Deep Convolutional Neural Networks with Merge-and-Run Mappings

Liming Zhao[1] Jingdong Wang[2] Xi Li[1] Zhuowen Tu[3] Wenjun Zeng[2]
[1]Zhejiang University [2]Microsoft Research [3]UC San Diego
{zhaoliming,xilizju}@zju.edu.cn {jingdw,wezeng}@microsoft.com ztu@ucsd.edu

Abstract

A deep residual network, built by stacking a sequence of residual blocks, is easy to train, because identity mappings skip residual branches and thus improve information flow. To further reduce the training difficulty, we present a simple network architecture, deep merge-and-run neural networks. *The novelty lies in a modularized building block,* merge-and-run block, *which assembles residual branches in parallel through a* merge-and-run mapping*: Average the inputs of these residual branches (*Merge*), and add the average to the output of each residual branch as the input of the subsequent residual branch (*Run*), respectively. We show that the merge-and-run mapping is a linear idempotent function in which the transformation matrix is idempotent, and thus improves information flow, making training easy. In comparison to residual networks, our networks enjoy compelling advantages: they contain much shorter paths, and the width, i.e., the number of channels, is increased. We evaluate the performance on the standard recognition tasks. Our approach demonstrates consistent improvements over ResNets with the comparable setup, and achieves competitive results (e.g.,* 3.57% *testing error on CIFAR-*10, 19.00% *on CIFAR-*100, 1.51% *on SVHN).*

1. Introduction

Deep convolutional neural networks, since the breakthrough result in the ImageNet classification challenge [13], have been widely studied [30, 27, 7]. Surprising performances have been achieved in many other computer vision tasks, including object detection [5], semantic segmentation [17], edge detection [38], and so on.

Residual networks (ResNets) [7] have been attracting a lot of attentions since it won the ImageNet challenge and various extensions have been studied [39, 32, 40, 1]. The basic unit is a residual block consisting of a residual branch and an identity mapping. Identify mappings introduce short paths from the input to the intermediate layers and from the intermediate layers to the output layers [35, 34], and thus reduce the training difficulty.

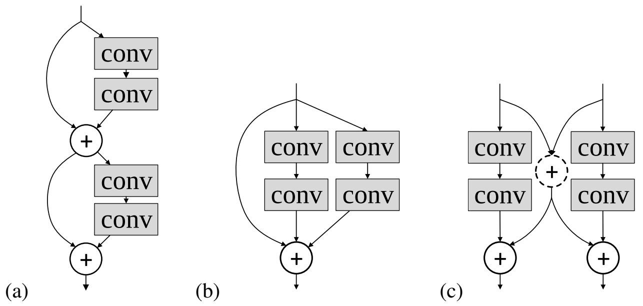


Figure 1. Illustrating the building blocks: (a) Two residual blocks; (b) An inception-like block; (c) A merge-and-run block. (a) corresponds to two blocks in ResNets and assembles two residual branches sequentially while (b) and (c) both assemble the same two residual branches in parallel. (b) and (c) adopt two different skip connections: identity mappings and our proposed merge-and-run mappings. The dot circle denotes the average operation, and the solid circle denotes the sum operation.

In this paper, we are interested in further reducing the training difficulty and present a simple network architecture, called deep merge and run neural networks, which assemble residual branches more effectively. The key point is a novel building block, the *merge-and-run* block, which assembles residual branches in parallel with a *merge-and-run mapping*: Average the inputs of these residual branches (*Merge*), and add the average to the output of each residual branch as the input of the subsequent residual branch (*Run*), respectively. Figure 1 depicts the architectures by taking two residual branches as an example: (a) two residual blocks that assemble two residual branches sequentially, (b) an inception-like block and (c) a merge and run block both assemble the same two residual branches in parallel.

Obviously, the resulting network contains shorter paths as the parallel assembly of residual branches directly reduces the network depth. We give a straightforward verification: The average length of two residual blocks is 2, while the average lengths of the corresponding inception-like block and merge-and-run block are $\frac{4}{3}$ and $\frac{2}{3}$, respectively. Our networks, built by stacking merge-and-run blocks, are less deep and thus easier to train.

We show that the merge-and-run mapping is a linear idempotent function, where the transformation matrix is idempotent. This implies that the information from the early blocks can quickly flow to the later blocks, and the gradient can be quickly back-propagated to the early blocks from the later blocks. This point essentially provides a theoretic counterpart of short paths, showing the training difficulty is reduced.

We further show that merge-and-run blocks are wider than residual blocks. Empirical results validate that for very deep networks, as a way to increase the number of layers, increasing the width is more effective than increasing the depth. Besides, we discuss the generalizability of merge-and-run mappings to other linear idempotent transformations, and the extension to more residual branches.

The empirical results demonstrate that the performances of our networks are superior to the corresponding ResNets with comparable setup on CIFAR-10, CIFAR-100, SVHN and ImageNet. Our networks achieve competitive results compared to state-of-the-arts (e.g., 3.57% testing error on CIFAR-10, 19.00% on CIFAR-100, 1.51% on SVHN).

2. Related Works

There have been rapid and great progress of deep neural networks in various aspects, such as optimization techniques [28, 11, 18], initialization schemes [19], regularization strategies [26], activation and pooling functions [6, 3], network architecture [16, 22, 20, 24], and applications. In particular, recently network architecture design has been attracting a lot of attention.

Highway networks [27], residual networks [7, 8], and GoogLeNet [30] are shown to be able to effectively train a very deep (over 40 and even hundreds or thousands) network. The identity mapping or the bypass path are thought as the key factor to make the training of very deep networks easy. Following the ResNet architecture, several variants are developed by modifying the architectures, such as wide residual networks [39], ResNet in ResNet [32], multilevel residual networks [40], multi-residual networks [1], and so on. Another variant, DenseNets [9], links all layers with identity mappings and are able to improve the effectiveness through feature reuse. In addition, optimization techniques, such as stochastic depth [10] for ResNet optimization, are developed.

Deeply-fused networks [35], FractalNet [14], and ensemble view [34] point out that a ResNet and a GoogleNet [30, 31, 29] are a mixture of many dependent networks. Ensemble view [34] observes that ResNets behave like an exponential ensemble of relatively shallow networks, and points out that introducing short paths helps ResNets to avoid the vanishing gradient problem, which is similar to the analysis in deeply-fused networks [35] and FractalNet [14].

The architecture of our approach is closely related to Inception [30] and Inception-ResNet blocks [29], multi-residual networks [1], and ResNeXt [37], which also contain multiple branches in each block. One notable point is that we introduce merge-and-run mappings, which are linear idempotent functions, to improve information flow for building blocks consisting of parallel residual branches.

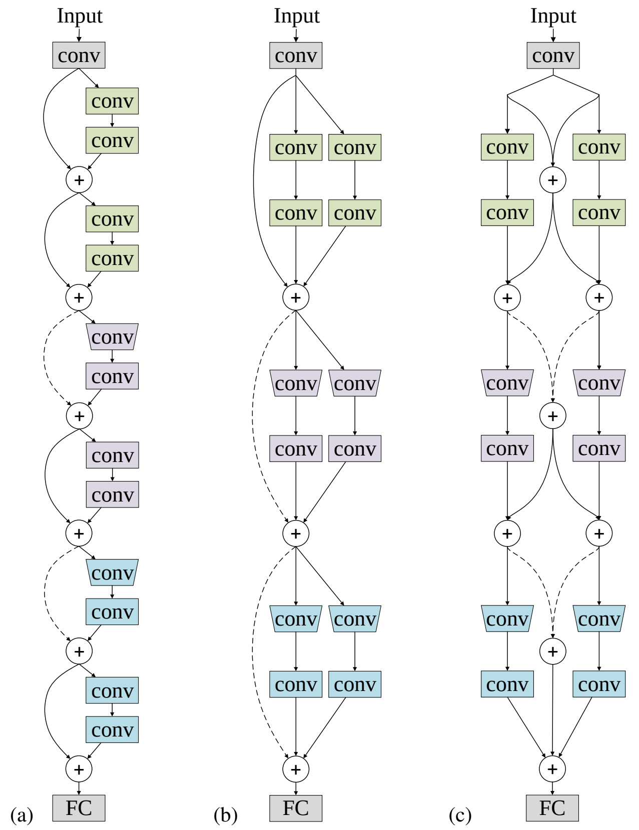


Figure 2. (a) a deep residual network; (b) a network built by stacking inception-like blocks; (c) our deep merge-and-run neural network built by stacking merge-and-run blocks. The trapezoid shape indicates that down-sampling occurs in the corresponding layer, and the dashed line denotes a projection shortcut as in [7].

In comparison to ResNeXts [37] that also assemble residual branches in parallel, our approach adopts parallel assembly to directly reduce the depth and does not modify residual branches, while ResNeXts [37] transform a residual branch to many small residual branches. Compared to Inception [30] and Inception-ResNet blocks [29] that are highly customized, our approach requires less efforts to design and more flexible.

3. Deep Merge-and-Run Neural Networks

3.1. Architectures

We introduce the architectures by considering a simple realisation, assembling two residual branches in parallel to form the building blocks. We first introduce the building blocks in ResNets, then a straightforward manner to assemble residual branches in parallel, and finally our building

Table 1. Network architectures. Inside the brackets are the shape of the residual, inception-like and merge-and-run blocks, and outside the brackets is the number of stacked blocks on a stage. Downsampling is performed in conv2_1, and conv3_1 with stride 2. As done [7], we use 1×1 convolutions to replace identity mappings for ResNets and Inception-like blocks, and perform 1×1 convolutions before merging in merge-and-run blocks when the widths increase across stages. In each convolution, the input channel number can be known from the preceding layer.

Layers	Output size	ResNets	DMRNets/DILNets
conv0	32×32	$\text{conv}(3 \times 3, 16)$	
con1_x	32×32	$\begin{bmatrix}\text{conv}(3 \times 3, 16)\\ \text{conv}(3 \times 3, 16)\end{bmatrix} \times \frac{2L}{3}$	$\begin{bmatrix}\text{conv}(3 \times 3, 16) & \text{conv}(3 \times 3, 16)\\ \text{conv}(3 \times 3, 16) & \text{conv}(3 \times 3, 16)\end{bmatrix} \times \frac{L}{3}$
conv2_x	16×16	$\begin{bmatrix}\text{conv}(3 \times 3, 32)\\ \text{conv}(3 \times 3, 32)\end{bmatrix} \times \frac{2L}{3}$	$\begin{bmatrix}\text{conv}(3 \times 3, 32) & \text{conv}(3 \times 3, 32)\\ \text{conv}(3 \times 3, 32) & \text{conv}(3 \times 3, 32)\end{bmatrix} \times \frac{L}{3}$
conv3_x	8×8	$\begin{bmatrix}\text{conv}(3 \times 3, 64)\\ \text{conv}(3 \times 3, 64)\end{bmatrix} \times \frac{2L}{3}$	$\begin{bmatrix}\text{conv}(3 \times 3, 64) & \text{conv}(3 \times 3, 64)\\ \text{conv}(3 \times 3, 64) & \text{conv}(3 \times 3, 64)\end{bmatrix} \times \frac{L}{3}$
Classifier	1×1	average pool, FC, softmax	

blocks.

The three building blocks are illustrated in Figure 1. Examples of the corresponding network structures, ResNets, DILNets (deep inception-like neural networks), and DMRNets (deep merge-and-run neural networks), are illustrated in Figure 2. The descriptions of network structures used in this paper are given in Table 1.

Residual blocks. A residual network is composed of a sequence of residual blocks. Each residual block contains two branches: identity mapping and residual branch. The corresponding function is given as,

$$\mathbf{x}_{t+1} = H_t(\mathbf{x}_t) + \mathbf{x}_t. \tag{1}$$

Here, $\mathbf{x}_t$ denotes the input of the tth residual block. $H_t(\cdot)$ is a transition function, corresponding to the residual branch composed of a few stacked layers.

Inception-like blocks. We assemble two residual branches in parallel and sum up the outputs from the two residual branches and the identity mapping. The functions, corresponding to the $(2t)$th and $(2t+1)$th residual branches, are as follows,

$$\mathbf{x}_{2(t+1)} = H_{2t}(\mathbf{x}_{2t}) + H_{2t+1}(\mathbf{x}_{2t}) + \mathbf{x}_{2t}, \tag{2}$$

where $\mathbf{x}_{2t}$ and $\mathbf{x}_{2(t+1)}$ are the input and the output of the tth inception-like block. This structure resembles the building block in the concurrently-developed ResNeXt [37], but the purposes are different: Our purpose is to reduce the depth through assembling residual branches in parallel while the purpose of ResNeXt [37] is to transform a single residual branches to many small residual branches.

Merge-and-run. A merge-and-run block is formed by assembling two residual branches in parallel with a *merge-and-run* mapping: Average the inputs of two residual branches (*Merge*), and add the average to the output of each residual branch as the input of the subsequent residual branch (*Run*), respectively. It is formulated as below,

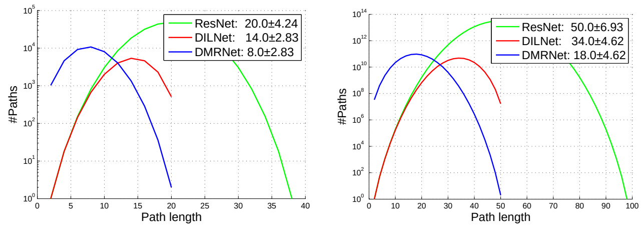


Figure 3. Comparing the distributions of the path lengths for three networks. Different networks: (avg length ± std). Left: $L = 9$. Right: $L = 24$.

$$\begin{aligned}\mathbf{x}_{2(t+1)} &= H_{2t}(\mathbf{x}_{2t}) + \frac{1}{2}(\mathbf{x}_{2t} + \mathbf{x}_{2t+1}),\\ \mathbf{x}_{2(t+1)+1} &= H_{2t+1}(\mathbf{x}_{2t+1}) + \frac{1}{2}(\mathbf{x}_{2t} + \mathbf{x}_{2t+1}),\end{aligned} \tag{3}$$

where $\mathbf{x}_{2t}$ and $\mathbf{x}_{2t+1}$ ($\mathbf{x}_{2(t+1)}$ and $\mathbf{x}_{2(t+1)+1}$) are the inputs (outputs) of two residual branches of the tth block. There is a clear difference from inception-like blocks in Equation 2: the inputs of two residual branches are different, and their outputs are also separated.

3.2. Analysis

Information flow improvement. We transform Equation 3 into the matrix form,

$$\begin{bmatrix}\mathbf{x}_{2(t+1)}\\ \mathbf{x}_{2(t+1)+1}\end{bmatrix} = \begin{bmatrix}H_{2t}(\mathbf{x}_{2t})\\ H_{2t+1}(\mathbf{x}_{2t+1})\end{bmatrix} + \frac{1}{2}\begin{bmatrix}\mathbf{I} & \mathbf{I}\\ \mathbf{I} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{x}_{2t}\\ \mathbf{x}_{2t+1}\end{bmatrix}, \tag{4}$$

where $\mathbf{I}$ is an $d \times d$ identity matrix and d is the dimension of $\mathbf{x}_{2t}$ (and $\mathbf{x}_{2t+1}$). $\mathbf{M} = \frac{1}{2}\begin{bmatrix}\mathbf{I} & \mathbf{I}\\ \mathbf{I} & \mathbf{I}\end{bmatrix}$ is the transformation matrix of the merge-and-run mapping.

It is easy to show that like the identity matrix $\mathbf{I}$, $\mathbf{M}$ is an *idempotent* matrix, i.e., $\mathbf{M}^n = \mathbf{M}$, where n is an arbitrary positive integer. Thus, we have[1]

[1]Similar to identity mappings, the analysis is applicable to the case

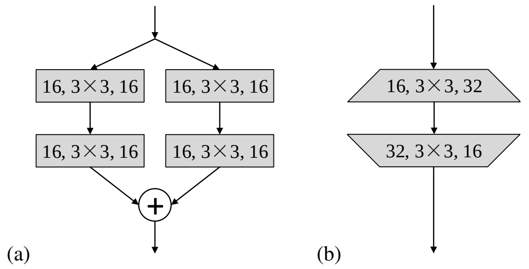


Figure 4. Illustrating the two residual branches shown in (a) are transformed to a single residual branch shown in (b). (a) All 4 convolutions are $(16, 3 \times 3, 16)$. (b) The 2 convolutions are $(16, 3 \times 3, 32)$ and $(32, 3 \times 3, 16)$, from narrow (16) to wide (32), and then from wide (32) back to narrow (16).

$$\begin{bmatrix}\mathbf{x}_{2(t+1)} \\ \mathbf{x}_{2(t+1)+1}\end{bmatrix} = \begin{bmatrix}H_{2t}(\mathbf{x}_{2t}) \\ H_{2t+1}(\mathbf{x}_{2t+1})\end{bmatrix} + \mathbf{M}\sum_{i=t'}^{t-1}\begin{bmatrix}H_{2i}(\mathbf{x}_{2i}) \\ H_{2i+1}(\mathbf{x}_{2i+1})\end{bmatrix} + \mathbf{M}\begin{bmatrix}\mathbf{x}_{2t'} \\ \mathbf{x}_{2t'+1}\end{bmatrix}, \tag{5}$$

where $t' < t$ corresponds to an earlier block[2]. This implies that during the forward propagation there are quick paths directly sending the input and the outputs of the intermediate residual branches to the later block. We have a similar conclusion for gradient back-propagation. Consequently, merge-and-run mappings improve both forward and backward information flow.

Shorter paths. All the three networks are mixtures of paths, where a path is defined as a sequence of connected residual branches, identity mappings, and possibly other layers (e.g., the first convolution layer, the FC layer) from the input to the output. Suppose each residual branch contains B layers (there are 2 layers for the example shown in Figure 1), and the ResNet, DIRNet and DMRNet contain $2L$, L, and L building blocks, the average lengths (without counting projections in short-cut connections) are $BL + 2$, $\frac{2B}{3}L + 2$, and $\frac{B}{3}L + 2$, respectively. Figure 3 shows the distributions of path lengths of the three networks. Refer to Table 1 for the details of the network structures.

It is shown in [7, 27] that for very deep networks the training becomes hard and that a shorter (but still very deep) plain network performs even better than a longer plain network[3]. According to Figure 3 showing that the lengths of the paths in our proposed network are distributed in the range of lower lengths, the proposed deep merge-and-run

that the flow is not stopped by nonlinear activation ReLU. This equation is similar to the derivation with identity mappings in [8].

[2]The second term in the right-hand side of Equation 5 does not exist if $(t+1)$ corresponds to the block right after the input of the whole network.

[3]Our empirical results even show that the deepest path in ResNets hurts the training of other paths and thus deteriorates the performance.

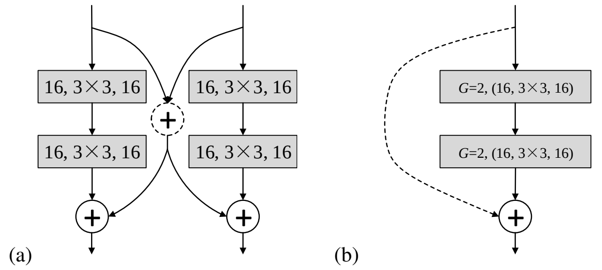


Figure 5. Transform the merge-and-run block shown in (a) to a two-branch block shown in (b). (b) The 2 convolutions are group convolutions. A group convolution contains two $(G = 2)$ convolutions of $(16, 3 \times 3, 16)$: each receives a different 16-channel input and the two outputs are concatenated as the final output with 32 channels. The width is greater than 16. The skip connection (dot line) is a linear transformation, where the transformation matrix of size 32×32 is idempotent.

network potentially performs better.

Inception-like blocks are wider. We rewrite Equation 2 in a matrix form,

$$\mathbf{x}_{2(t+1)} = \begin{bmatrix}\mathbf{I} & \mathbf{I}\end{bmatrix}\begin{bmatrix}H_{2t}(\mathbf{x}_{2t}) \\ H_{2t+1}(\mathbf{x}_{2t})\end{bmatrix} + \mathbf{x}_{2t}. \tag{6}$$

Considering the two parallel residual branches, i.e., the first term of the right-hand side, we have several observations. (1) The intermediate representation, $\begin{bmatrix}H_{2t}(\mathbf{x}_{2t}) \\ H_{2t+1}(\mathbf{x}_{2t})\end{bmatrix}$ is $(2d)$-dimensional and wider. (2) The output becomes narrower after multiplication by $\begin{bmatrix}\mathbf{I} & \mathbf{I}\end{bmatrix}$, and the width is back to d. (3) The block is indeed wider except some trivial cases, e.g., each residual branch does not contain nonlinear activations.

Figure 4 presents an example to illustrate that inception-like block is wider. There are two layers in each branch. We have that the two residual branches is equivalent to a single residual branch, also containing two layers: the first layer increases the width from d ($d = 16$ in Figure 4) to $2d$, and the second layer reduces the width back to d. There is no such simple transformation for residual branches with more than two layers, but we have similar observations.

Merge-and-run blocks are much wider. Consider Equation 4, we can see that the widths of the input, the intermediate representation, and the output are all $2d$[4]. The block is wider than an inception-like block because the outputs of two residual branches in the merge-and-run block are separated and the outputs for the inception-like block are aggregated. The two residual branches are not independent as the merge-and-run mapping adds the input of one residual branch to the output of the other residual branch.

Figure 5 shows that the merge-and-run block is transformed to a two-branch block. The dot line corresponds

[4]In essence, the $2d$ space is not fully exploited because the convolutional kernel is block-diagonal.

to the merge-and-run mapping, and now becomes an integrated linear transformation receiving a single $(2d)$-dimensional vector as the input. The residual branch consists of two group convolutions, each with two partitions. A group convolution is equivalent to a single convolution with the larger convolution kernel, being a block-diagonal matrix with each block corresponding to the kernel of each partition in the group convolution.

4. Experiments

We empirically show the superiority of DILNets and DMRNets, compared with ResNets. We demonstrate the effectiveness of our DMRNets on several benchmark datasets and compare it with the state-of-the-arts.

4.1. Datasets

CIFAR-10 **and CIFAR-**100**.** The two datasets are both subsets [12] drawn from the 80-million tiny image database [33]. CIFAR-10 consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. CIFAR-100 is like CIFAR-10, except that it has 100 classes each containing 600 images. We follow a standard data augmentation scheme widely used for this dataset [15, 7, 9]: we first zero-pad the images with 4 pixels on each side, and then randomly crop to produce 32×32 images, followed by horizontally mirroring half of the images. We preprocess the images by normalizing the images using the channel means and standard deviations.

SVHN. The SVHN (street view house numbers) dataset consists of digit images of size 32×32. There are $73,257$ images as the training set, $531,131$ images as a additional training set, and $26,032$ images as the testing set. Following the common practice [16, 15, 10], we select out 400 samples per class from the training set and 200 samples per class from the additional set, and use the remaining $598,388$ images for training.

4.2. Setup

Networks. We follow ResNets to design our layers: use three stages (conv1_x, conv2_x, conv3_x) of merge-and-run blocks with the number of filter channels being $16, 32, 64$, respectively, and use a Conv-BN-ReLU as a basic layer with kernel size 3×3. The image is fed into the first convolutional layer (conv0) with 16 output channels, which then go to the subsequent merge-and-run blocks. In the experiments, we implement our approach by taking two parallel residual branches as an example. For convolutional layers with kernel size 3×3, each side of the inputs is zero-padded by one pixel. At the end of the last merge-and-run block, a global average pooling is performed and then a soft-max classifier is attached. All the $+$ operations in solid circles in Figures 1 are between BN and ReLU.

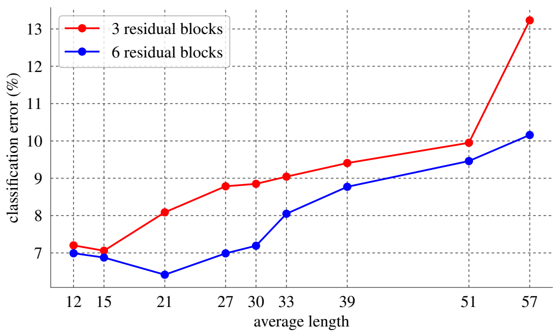


Figure 6. Illustrating how the testing errors of residual networks change as the average path length increases. The results are reported on CIFAR-10.

Training. We use the SGD algorithm with the Nesterov momentum to train all the models for 400 epochs on CIFAR-10/CIFAR-100 and 40 epochs on SVHN, both with a total mini-batch size 64 on two GPUs. The learning rate starts with 0.1 and is reduced by a factor 10 at the $1/2$, $3/4$ and $7/8$ fractions of the number of training epochs. Similar to [7, 9], the weight decay is 0.0001, the momentum is 0.9, and the weights are initialized as in [6]. Our implementation is based on MXNet [2].

4.3. Empirical Study

Shorter paths. We study how the performance changes as the average length of the paths changes, based on two kinds of residual networks. They are formed from the same plain network of depth $2L + 2$, whose structure is like the one forming ResNets given in Table 1: (i) Each residual branch is of length $\frac{2}{3}L$ and corresponds to one stage. There are totally 3 residual blocks. (ii) Each residual branch is of length $\frac{1}{3}L$. There are totally 6 residual blocks (like Figure 2 (a)). The averages of the depths of the paths are both $(L+2+1)$, with counting two projection layers in the shortcut connections.

We vary L and record the classification errors for each kind of residual network. Figure 6 shows the curves in terms of the average depth of all the paths vs. classification error over the example dataset CIFAR-10. We have the following observations. When the network is not very deep and the average length is small ($\leqslant$ 15 for 3 blocks, $\leqslant$ 21 for 6 blocks[5]), the testing error becomes smaller as the average length increases, and when the length is large, the testing error becomes larger as the length increases. This indicates that *shorter paths* result in the higher accuracy for very deep networks.

Comparison with ResNets. We compare DILNets and DMRNets, and the baseline ResNets algorithm. They are

[5]There are more short paths for 6 blocks, which leads to lower testing error than 3 blocks.

Table 2. Empirical comparison of DILNets, DMRNets, and ResNets. The average classification error from 5 runs and the standard deviation (mean $\pm$ std.) are reported. Refer to Table 1 for network structure descriptions.

		CIFAR-10			CIFAR-100			SVHN		
Params.	L	ResNets	DILNets	DMRNets	ResNets	DILNets	DMRNets	ResNets	DILNets	DMRNets
0.4M	12	6.62 ± 0.24	6.53 ± 0.12	$\mathbf{6.48 \pm 0.04}$	29.69 ± 0.15	29.75 ± 0.27	$\mathbf{29.62 \pm 0.08}$	$\mathbf{1.90 \pm 0.08}$	2.13 ± 0.09	2.00 ± 0.04
0.6M	18	5.93 ± 0.17	5.83 ± 0.09	$\mathbf{5.79 \pm 0.13}$	27.90 ± 0.26	27.87 ± 0.22	$\mathbf{27.80 \pm 0.26}$	1.97 ± 0.09	1.96 ± 0.10	$\mathbf{1.87 \pm 0.09}$
0.8M	24	5.60 ± 0.14	5.59 ± 0.17	$\mathbf{5.47 \pm 0.14}$	27.03 ± 0.66	26.88 ± 0.22	$\mathbf{26.76 \pm 0.16}$	1.93 ± 0.17	1.86 ± 0.14	$\mathbf{1.86 \pm 0.05}$
1.0M	30	5.50 ± 0.09	5.45 ± 0.09	$\mathbf{5.10 \pm 0.08}$	26.44 ± 0.69	26.19 ± 0.41	$\mathbf{25.87 \pm 0.04}$	1.89 ± 0.03	1.86 ± 0.07	$\mathbf{1.81 \pm 0.04}$
1.2M	36	5.35 ± 0.14	5.26 ± 0.20	$\mathbf{5.18 \pm 0.20}$	26.00 ± 0.48	25.98 ± 0.23	$\mathbf{25.41 \pm 0.19}$	1.90 ± 0.04	1.81 ± 0.11	$\mathbf{1.77 \pm 0.11}$
1.5M	48	5.26 ± 0.09	5.05 ± 0.20	$\mathbf{4.99 \pm 0.13}$	25.44 ± 0.20	24.76 ± 0.33	$\mathbf{24.73 \pm 0.40}$	1.91 ± 0.03	1.84 ± 0.06	$\mathbf{1.84 \pm 0.15}$
1.7M	54	5.24 ± 0.23	5.02 ± 0.17	$\mathbf{4.96 \pm 0.06}$	24.56 ± 0.15	24.56 ± 0.69	$\mathbf{24.41 \pm 0.09}$	2.00 ± 0.12	1.85 ± 0.09	$\mathbf{1.68 \pm 0.02}$
3.1M	96	5.47 ± 0.46	4.97 ± 0.03	$\mathbf{4.84 \pm 0.11}$	24.41 ± 0.10	24.06 ± 0.68	$\mathbf{23.98 \pm 0.15}$	1.85 ± 0.03	1.75 ± 0.06	$\mathbf{1.70 \pm 0.03}$

formed with the same number of layers, and each block in a DILNet and a DMRNet corresponds to two residual blocks in a ResNet. Table 1 depicts the network structures.

The comparison on CIFAR-10 is given in Table 2. One can see that compared with ResNets, DILNets and DMRNets consistently perform better, and DMRNets perform the best. The superiority of DILNets over ResNets stems from the less long paths and greater width. The additional advantages of a DMRNet are much greater width than a DILNet.

The comparisons over CIFAR-100 and SVHN shown in Table 2 are consistent. One exception is that on CIFAR-100 the ResNet of depth 26 ($L = 12$) performs better than DILNet and on SVHN the ResNet of depth 26 performs better than the DILNet and DMRNet. The reason might be that the paths in the DILNet and DMRNet are not very long and that too many short paths lower down the performance and that for networks of such a depth, the benefit from increasing the width is less than the benefit from increasing the depth.

Convergence curves. Figure 7 shows the convergence curves of ResNets and DMRNets over CIFAR-10, CIFAR-100, and SVHN. We show training losses instead of training errors because the training errors in CIFAR-10 almost reach zero at the convergence and are not distinguishable. One can see that the testing errors of DMRNets are smaller than ResNets and that the training losses are also smaller during the optimization process, suggesting that our gains are not from regularization but from richer representation.

4.4. Comparison with State-of-the-Arts

The comparison is reported in Table 3. We report the results of DMRNets since it is superior to DILNets. Refer to Table 1 for network architecture descriptions. We also report the results from the wide DMRNets (denoted by DMRNet-Wide), $4\times$ wider, i.e., the widths of the threes stages are 64, 128, and 256, respectively. We mark the results that outperform existing state-of-the-arts in bold and the best results in blue.

One can see that the DMRNet-Wide of depth 50 outperforms existing state-of-the-art results and achieves the best results on CIFAR-10 and CIFAR-100. Compared with the second best approach DenseNet[6] that includes more parameters (27.2M), our network includes only 24.8M parameters. DMRNet-Wide (depth $=$ 32) is very competitive: outperform all existing state-of-the-art results on SVHN. It contains only 14.9M parameters, almost half of the parameters (27.2M) of the competitive DenseNet. These results show that our networks are parameter-efficient.

Compared with the FractalNet with depth 21, DMRNets-Wide with depths 32, 50 are much deeper and contain fewer parameters (14.9M, 24.8M vs. 38.6M). Our networks achieve superior performances over all the three datasets. This also shows that because *merge-and-run mappings* improve information flow for both forward and backward propagation, our networks are less difficult to train even though our networks are much deeper.

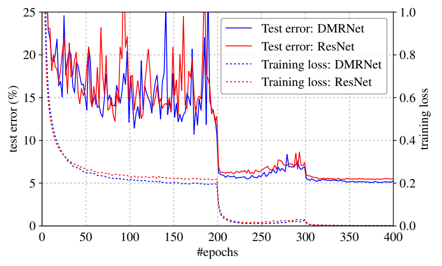
4.5. ImageNet Classification

We compare our DMRNet against the ResNet on the ImageNet 2012 classification dataset [4], which consists of 1000 classes of images. The models are trained on the 1.28 million training images, and evaluated on the $50,000$ validation images.

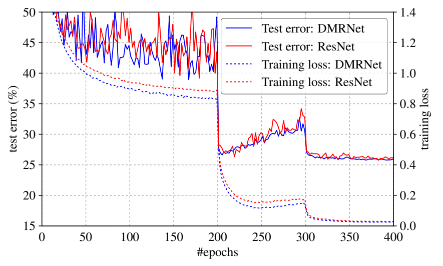
Network architecture. We compare the results of ResNet-101 with 101 layers. The ResNet-101 [7] (44.5M) is equipped with 4 stages of residual blocks with bottleneck layers, and the numbers of blocks in the four stages are $(3, 4, 23, 3)$, respectively. We form our DMRNet by replacing the residual blocks with our merge-and-run blocks and setting the numbers of blocks in the four stages to $(2, 2, 8, 2)$, resulting in our DMRNet with depth 44 (43.3M).

Optimization. We follow [7] and use SGD to train the two models using the same hyperparameters (weight decay $=$ 0.0001, and momentum $=$ 0.9) with [7]. The mini-batch size is 256, and we use 8 GPUs (32 samples per GPU). We adopt the same data augmentation as in [7]. We train the models for 100 epochs, and start from a learning rate of 0.1,

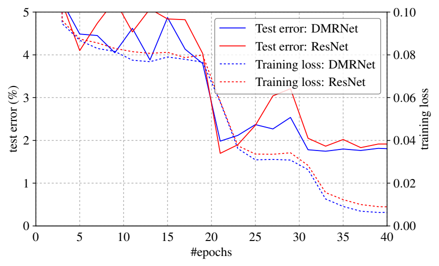
[6] Both DMRNets and DenseNets do not use bottlenecks. DenseNets with bottleneck layers would perform better. We will combine bottleneck layers into our approach as our future work to further improve the accuracy.



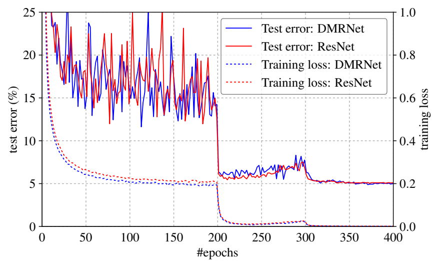
(a) CIFAR-10 ($L = 30$)



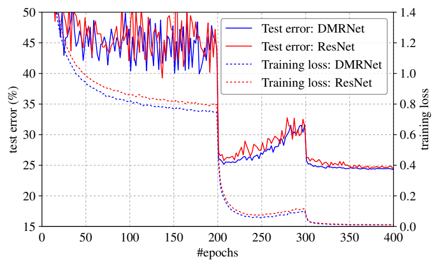
(b) CIFAR-100 ($L = 30$)



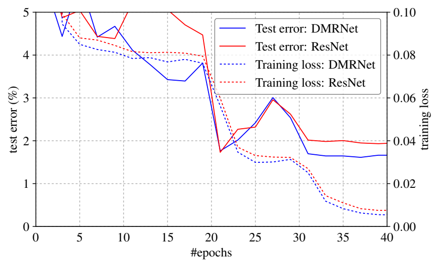
(c) SVHN ($L = 30$)



(d) CIFAR-10 ($L = 54$)



(e) CIFAR-100 ($L = 54$)



(f) SVHN ($L = 54$)

Figure 7. Comparing the optimization of ResNets and the DMRNets with the same number of layers/parameters. The vertical axis corresponds to training losses and testing errors, and the horizontal axis corresponds to #epochs.

Table 3. Classification error comparison with state-of-the-arts. The results of DenseNets are based on the networks without bottlenecks. The DMRNet-Wide is the wide version of a DMRNet, $4\times$ wider, i.e., the widths of the threes stages are 64, 128, and 256, respectively.

	Depth	Params.	CIFAR-10	CIFAR-100	SVHN
Network in Network [16]	-	-	8.81	-	2.35
All-CNN [25]	-	-	7.25	33.71	-
FitNet [21]	-	-	8.39	35.04	2.42
Deeply-Supervised Nets [15]	-	-	7.97	34.57	1.92
Swapout [23]	20	1.1M	6.85	25.86	-
	32	7.4M	4.76	22.72	-
Highway [27]	-	-	7.72	32.39	-
DFN [35]	50	3.7M	6.40	27.61	-
	50	3.9M	6.24	27.52	-
FractalNet [14]	21	38.6M	5.22	23.30	2.01
W/ dropout & droppath	21	38.6M	4.60	23.73	1.87
ResNet [7]	110	1.7M	6.61	-	-
ResNet [10]	110	1.7M	6.41	27.22	2.01
ResNet (pre-activation) [8]	164	1.7M	5.46	24.33	-
	1001	10.2M	4.62	22.71	-
ResNet W/ stochastic depth [10]	110	1.7M	5.23	24.58	1.75
	1202	10.2M	4.91	-	-
Wide ResNet [39]	16	11.0M	4.81	22.07	-
	28	36.5M	4.17	20.50	-
W/ dropout	16	2.7M	-	-	1.64
RiR [32]	18	10.3M	5.01	22.90	-
Multi-ResNet [1]	200	10.2M	4.35	20.42	-
	398	20.4M	3.92	-	-
DenseNet [9]	100	27.2M	3.74	19.25	1.59
DMRNet (ours)	56	1.7M	4.96	24.41	1.68
DMRNet-Wide (ours)	32	14.9M	3.94	**19.25**	**1.51**
DMRNet-Wide (ours)	50	24.8M	**3.57**	**19.00**	**1.55**

and then divide it by 10 every 30 epochs which are the same as the learning rate changing in [7]. We evaluate on the single 224×224 center crop from an image whose shorter side is 256.

Results. Table 4 shows the results of our approach and our

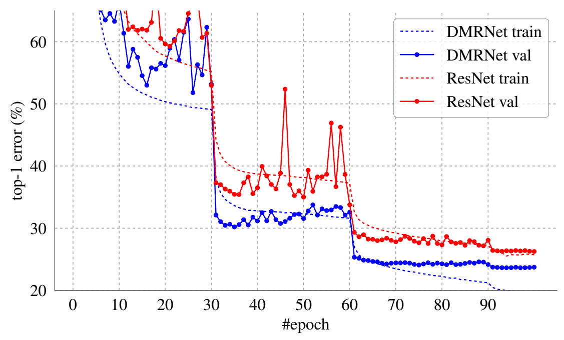


Figure 8. Training error and validation error curves of ResNet-101 (44.5M) and DFN-MR (43.3M) with the same optimization setting on ImageNet. We report the (top-1 error) results for training and single-crop validation. It can be observed that our approach performs better for both training errors and validation errors.

Table 4. The validation (single 224×224 center crop) and training errors (%) of ResNet-101 (44.5M) and our DMRNet (43.3M) on ImageNet.

	ResNet-101 [7]	ResNet-101	DMRNet
#parameters	44.5M		43.3M
Top-1 validation error	23.60	26.41	23.66
Top-5 validation error	7.10	8.50	**6.81**
Top-1 training error	17.00	25.75	19.72
Top-5 training error	-	8.12	6.59

MXNet implementation of ResNet-101, and the results of ResNet-101 from [7]. We can see that our approach performs the best in terms of top-5 validation error: our approach gets 1.7 gain, compared with the results of ResNet-101 from our implementation, and 0.3 gain compared with the result from [7].

The training and validation error curves of ResNet-101 and our DMRNet are given in Figure 8. It can be observed that our approach performs better for both training errors and validation errors, which also suggests that the gains are not from regularization but from richer representation. For example, the top-1 validation error of our approach is lower about 5% than that of the ResNet from the 30th epoch to the 55th epoch.

We notice that the results of our implemented ResNet on MXNet and the results from [7] are different. We want to point out that the settings are the *same* with [7]. We think that the difference might be from the MXNet platform, or there might be some other untested issues pointed by the authors of ResNets[7].

5. Discussions

Merge-and-run mappings for K branches. The merge-and-run mapping studied in this paper is about two residual branches. It can be easily extended to more (K) branches,

[7]https://github.com/KaimingHe/deep-residual-networks

Table 5. Comparison between merge-and-run mappings and identity mappings. Sharing = share the first conv. and the last FC.

		CIFAR-10		CIFAR-100	
	L	Identity	Merge-and-run	Identity	Merge-and-run
w/ sharing	48	5.21	**4.99**	25.31	**24.73**
	96	5.10	**4.84**	24.16	**23.98**
w/o sharing	48	4.67	**4.41**	23.96	**23.75**
	96	4.51	**4.37**	**22.23**	22.62

and accordingly merge-and-run mappings become a linear transformation where the corresponding transformation matrix is of $K \times K$ blocks, with each block being $\frac{1}{K}\mathbf{I}$.

Idempotent mappings. A merge-and-run mapping is a linear idempotent mapping, which is a linear transformation with the transformation matrix $\mathbf{M}$ being idempotent, $\mathbf{M}^n = \mathbf{M}$. Other idempotent mappings can also be applied to improve information flow. For examples, the identity matrix $\mathbf{I}$ is also idempotent and can be an alternative to the merge-and-run mappings. Compared with identity mappings, an additional advantage is that merge-and-run mappings introduce interactions between residual branches.

We conducted experiments using a simple idempotent mapping, $\mathbf{I}$, for which there is no interaction between the two residual branches and accordingly the resulting network consists of two ResNets that are separate except only sharing the first convolution layer and the last FC layer. We also compare the performances of the two schemes without sharing those two layers. The overall superior results of our approach, from Table 5, show that the interactions introduced by merge-and-run mappings are helpful.

Our merge-and-run mapping is complementary to other design patterns, such as dense connection in DenseNet [9], bottleneck design, and so on. As our future work, we will study the integration with other design patterns.

Deeper or wider. Numerous studies have been conducted on going deeper, learning very deep networks, even of depth 1000+. Our work can be regarded as a way to going wider and less deep, which is also discussed in [36, 39]. The manner of increasing the width in our approach is different from Inception [30], where the outputs of the branches are concatenated for *width increase* and then a convolution/pooling layer for each branch in the subsequent Inception block is conducted but for width decrease. Our merge-and-run mapping suggests a novel and cheap way of increasing the width.

6. Conclusions

In this paper, we propose deep merge-and-run neural networks, which improves residual networks by assembling residual branches in parallel with merge-and-run mappings for further reducing the training difficulty. The superior performance stems from several benefits: Information flow is improved, the paths are shorter, and the width is increased.

References

[1] M. Abdi and S. Nahavandi. Multi-residual networks. *CoRR*, abs/1609.05672, 2016. 1, 2, 7

[2] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015. 5

[3] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015. 2

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 6

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587, 2014. 1

[6] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 2, 5

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 3, 4, 5, 6, 7, 8

[8] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 2, 4, 7

[9] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. 2, 5, 7, 8

[10] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016. 2, 5, 7

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. 2

[12] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 5

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[14] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *CoRR*, abs/1605.07648, 2016. 2, 7

[15] C. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *AISTATS*, 2015. 5, 7

[16] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013. 2, 5, 7

[17] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1

[18] I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. 2

[19] D. Mishkin and J. Matas. All you need is a good init. *CoRR*, abs/1511.06422, 2015. 2

[20] M. Pezeshki, L. Fan, P. Brakel, A. C. Courville, and Y. Bengio. Deconstructing the ladder network architecture. In *ICML*, pages 2368–2376, 2016. 2

[21] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *CoRR*, abs/1412.6550, 2014. 7

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 2

[23] S. Singh, D. Hoiem, and D. A. Forsyth. Swapout: Learning an ensemble of deep architectures. *CoRR*, abs/1605.06465, 2016. 7

[24] L. N. Smith and N. Topin. Deep convolutional neural network design patterns. *CoRR*, abs/1611.00847, 2016. 2

[25] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014. 7

[26] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 2

[27] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *NIPS*, pages 2377–2385, 2015. 1, 2, 4, 7

[28] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, pages 1139–1147, 2013. 2

[29] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016. 2

[30] 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*, pages 1–9, 2015. 1, 2, 8

[31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 2

[32] S. Targ, D. Almeida, and K. Lyman. Resnet in resnet: Generalizing residual architectures. *CoRR*, abs/1603.08029, 2016. 1, 2, 7

[33] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):1958–1970, 2008. 5

[34] A. Veit, M. J. Wilber, and S. J. Belongie. Residual networks are exponential ensembles of relatively shallow networks. *CoRR*, abs/1605.06431, 2016. 1, 2

[35] J. Wang, Z. Wei, T. Zhang, and W. Zeng. Deeply-fused nets. *CoRR*, abs/1605.07716, 2016. 1, 2, 7

[36] Z. Wu, C. Shen, and A. van den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *CoRR*, abs/1611.10080, 2016. 8

[37] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. 2, 3

[38] S. Xie and Z. Tu. Holistically-nested edge detection. In *ICCV*, pages 1395–1403, 2015. 1

[39] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016. 1, 2, 7, 8

[40] K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, and T. Liu. Residual networks of residual networks: Multilevel residual networks. *CoRR*, abs/1608.02908, 2016. 1, 2