Deep Convolutional Neural Networks with Merge-and-Run Mappings

Liming Zhao[1], **Mingjie Li**[2], **Depu Meng**[2], **Xi Li**[1*], **Zhaoxiang Zhang**[3]
Yueting Zhuang[1], **Zhuowen Tu**[4], **Jingdong Wang**[5*]
[1] Zhejiang University
[2] University of Science and Technology of China
[3] Institute of Automation, Chinese Academy of Sciences
[4] UC San Diego
[5] Microsoft Research

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 with residual networks, our networks enjoy compelling advantages: they contain much shorter paths and the width, i.e., the number of channels, is increased, and the time complexity remains unchanged. 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., 3.06% testing error on CIFAR-10, 17.55% on CIFAR-100, 1.51% on SVHN) [1].

1 Introduction

Deep convolutional neural networks have been widely studied, and surprising performances have been achieved in many computer vision tasks, including object detection [Girshick *et al.*, 2014], semantic segmentation [Long *et al.*, 2015], edge detection [Xie and Tu, 2015], and so on.

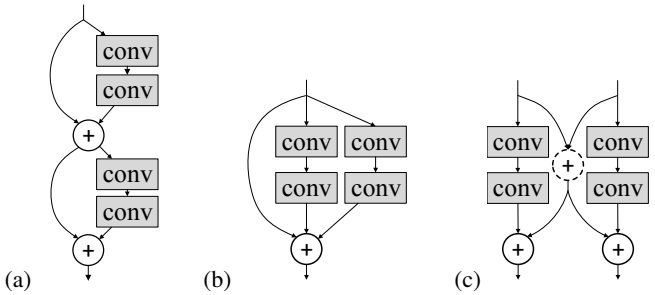

Figure 1: Illustrating the building blocks: (a) Two residual blocks; (b) A Vanilla-assembly block; (c) A merge-and-run block. (a) corresponds to two blocks in ResNets and assembles two residual branches sequentially while (b) and (c) both assemble the same two residual branches in parallel. (b) and (c) 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.

Residual networks (ResNets) [He *et al.*, 2016a] have been attracting a lot of attentions since it won the ImageNet challenge and various extensions have been studied [Zagoruyko and Komodakis, 2016; Zhang *et al.*, 2017a]. The basic unit is a residual block consisting of a residual branch and an identity mapping. Identity mappings introduce short *paths* from the input to the intermediate layers and then to the output layers [Veit *et al.*, 2016], and thus reduce the training difficulty.

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 (4 paths), (b) a Vanilla-assembly block (3 paths) and (c) a merge-and-run block (6 paths).

Obviously, the resulting network contains shorter paths as the parallel assembly of residual branches directly reduces

*Corresponding authors.
This work was done when Liming Zhao, Mingjie Li and Depu Meng were interns at MSR.
[1] https://github.com/zlmzju/fusenet

the network depth. We give a straightforward verification: the average length of two residual blocks is 2, while the average lengths of the corresponding Vanilla-assembly 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 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.

The experimental 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 with state-of-the-arts (e.g., 3.06% testing error on CIFAR-10, 17.55% on CIFAR-100, 1.51% on SVHN).

2 Related Works

Recently, network architecture design has been attracting a lot of attention. Highway networks [Srivastava *et al.*, 2015], residual networks [He *et al.*, 2016a], and GoogLeNet [Szegedy *et al.*, 2015] 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 easy. Ensemble view [Veit *et al.*, 2016] observes that ResNets behave like an exponential ensemble of relatively shallow networks, and shows that introducing short paths helps ResNets to avoid the vanishing gradient problem, which is similar to the analysis in deeply-fused networks [Wang *et al.*, 2016] and FractalNet [Larsson *et al.*, 2017].

The architecture of our approach is closely related to IGC [Zhang *et al.*, 2017b; Xie *et al.*, 2018a], Decoupled Convolution [Xie *et al.*, 2018b], Inception [Szegedy *et al.*, 2015], Xception [Chollet, 2017] and Inception-ResNet blocks [Szegedy *et al.*, 2017], multi-residual networks [Abdi and Nahavandi, 2016] and ResNeXt [Xie *et al.*, 2017], 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.

In comparison with contemporary work, ResNeXts [Xie *et al.*, 2017] also assemble residual branches in parallel, our approach [Zhao *et al.*, 2016] adopts parallel assembly to directly reduce the depth and does not modify residual branches, while ResNeXts [Xie *et al.*, 2017] transform a residual branch to many small residual branches. Compared with Inception [Szegedy *et al.*, 2015] and Inception-ResNet blocks [Szegedy *et al.*, 2017] that are highly customized, our approach requires less efforts to design and more flexible.

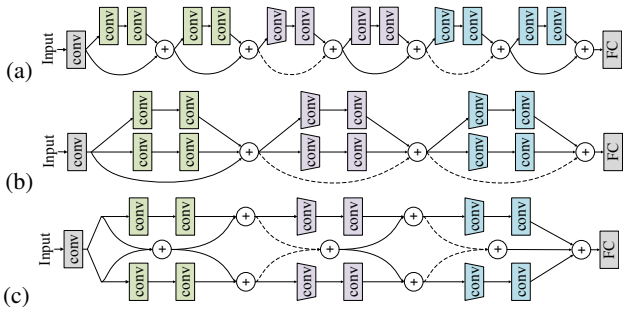

Figure 2: (a) a deep residual network; (b) a network built by stacking Vanilla-assembly blocks; (c) our deep merge-and-run neural network built by stacking merge-and-run blocks. The trapezoid shape indicates that down-sampling occurs in the corresponding layer, and the dashed line denotes a projection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

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 blocks.

The three building blocks are illustrated in Figure 1. Examples of the corresponding network structures, ResNets, DVANets (deep vanilla-assembly 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 t-th residual block. $H_t(\cdot)$ is a transition function, corresponding to the residual branch composed of a few stacked layers.

Vanilla-assembly 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 t-th Vanilla-assembly block. This structure resembles the building block in the concurrently-developed ResNeXt [Xie *et al.*, 2017], but the purposes are different: our purpose is to reduce the depth through assembling residual branches in parallel while the purpose of ResNeXt is to transform a single residual branch 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,

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

Table 1: Network architectures. Inside the brackets are the shape of the residual, Vanilla-assembly 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.

$$\mathbf{x}_{2(t+1)} = H_{2t}(\mathbf{x}_{2t}) + \frac{1}{2}(\mathbf{x}_{2t} + \mathbf{x}_{2t+1}),$$
$$\mathbf{x}_{2(t+1)+1} = H_{2t+1}(\mathbf{x}_{2t+1}) + \frac{1}{2}(\mathbf{x}_{2t} + \mathbf{x}_{2t+1}), \quad (3)$$

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 t-th block. There is a clear difference from Vanilla-assembly 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,

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

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 ($\frac{1}{2}$ in $\mathbf{M}$ is necessary). Thus, we have

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

where $t' < t$ corresponds to an earlier block. This shows that during the forward flow there are quick paths directly sending the input and the intermediate outputs to the later block. We have a similar conclusion for gradient back-propagation. Consequently, merge-and-run mappings can 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, DVANet 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.

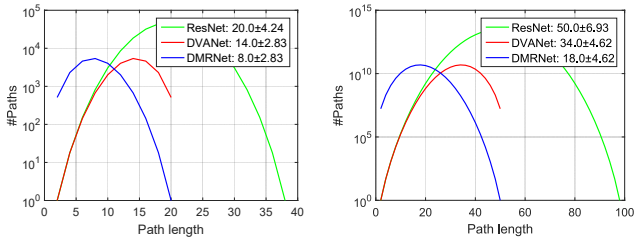

Figure 3: Comparing the distributions of the path lengths for three networks. Different networks: (avg length ± std). Left: $L = 9$. Right: $L = 24$.

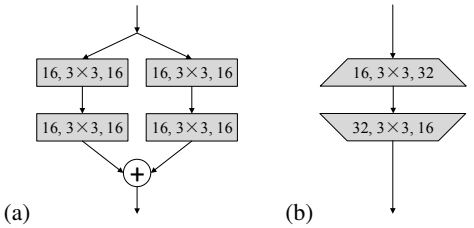

Figure 4: Illustrating the two residual branches shown in (a) are transformed to a single residual branch shown in (b). (a) All 4 convolutions are $(16, 3 \times 3, 16)$. (b) The 2 convolutions are $(16, 3 \times 3, 32)$ and $(32, 3 \times 3, 16)$, from narrow (16) to wide (32), and then from wide (32)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

It is shown in [He *et al.*, 2016a; Srivastava *et al.*, 2015] 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. 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 network potentially performs better.

Vanilla-assembly blocks are wider. We rewrite Equation 2 in a matrix form,

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

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 Vanilla-assembly block is wider. There are two layers in each branch. We have that the two residual branches are equivalent to a single residual branch 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.

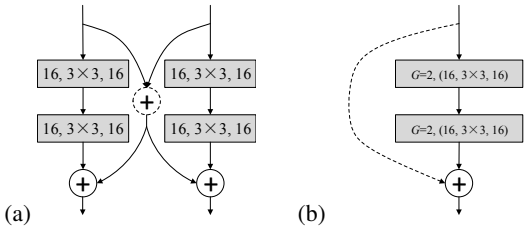

Figure 5: Transform the merge-and-run block shown in (a) to a two-branch block shown in (b). (b) The 2 convolutions are group convolutions. A group convolution contains two ($G = 2$) convolutions of $(16, 3 \times 3, 16)$: each receives a different 16-channel input and the two outputs are concatenated 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.

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 $2d$. The block is wider than a Vanilla-assembly block because the outputs of two residual branches in the merge-and-run block are separated and the outputs for the Vanilla-assembly 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. Besides, merge-and-run blocks haven't introduces other operations than Residual blocks and Vanilla-assembly blocks, which means the computing complexity remains unchanged.

Figure 5 shows that the merge-and-run block is transformed to a two-branch block. The dot line corresponds to the merge-and-run mapping, and 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 larger kernel, being a block-diagonal matrix with each block corresponding to the kernel of each partition in the group convolution.

4 Experiments

4.1 Datasets

CIFAR-10 and CIFAR-100. The two datasets are drawn from the 80-million tiny image database [Krizhevsky, 2009]. 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 use a standard data augmentation scheme widely adopted for these datasets [He *et al.*, 2016a; Huang *et al.*, 2017].

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. We use the same training strategy as [Lee *et al.*, 2015; Huang *et al.*, 2016].

4.2 Setup

Networks. As shown in Table 1, we follow ResNets to design our layers: use three stages of merge-and-run blocks with comparative number of channels to achieve the same parameter number with the opponent models in the following comparison, 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 the same output channels as the conv1_x, 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. 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 Figures 1 are between BN and ReLU.

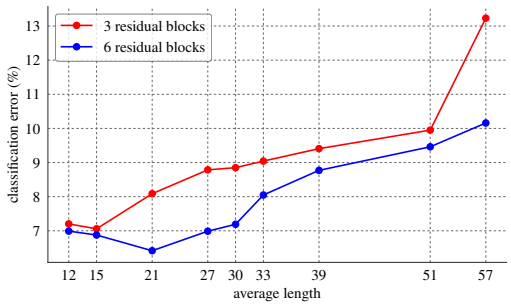

Figure 6: Illustrating how the testing errors of ResNets change as the average path length increases. The results are tested on CIFAR-10.

Training. We use SGD 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 [He *et al.*, 2016a], the weight decay is 0.0001, the momentum is 0.9, and the weights are initialized as in [He *et al.*, 2015]. Our implementation is based on MXNet [Chen *et al.*, 2015].

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 block), 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 and wide ResNets. We compare DVANets and DMRNets, and the baseline ResNets algorithm. They are formed with the same number of layers,

Params.	L	CIFAR-10				CIFAR-100				SVHN			
		ResNets	Wide-ResNets	DVANets	DMRNets	ResNets	Wide-ResNets	DVANets	DMRNets	ResNets	Wide-ResNets	DVANets	DMRNets
0.4M	12	6.62 ± 0.24	6.62 ± 0.20	6.53 ± 0.12	$\mathbf{6.48 \pm 0.04}$	29.69 ± 0.15	$\mathbf{29.59 \pm 0.31}$	29.75 ± 0.27	29.62 ± 0.08	$\mathbf{1.90 \pm 0.08}$	2.25 ± 0.03	2.13 ± 0.09	2.00 ± 0.04
0.6M	18	5.93 ± 0.17	6.12 ± 0.13	5.83 ± 0.09	$\mathbf{5.79 \pm 0.13}$	27.90 ± 0.26	27.95 ± 0.26	27.87 ± 0.22	$\mathbf{27.80 \pm 0.26}$	1.97 ± 0.09	2.10 ± 0.06	1.96 ± 0.10	$\mathbf{1.87 \pm 0.09}$
1.2M	36	5.35 ± 0.14	5.47 ± 0.18	5.26 ± 0.20	$\mathbf{5.18 \pm 0.20}$	26.00 ± 0.48	25.99 ± 0.28	25.98 ± 0.23	$\mathbf{25.41 \pm 0.19}$	1.90 ± 0.04	2.05 ± 0.02	1.81 ± 0.11	$\mathbf{1.77 \pm 0.11}$
1.5M	48	5.26 ± 0.09	5.55 ± 0.13	5.05 ± 0.20	$\mathbf{4.99 \pm 0.13}$	25.44 ± 0.20	25.38 ± 0.37	24.76 ± 0.33	$\mathbf{24.73 \pm 0.40}$	1.91 ± 0.03	2.08 ± 0.06	1.84 ± 0.06	$\mathbf{1.84 \pm 0.15}$

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

Depth	CIFAR-10		CIFAR-100	
	ResNeXt	DMRNeXt	ResNeXt	DMRNeXt
20	6.79 ± 0.38	$\mathbf{6.70 \pm 0.19}$	26.64 ± 0.48	$\mathbf{26.61 \pm 0.19}$
29	5.77 ± 0.22	$\mathbf{5.62 \pm 0.21}$	25.45 ± 0.18	$\mathbf{25.12 \pm 0.23}$
38	5.61 ± 0.19	$\mathbf{5.45 \pm 0.19}$	25.01 ± 0.36	$\mathbf{24.52 \pm 0.42}$

Table 3: Comparison of ResNeXt and our DMRNeXt with different depth. The results (mean $\pm$ std.) are reported from 5 runs.

and each block in a DVANet and a DMRNet corresponds to two residual blocks in a ResNet. Table 1 depicts the network structures. We also report the results of wide ResNets: Its depth is the same to DMRNet; It contains three stages similar to the ResNets as shown in Table 1, with the widths 23, 46 and 92 (slightly more parameters than DMRNets).

The comparison on CIFAR-10 is given in Table 2. One can see that compared with ResNets, DVANets and DMRNets consistently perform better, and DMRNets perform the best. The superiority of DVANets over ResNets stems from the less long paths and greater width. The additional advantages of a DMRNet are much greater width than a DVANet. Compared with wide ResNets with the depth same to ours which increases the width by adding more channels, our approach performs better.

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

Combination with ResNeXt. We study one kind of merge-and-run building block, where the parallel branch is a ResNeXt block. The ResNeXt approach [Xie *et al.*, 2017] transforms the bottleneck branch into a set of K bottleneck branches which are aggregated into a ResNeXt building block. We exploit such a transformation, but aggregate the bottleneck branches into two parallel branches, where each branch is a ResNeXt building block formed by $\frac{K}{2}$ transformed bottleneck branches. The two parallel branches are then equipped with the merge-and-run mapping, forming a merge-and-run block. We compare ResNeXt and our network, named DMRNeXt, by stacking the same numbers of building blocks.

The comparisons on CIFAR-10 and CIFAR-100 are presented in Table 3. We observe that our approach consistently outperforms ResNeXt on both CIFAR-10 and CIFAR-100. This demonstrates the effectiveness of our approach even in the case the parallel branch is a ResNeXt building block.

Depth	CIFAR-10		CIFAR-100	
	Xception	DMRNets	Xception	DMRNets
14	8.16 ± 0.54	$\mathbf{8.00 \pm 0.18}$	30.84 ± 0.44	$\mathbf{30.40 \pm 0.09}$
20	7.87 ± 0.14	$\mathbf{7.83 \pm 0.2}$	30.58 ± 0.98	$\mathbf{29.71 \pm 0.20}$
38	7.85 ± 0.6	$\mathbf{7.77 \pm 0.08}$	30.39 ± 0.94	$\mathbf{29.52 \pm 0.31}$

Table 4: Comparison between Xception, and DMRNets where the parallel branch contains two Xception blocks. The results (mean $\pm$ std.) are reported from 5 runs.

	Depth	Params.	CIFAR-10	CIFAR-100	SVHN
Swapout [Singh *et al.*, 2016]	20	1.1M	6.85	25.86	-
	32	7.4M	4.76	22.72	-
DFN [Wang *et al.*, 2016]	50	3.7M	6.40	27.61	-
	50	3.9M	6.24	27.52	-
FractalNet [Larsson *et al.*, 2017]	21	38.6M	5.22	23.30	2.01
W/ dropout & droppath	21	38.6M	4.60	23.73	1.87
ResNet [He *et al.*, 2016a]	110	1.7M	6.61	-	-
ResNet [Huang *et al.*, 2016]	110	1.7M	6.41	27.22	2.01
ResNet (pre-act) [He *et al.*, 2016b]	164	1.7M	5.46	24.33	-
	1001	10.2M	4.62	22.71	-
ResNet stochastic depth	110	1.7M	5.23	24.58	1.75
[Huang *et al.*, 2016]	1202	10.2M	4.91	-	-
Wide ResNet	16	11.0M	4.81	22.07	-
[Zagoruyko and Komodakis, 2016]	28	36.5M	4.17	20.50	-
W/ dropout	16	2.7M	-	-	1.64
DenseNet [Huang *et al.*, 2017]	100	27.2M	3.74	19.25	1.59
DenseNet-BC ($k = 24$)	250	15.3M	3.62	17.6	1.74
DenseNet-BC ($k = 40$)	190	25.6M	3.46	17.18	-
ResNeXt-29 [Xie *et al.*, 2017]	29	34.4M	3.65	17.77	-
PyramidNet [Han *et al.*, 2017]	272	26.0M	3.31	16.35	-
IGC-L450M2 [Zhang *et al.*, 2017b]	20	19.3M	3.25	19.25	-
IGC-L32M26 [Zhang *et al.*, 2017b]	20	24.1M	3.31	18.75	1.56
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
DMRNeXt (ours)	29	26.7M	3.06	17.55	-

Table 5: Classification error comparison with state-of-the-arts. 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.

Combination with Xception. We study our merge-and-run building block with two branches, where each branch contains two Xception blocks [Chollet, 2017]. We build the Xception network with 3 stages by stacking (e.g., M) Xception blocks, where there is an identity connection for two Xception blocks. and our DMRNets with 3 stages by stacking $\frac{M}{2}$ blocks so that the depths are the same.

The widths of the three stages are set as 88, 176, and 352 for our net and 64, 128, and 256 for Xception, so that the parameter complexities are also the same.

The comparisons on CIFAR-10 and CIFAR-100 are presented in Table 4. We can see that our approach consistently outperforms better both on CIFAR-10 and CIFAR-100, which shows the effectiveness of our approach in the case the parallel branch is an Xception building block.

4.4 Comparison with State-of-the-Arts

The comparison is reported in Table 5. We report the results of DMRNets and wide DMRNets (denoted by DMRNet-

	ResNet-98	DMRNet-50
#parameters	45.0M	46.4M
Top-1 validation error	23.38	**23.16**
Top-5 validation error	6.79	**6.64**
Top-1 training error	15.09	**14.46**
Top-5 training error	3.25	**3.16**

Table 6: The validation (single 224×224 center crop) and training errors (%) of ResNet-98 (45.0M) and our DMRNet-50 (46.4M) on ImageNet.

Wide), $4\times$ wider, i.e., the widths of the threes stages are 64, 128, and 256, respectively. Refer to Table 1 for network architecture descriptions. We also report the results of the merge-and-run block with the branch formed by 2 ResNeXt building blocks, denoted by DMRNeXt, transformed by ResNeXt-29 ($6 \times 64d$) (see Section 4.3). The best, second-best and the third-best accuracies are highlighted in red, green and blue.

One can see that our DMRNeXt outperforms existing state-of-the-art results and achieves the best results on CIFAR-10. and that achieves the third-best results on CIFAR-100. Compared with the ResNeXt-29 ($8 \times 64d$), the improvement is very signification on CIFAR-10 and CIFAR-100 with smaller parameter complexity. DMRNet-Wide (depth $= 32$) is very competitive: outperform all existing state-of-the-art results on SVHN. It contains only 14.9M parameters, only two third of the parameters (24.1M) of the competitive IGC-L32M26. 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 on all the three datasets. This is because *merge-and-run mappings* improve information flow for both forward and backward propagation and are less difficult to train even though our networks are much deeper.

We also compare our DMRNet-50 against the ResNet-98 with the same experimental settings on the ImageNet 2012 classification dataset [Deng *et al.*, 2009]. We train models for 95 epochs with extra 25 epochs for retraining on MXNet [Chen *et al.*, 2015]. The training and validation errors of ResNet-98 and our DMRNet-50 are given in Table 6. It can be observed that our approach performs better for both training and validation errors, which also suggests that the gains are not from regularization but from richer representation.

5 Discussions

Merge-and-run mappings for K branches. The merge-and-run mapping studied in this paper is the case that contains two residual branches. It can be easily extended to more (K) branches, 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 $\frac{1}{K}\mathbf{I}$.

We empirically study how K affects the performance by conducting two experiments: (i) fix the depth as 50, and adjust the width of each branch: $(23, 46, 92)$, $(16, 32, 64)$, $(12, 24, 48)$, and $(8, 16, 32)$ for $K = 1$, 2, 4, and 8, guaranteeing the same number of parameters ; and (ii) fix the width of each branch: $(16, 32, 64)$ for the three stages, and adjust the depth: 98, 50, 26, and 14 for $K = 1$, 2, 4, and 8, guaranteeing the same number of parameters. The results are given in Figure 7, which empirically suggests to choose $K = 2$.

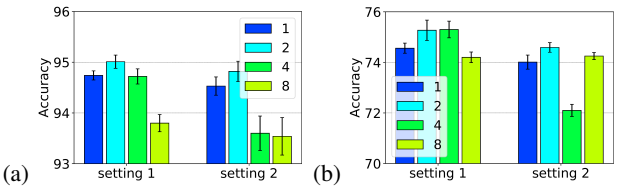

Figure 7: The effect of #branches with the same #parameters for the same depth (setting 1) or the same branch width (setting 2): (a) CIFAR-10, and (b) CIFAR-100. It can be seen that fewer or more branches do not lead to better performance and that 2 branches result in overall good performance.

		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

Table 7: Comparison between merge-and-run mappings and identity mappings. Sharing = share the first conv. and the last FC.

Idempotent mappings. A merge-and-run mapping is a linear idempotent mapping, and 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 identity 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 7, show that the interactions introduced by merge-and-run mappings are helpful.

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 [Wu *et al.*, 2016; Zagoruyko and Komodakis, 2016]. The manner of increasing the width in our method is different from Inception [Szegedy *et al.*, 2015], 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 improve residual networks by assembling residual branches in parallel with merge-and-run mappings for easing the training without the introduction of extra model and time complexities. The superior performance stems from several factors: information flow is improved; the paths are shorter; the width is increased.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grants (U1509206, 61472353, and 61751209), in part by the National Basic Research Program of China under Grant Grant 2015CB352302, and partially funded by the MOE-Microsoft Key Laboratory of Visual Perception, Zhejiang University.

References

[Abdi and Nahavandi, 2016] Masoud Abdi and Saeid Nahavandi. Multi-residual networks. *CoRR*, abs/1609.05672, 2016.

[Chen *et al.*, 2015] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[Chollet, 2017] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.

[Deng *et al.*, 2009] 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.

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

[Han *et al.*, 2017] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *CVPR*, 2017.

[He *et al.*, 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

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

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

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

[Huang *et al.*, 2017] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[Krizhevsky, 2009] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[Larsson *et al.*, 2017] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*, 2017.

[Lee *et al.*, 2015] Chen-Yu Lee, Saining Xie, Patrick W. Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *AISTATS*, 2015.

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

[Singh *et al.*, 2016] Saurabh Singh, Derek Hoiem, and David A. Forsyth. Swapout: Learning an ensemble of deep architectures. In *NIPS*, 2016.

[Srivastava *et al.*, 2015] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *NIPS*, 2015.

[Szegedy *et al.*, 2015] 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.

[Szegedy *et al.*, 2017] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017.

[Veit *et al.*, 2016] Andreas Veit, Michael J. Wilber, and Serge J. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NIPS*, 2016.

[Wang *et al.*, 2016] Jingdong Wang, Zhen Wei, Ting Zhang, and Wenjun Zeng. Deeply-fused nets. *CoRR*, abs/1605.07716, 2016.

[Wu *et al.*, 2016] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *CoRR*, abs/1611.10080, 2016.

[Xie and Tu, 2015] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *ICCV*, 2015.

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

[Xie *et al.*, 2018a] Guotian Xie, Jingdong Wang, Ting Zhang, Jianhuang Lai, Richang Hong, and Guo-Jun Qi. Igcv2: Interleaved structured sparse convolutional neural networks. In *CVPR*, 2018.

[Xie *et al.*, 2018b] Guotian Xie, Ting Zhang, Kuiyuan Yang, Jianhuang Lai, and Jingdong Wang. Decoupled convolutions for cnns. In *AAAI*, 2018.

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

[Zhang *et al.*, 2017a] K. Zhang, M. Sun, X. Han, X. Yuan, L. Guo, and T. Liu. Residual networks of residual networks: Multilevel residual networks. *IEEE Trans. Cir. and Sys. for Video Technol.*, pages 1–1, 2017.

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

[Zhao *et al.*, 2016] Liming Zhao, Jingdong Wang, Xi Li, Zhuowen Tu, and Wenjun Zeng. On the connection of deep fusion to ensembling. *CoRR*, abs/1611.07718, 2016.