet50 backbone to do ablation studies. The test is conducted on CIFAR100 and the results are available in Table 1. Note that the width of the network is set to 0.25 times the width of ResNet in [17] to save computation resources. We have two major observations from Table 1. First, normalization techniques are important to improve performance. Specifically, group normalization (GN) outperforms batch normalization (BN) by a visible margin. Instance normalization (IN) and layer normalization (LN) are two special cases of GN. LN performs slightly inferior to GN, and IN performs the worst. Second, the weighted parameters are also critical to improve performance. In particular, we empirically find that adopting channel-wise weight is crucial for the performance gain which indicates that different channels should have different weights. Overall, these two observations support the intuitions of adopting weighted normalized shortcuts. Other observations include the superiority of DS2Net to DSNet and the inferiority of DSNet-a to DSNet.

Table 1. Classification error (%) on CIFAR-100 validation dataset for ablation study, "-a" indicates structure (a) in Figure 3, others by default adopts structure (b) in Figure 3.

| Architecture | Top-1 (%) |
|---|---|
| ResNet50 (0.25×) [17] | 26.13 |
| ResNet50-dense (0.25×) | 26.33 |
| DSNet50-a (GN + weight) (0.25×) | 25.23 |
| DSNet50 (GN + weight) (0.25×) | 24.12 |
| DS2Net50 (GN + weight) (0.25×) | 23.72 |
| DSNet50 (LN + weight) (0.2×) | 24.43 |
| DSNet50 (BN + weight) (0.25×) | 24.90 |
| DSNet50 (IN + weight) (0.25×) | 28.11 |
| DSNet50 (None + weight) (0.25×) | 26.26 |
| DSNet50 (GN + no weight) (0.25×) | 25.85 |

Table 2. Classification error (%) on CIFAR-100 and CIFAR-10 validation dataset for different widths and depths.

| Architecture | CIFAR-100 | CIFAR-10 |
|---|---|---|
| ResNet50 (1×) [17] | 21.43 | 4.82 |
| DSNet50 (1×) | 19.95 | 4.54 |
| DS2Net50 (1×) | 19.00 | 4.33 |
| ResNet50 (0.25×) | 26.13 | 6.65 |
| DSNet50 (0.25×) | 24.12 | 5.95 |
| DS2Net50 (0.25×) | 23.72 | 5.77 |
| ResNet101 (0.25×) | 25.00 | 6.00 |
| DSNet101 (0.25×) | 23.94 | 5.82 |
| DS2Net101 (0.25×) | 23.61 | 5.68 |

## 4. Experimental Results and Analysis

In this section, we conduct experiments across a range of datasets to evaluate the proposed dense shortcut.

### 4.1. CIFAR experiments

We first evaluate it on CIFAR datasets. Both CIFAR-100 and CIFAR-10 contain 60,000 colour images with the resolution of 32 × 32. Totally 50,000 images are used as the training dataset and the remaining 10,000 images as the validation dataset. For the data augmentation, we adopt widely adopted cropping with 4-pixel padding and horizontal flipping. The preprocessing is done with the normalization through the training dataset mean and standard deviations values. For all CIFAR experiments, by default, we set the weight decay to 0.0005 and train for 64k iterations and the initial learning rate is set to 0.1 which is then divided by 10 at 32k and 48k iterations respectively similar to [17]. Since we adopt the same backbone as ResNet and replace the identity shortcut with our dense shortcut, we mainly compare our results with the original ResNets [17].

The results in Table 2 show that our proposed approach outperforms ResNet by a significant margin. On CIFAR-100, DSNet50 (×0.25) outperforms ResNet50 (×0.25) by a margin of 2%. DS2Net further improves the performance of DSNet by a visible margin. The same trend can be observed for both wider (1×) and deeper networks (depth 101). CIFAR-10 mirrors the story and for simplicity in the remainder of the paper, we only report the result for

CIFAR-100. To further evaluate the robustness of the proposed DSNet to different depths and widths, we conduct extra experiments and the results are available in Table 3 and Table 4 respectively.

Table 3. Classification error (%) on CIFAR-100 validation dataset for different depths with the width 0.25×.

| Depth | block design | ResNet | DSNet | DS2Net |
|---|---|---|---|---|
| 26 | [2, 2, 2, 2] | 27.74 | 26.80 | 26.30 |
| 38 | [3, 3, 3, 3] | 27.26 | 25.59 | 25.30 |
| 50 | [3, 4, 6, 3] | 26.13 | 24.12 | 23.72 |
| 77 | [3, 4, 15, 3] | 25.32 | 23.95 | 23.55 |
| 101 | [3, 4, 23, 3] | 25.01 | 23.63 | 23.40 |

Table 4. Classification error (%) on CIFAR-100 validation dataset for different widths with ResNet as the backbone.

| Width | ResNet50 | DSNet50 | DS2Net50 |
|---|---|---|---|
| 0.25 | 26.13 | 24.12 | 23.72 |
| 0.25(WRN) | 23.51 | 22.00 | 21.58 |
| 0.5 | 23.38 | 21.59 | 21.02 |
| 1.0 | 21.43 | 19.95 | 19.00 |

For the depth exploration, we set the width to $0.25\times$ and for the width exploration, we set the depth to 50 layers. We observe that DSNet consistently outperforms ResNet over a wide range of depth and widths. Since it has been shown by [54] that increasing the width of the network is more effective than increasing the depth to improve the performance, in the following exploration, we always choose the depth to be 50 layers. Furthermore, we evaluate the proposed dense shortcut on one famous variant of Resnet: ResNext. The results are available in Table 5. The results show that a similar trend has been observed as the original ResNet. Note that ResNext has a similar number of parameters as ResNet, while wide ResNet has almost three times more parameters and GFLOPS. Since the wide ResNets essentially adopts the same structure as ResNet and the small difference is only doubling the Conv 3×3 feature-maps, to avoid redundancy we only report one case in Table 4 and do not perform more experiments on this structure. Even though ResNeXt is a well-optimized structure, our proposed approach can further improve its performance with a significant margin. Interestingly, DS2NeXt (0.5×) can even outperform ResNeXt (1×) with a margin of 0.66%. This is quite surprising because ResNeXt (1×) has around four times more parameters and GFLOPS than DS2NeXt (0.5×). Note that the number of parameters and GFLOPS increase linearly with the increase of depth but quadratically with the increase of width. We further compare our performance on CIFAR-100 with the performance reported in previous works. WRN-28-10 does not adopt the bottleneck structure, thus DS2WRN-28-10 is not applicable (the shortcut within the Conv block can only be inserted into the Conv 3×3 in the bottleneck for the dimension to match). From

Table 5. Classification error (%) on CIFAR-100 validation dataset for different widths with ResNeXt as the backbone.

| Width | ResNeXt50 | DSNeXt50 | DS2NeXt50 |
|---|---|---|---|
| 0.25 | 23.86 | 22.98 | 22.58 |
| 0.5 | 21.16 | 19.95 | 19.51 |
| 1.0 | 20.17 | 18.58 | 18.24 |

Table 6. Classification error (%) on CIFAR-100 validation dataset.

| Architecture | params | Top-1 (%) |
|---|---|---|
| ALL-CNN [43] | - | 33.71 |
| Deeply supervised Net [29] | - | 34.57 |
| HighWay Network [44] | - | 32.39 |
| FractalNet [28] | 38.6M | 23.30 |
| with dropout [28] | 38.6.M | 23.73 |
| ResNet [17] | 1.7M | 27.22 |
| ResNet with stochastic depth [22] | 1.7M | 24.58 |
| Preacitvation ResNet [18] | 10.2M | 22.71 |
| DenseNet($k$ = 24) [21] | 27.2M | 19.25 |
| DenseNet-BC ($k$ = 24) [21] | 15.3M | 17.60 |
| DenseNet-BC ($k$ = 40) [21] | 25.6M | 17.18 |
| WRN-28-10 [54] | 36.5M | 19.25 |
| with dropout [54] | 36.5M | 18.85 |
| ResNeXt-29, 8×64$d$ [53] | 34.4M | 17.77 |
| ResNeXt-29, 16×64$d$ [53] | 68.1M | 17.31 |
| DSWRN-28-10 | 36.5M | 18.33 |
| DSNeXt-29, 6×64$d$ | 34.4M | 16.85 |
| DS2NeXt-29, 6×64$d$ | 34.4M | 16.39 |

Table 6 we observe that the networks that do not use dense connection perform much worse than those adopting dense connection. FractalNet adopting neither identity shortcut nor dense concatenation but with careful engineering design performs much better than other networks without dense connection; however, the performance is still not comparable to ResNet variants or DenseNets. Similar to FractalNet our proposed DSNet adopts neither identity shortcut nor dense concatenation, but it outperforms both ResNets (including WRN and ResNeXt) and DenseNets. The results show that dense weight normalized shortcuts constitute a competitive dense connection technique.

## 4.2. ImageNet experiments

ImageNet is the benchmark dataset for classification tasks to evaluate and compare different approaches. Our implementation details follow ResNet [17]. Specifically, we adopt the commonly used random $224 \times 224$ cropping with scale and aspect ratio augmentation for training and adopt SGD as the optimizer. For typical training on ImageNet, 8 GPUs are used and the batch size is set to 256 [17]. We used 4 GPUs to train the proposed DSNet and the batch size is set to 128 (32 per GPU). Accordingly, taking linear scaling rule into account, we set the initial learning rate to 0.05 instead of the commonly used 0.1. We train the network for 100 epochs and the learning rate is divided by 10 after every 30 epochs. We report the single-crop classification errors on the validation dataset with the input image size of $224 \times 224$. Both top-1 and top-5 errors are reported and the results are available in Table 7.

We note that the proposed DS2Net50 can achieve sig-

Table 7. Classification top-1 error (%) on ImageNet validation dataset.

| Architecture | Params | Top-1 (%) | Top-5 (%) |
|---|---|---|---|
| ResNet50 [17] | 25.6M | 24.01 | 7.02 |
| ResNet101 [17] | 44.6M | 22.44 | 6.21 |
| ResNet152 [17] | 60.2M | 22.16 | 6.16 |
| WRN-50-2-bottleneck [54] | 69.8M | 21.9 | 6.03 |
| ResNeXt-50, 32×4*d* [54] | 25M | 22.2 | - |
| SE-ResNet50 [19] | 28.1M | 23.29 | 6.62 |
| CBAM-ResNet50 [50] | 28.1M | 22.66 | 6.31 |
| *b*-RGSNet50 [59] | 25.6M | 22.68 | 6.42 |
| Res-RGSNet50 [59] | 25.6M | 22.21 | 5.99 |
| DenseNet201 [21] | 20M | 22.58 | 6.34 |
| DenseNet264 [21] | 33.3M | 22.15 | 6.12 |
| Res2Net [14] | 33.3M | 22.01 | 6.15 |
| DSNet50 | 25.6M | 22.49 | 6.29 |
| DS2Net50 | 25.6M | 22.03 | 5.93 |
| DS2Res2Net50 | 25.6M | 21.61 | 5.83 |

nificantly better performance than the original ResNet50. Somewhat surprisingly, DS2Net50 can outperform SENet[19] (which won the first place in ILSVRC 2017 challenge) as well as CBAM with improved attention module by a relatively large margin. DS2Net50 can achieve equivalent (if not better) performance with that of much deeper ResNet152. Compared with WRN-50-2-bottleneck, DS2Net achieves slightly worse result with the top-1 error metric and slightly better performance with the top-5 error metric. Note that WRN-50-2 has almost three times more parameters and GFLOPS than DS2Net. DS2Net-50 achieves slightly better performance than ResNeXt-50, 32×4*d* and Res-RGSNet50. It is claimed by [21] that DSNet201 can achieve equivalent performance with ResNet101 which has twice more parameters and computation GFLOPS. Even though DS2Net with a smaller number of parameters or GFLOPS outperforms DenseNet264, we do not aim to argue it as the main merit of DS2Net over DenseNet. Instead, the main advantage of DS2Net over DenseNet is that it is more time-efficient in practice with GPU implementation. In short, DSNet shows equivalent or better performance with DenseNet but it avoids the drawback of DenseNet. Our proposed DS2Net-50 also achieves comparable performance as the recent Rese2Net-50 [14]. By applying the dense shortcuts to Res2Net, we further improve the performance by 0.40% margin, which is not trivial considering Res2Net is already a very well designed architecture. Compared with ResNet50, DS2Net50 improves the performance of ResNet50 by a large margin. The number of the added weighted parameters is less than 0.15% of the original number of parameters and only a small computation overhead is added because it still adopts the same backbone structure as the original ResNet and only light operations are needed for the added shortcut.

The training curve is shown in Figure 4. We observe that the proposed dense weighted normalized shortcut is also beneficial to speed up the convergence. It is also important to note that the training error of DS2Net is much smaller.

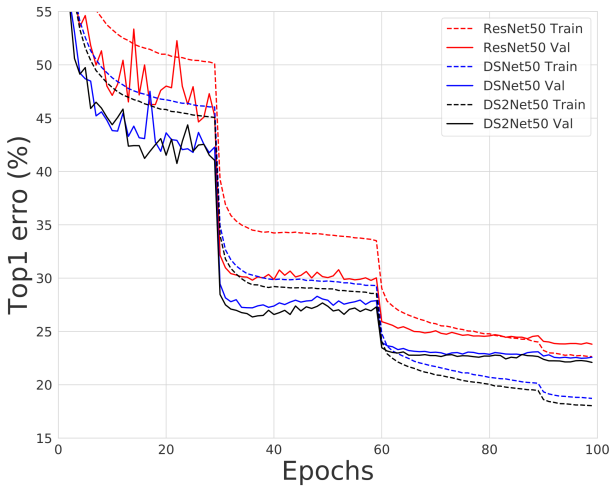


Figure 4. Training curves on ImageNet.

### 4.3. COCO detection dataset experiments

To evaluate the dataset generalization capability of the proposed DSNet, we further evaluate it on MS COCO 2014 detection dataset [30]. Faster-RCNN [38] is chosen as the detection method. The network is pre-trained on ImageNet and then finetuned on the COCO dataset with 5 epochs for fast performance validation. The results are available in Table 8. On COCO detection dataset, our proposed DSNet achieves better performance than ResNet50.

Table 8. mAP (%) on MS COCO validation dataset.

| Backbone | mAP.5 | mAP.75 | mAP [.5, .95] |
|---|---|---|---|
| ResNet50 | 51.3 | 33.6 | 31.4 |
| DSNet50 | 54.2 | 36.0 | 33.7 |
| DS2Net50 | 54.3 | 36.2 | 33.7 |

### 4.4. Visualization with Grad-CAM

We apply the widely used Grad-CAM [41] to ResNet50 [17] and our proposed DSNet/DS2Net, on the images from the ImageNet validation set. The visualization results are available in Figure 5. Grad-CAM calculates the gradients concerning a certain class, thus Grad-CAM result shows the attended regions in the image. We observe that the DSNet attends more on the objects and have a sharper focus (i.e. attention) than ResNet50. No obvious difference is observed between DSNet50 and DS2Net50.

### 4.5. Implementation design and memory/speed test

The straightforward implementation of dense normalized shortcut is to do normalization with affine transformation in every shortcut. This will cause unnecessary computation and slightly more parameter overhead burden. First, the normalization process is shared for a certain preceding layer feature. Second, the affine transformation by default has both scale and bias, while the summation of dense bias

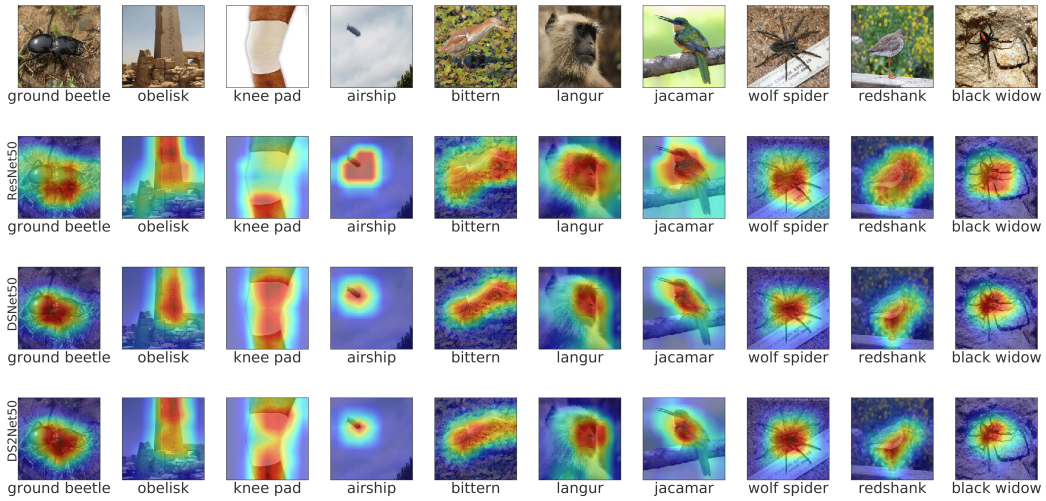


Figure 5. Grad-CAM [41] visualization results of ResNet50 (second row), DSNet50 (third row) and DS2Net50 (llast row).

is mathematically redundant. In our implementation, for every aggregation output $Y_l$, we only perform the normalization process once which is then shared by all the dense shortcuts linked to it. Thus, for the dense shortcut path, the operation is only to multiply the shared normalized feature by the corresponding weight, which is very light. This implementation choice does not influence the performance but requires less computation time.

Table 9. GPU memory and training time on ImageNet; memory indicates that per GPU and time indicates that per iteration.

| Architecture | Top-1 (%) | memory (MB) | time (s) |
|---|---|---|---|
| ResNet50 [17] | 24.01 | 3929 | 0.31 |
| ResNet152 [17] | 22.16 | 7095 | 0.63 |
| DenseNet264 [21] | 22.15 | 9981 | 0.60 |
| DSNet50 | 22.49 | 4777 | 0.37 |
| DS2Net50 | 22.03 | 5133 | 0.39 |

With the above implementation, we measure the ImageNet training memory and speed on the same machine (equipped with four 1080Ti GPUs) with the hyperparameters specified above. We report consumed memory per GPU and computation time per iteration in Table 9. DS2Net (with a smaller number of parameters as shown in Table 7) performs relatively better than ResNet152 and Dense264 with the advantage for both memory and speed. Compared with ResNet50, the increase of memory and computation for DSNet/DS2Net is marginal.

For the speed evaluation, we mainly report the training time. Note, however, that the inference time is proportional to training time and thus our DS2Net50 also has the advantage of being fast during the inference stage compared with ResNet152 and DenseNet264.

## 5. Conclusions

We provide a unified perspective of dense summation to facilitate the understanding of the core difference between ResNet and DenseNet. We demonstrate that the core difference lies in whether the convolution parameters are shared for the preceding feature-maps. We proposed a dense weighted normalized shortcut as an alternative dense connection method, which outperforms the two existing dense connection techniques: identity shortcut in ResNet and dense concatenation in DenseNet. We found that Dense summation from the aggregation output provides superior performance to that from the convolutional block output. In short, the dense shortcut mitigates the problem of representational capacity decrease in ResNet while avoiding the drawback of requiring more GPU resources in DenseNet. The proposed DSNet has been evaluated on multiple benchmark datasets to show superior performance than its counterpart ResNet. For example, on ImageNet, DenseNet50 can achieve better performance than the much deeper ResNet152. On ImageNet with few parameters and computation, it also achieves comparable performance as DenseNet. Moreover, it also achieves comparable performance as the recent Res2Net which can be further boosted by our dense shortcuts. Compared with other "free" performance boost module such as SE and CBAM, our dense shortcut also achieves more superior performance. The Grad-CAM result shows that DSNet, in general, focuses better on the object in the image than its counterpart ResNet.

## Acknowledgements

This work was funded by NaverLabs.

## References


[1] Muhammad Awais, Md Tauhid Bin Iqbal, and Sung-Ho Bae. Revisiting internal covariate shift for batch normalization. *Transactions on Neural Networks and Learning Systems*, 2020.

[2] Muhammad Awais, Fahad Shamshad, and Sung-Ho Bae. Towards an adversarially robust normalization approach. *arXiv preprint arXiv:2006.11007*, 2020.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Jiawang Bai, Bin Chen, Yiming Li, Dongxian Wu, Weiwei Guo, Shu-tao Xia, and En-hui Yang. Targeted attack for deep hashing based retrieval. In *ECCV*, 2020.

[5] Philipp Benz, Chaoning Zhang, Tooba Imtiaz, and In-So Kweon. Data from model: Extracting data from non-robust and robust models. *arXiv preprint arXiv:2007.06196*, 2020.

[6] Philipp Benz, Chaoning Zhang, Tooba Imtiaz, and In So Kweon. Double targeted universal adversarial perturbations. In *ACCV*, 2020.

[7] Philipp Benz, Chaoning Zhang, Adil Karjauv, and In So Kweon. Robustness may be at odds with fairness: An empirical study on class-wise accuracy. *arXiv preprint arXiv:2010.13365*, 2020.

[8] Philipp Benz, Chaoning Zhang, Adil Karjauv, and In So Kweon. Revisiting batch normalization for improving corruption robustness. *WACV*, 2021.

[9] Philipp Benz, Chaoning Zhang, and In So Kweon. Batch normalization increases adversarial vulnerability: Disentangling usefulness and robustness of model features. *arXiv preprint arXiv:2010.03316*, 2020.

[10] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. In *NeurIPS*, 2017.

[11] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[13] Yan Feng, Bin Chen, Tao Dai, and Shutao Xia. Adversarial attack on deep product quantization network for image retrieval. In *AAAI*, 2020.

[14] Shanghua Gao, Ming-Ming Cheng, Kai Zhao, Xin-Yu Zhang, Ming-Hsuan Yang, and Philip HS Torr. Res2net: A new multi-scale backbone architecture. *TPAMI*, 2019.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[16] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[19] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018.

[20] Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *CVPR*, 2018.

[21] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[22] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[24] Dahun Kim, Sanghyun Woo, Joon-Young Lee, and In So Kweon. Recurrent temporal aggregation framework for deep video inpainting. *TPAMI*, 2019.

[25] Dahun Kim, Sanghyun Woo, Joon-Young Lee, and In So Kweon. Video panoptic segmentation. In *CVPR*, 2020.

[26] Myungchul Kim, Sanghyun Woo, Dahun Kim, and In So Kweon. The devil is in the boundary: Exploiting boundary representation for basis-based instance segmentation. *WACV*, 2021.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[28] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[29] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial intelligence and statistics*, 2015.

[30] 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 *ECCV*, 2014.

[31] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[32] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[33] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015.

[34] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017.

[35] Fei Pan, Inkyu Shin, Francois Rameau, Seokju Lee, and In So Kweon. Unsupervised intra-domain adaptation for semantic segmentation through self-supervision. In *CVPR*, 2020.

[36] George Philipp, Dawn Song, and Jaime G. Carbonell. Gradients explode-deep networks are shallow-resnet explained. In *ICLR Workshop*, 2018.

[37] Geoff Pleiss, Danlu Chen, Gao Huang, Tongcheng Li, Laurens van der Maaten, and Kilian Q Weinberger. Memory-efficient implementation of densenets. *arXiv preprint arXiv:1707.06990*, 2017.

[38] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015.

[39] Tim Salimans and Durk P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NeurIPS*, 2016.

[40] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *NeurIPS*, 2018.

[41] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, 2017.

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[43] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[44] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[46] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[47] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[48] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[49] Wenhai Wang, Xiang Li, Jian Yang, and Tong Lu. Mixed link networks. *arXiv preprint arXiv:1802.01808*, 2018.

[50] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. CBAM: Convolutional block attention module. In *ECCV*, 2018.

[51] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

[52] Cihang Xie and Alan Yuille. Intriguing properties of adversarial training at scale. *ICLR*, 2020.

[53] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

[54] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[55] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

[56] Chaoning Zhang, Philipp Benz, Tooba Imtiaz, and In-So Kweon. Cd-uap: Class discriminative universal adversarial perturbation. In *AAAI*, 2020.

[57] Chaoning Zhang, Philipp Benz, Tooba Imtiaz, and In-So Kweon. Understanding adversarial examples from the mutual influence of images and perturbations. In *CVPR*, 2020.

[58] Chaoning Zhang, Francois Rameau, Junsik Kim, Dawit Mureja Argaw, Jean-Charles Bazin, and In So Kweon. Deepptz: Deep self-calibration for ptz cameras. In *WACV*, 2020.

[59] Chaoning Zhang, Francois Rameau, Seokju Lee, Junsik Kim, Philipp Benz, Dawit Mureja Argaw, Jean-Charles Bazin, and In So Kweon. Revisiting residual networks with nonlinear shortcuts. In *BMVC*, 2019.

[60] Ting Zhang, Guo-Jun Qi, Bin Xiao, and Jingdong Wang. Interleaved group convolutions. In *ICCV*, 2017.